GeoSparkSim: A Scalable Microscopic Road Network Traffic Simulator Based on Apache Spark

by

Zishan Fu

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved April 2019 by the
Graduate Supervisory Committee:

Mohamed Sarwat Abdelghany Aly Elsayed, Chair
Giulia Pedrielli
Jorge A. Sefair

ARIZONA STATE UNIVERSITY

May 2019

ABSTRACT

Researchers and practitioners have widely studied road network traffic data in different areas such as urban planning, traffic prediction and spatial-temporal databases. For instance, researchers use such data to evaluate the impact of road network changes. Unfortunately, collecting large-scale high-quality urban traffic data requires tremendous efforts because participating vehicles must install Global Positioning System(GPS) receivers and administrators must continuously monitor these devices. There have been some urban traffic simulators trying to generate such data with different features. However, they suffer from two critical issues (1) Scalability: most of them only offer single-machine solution which is not adequate to produce large-scale data. Some simulators can generate traffic in parallel but do not well balance the load among machines in a cluster. (2) Granularity: many simulators do not consider microscopic traffic situations including traffic lights, lane changing, car following. This paper proposed GeoSparkSim, a scalable traffic simulator which extends Apache Spark to generate large-scale road network traffic datasets with microscopic traffic simulation. The proposed system seamlessly integrates with a Spark-based spatial data management system, GeoSpark, to deliver a holistic approach that allows data scientists to simulate, analyze and visualize large-scale urban traffic data. To implement microscopic traffic models, GeoSparkSim employs a simulation-aware vehicle partitioning method to partition vehicles among different machines such that each machine has a balanced workload. The experimental analysis shows that GeoSparkSim can simulate the movements of 200 thousand cars over an extensive road network (250 thousand road junctions and 300 thousand road segments).

ACKNOWLEDGMENTS

I want to thank my advisor Dr. Mohamed Sarwat in particular for providing me the great opportunity working with him and his lab members. He is very knowledgeable and gives me endless guidance, suggestions and supports through my thesis. Besides my advisor, I would like to thank my committee members Dr. Giulia Pedrielli and Dr. Jorge Sefair for their great supports, professional suggestions and invaluable advice. I am also grateful to all the members in the Data Systems Lab for their insights and comments to my thesis. Mainly, I would like to express my gratitude to Jia Yu. He gives me many great suggestions and ideas that help me further improve the system. Also, I would like to thank all my friends, especially Osama Khalid. He gives me many feedbacks to my thesis. Finally, I would like to give my deepest gratitude to my family. Without their love and support, I won't get the chance to finish the thesis and they are my rock in spirit.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Chapter 1

PROBLEM OVERVIEW

1.1 Introduction

In modern life, we have to face many troubles one of which is traffic congestion becoming more serious day after day. Traffic congestion can be attributed to a number of factors, including the high volume of vehicles on the road, irrational urban sprawl and development. Irrational urban sprawl means the unrestricted growth of housing, commercial development and road street. There are many studies widely on traffic issues and researchers have been trying to solve these problems in past century Greenshields *et al.* (1935). The first step is to understand physical road network traffic data. Traffic data gives researchers information about how travel speed in a particular street change over time which is critical to road network related analysis because these data affect travel times and thus the relevant results. Urban road network analysis includes route planning, urban planning, closest facility, and accessibility. Traffic reflects a significant impact on time-dependent road network analysis. For example, if a person plans the route without taking account of traffic, the expected journey time may be far from accurate. Moreover, he might lose the chance to schedule an efficient path with less travel time by avoiding the slower, more congested roads. GeoSparkSim is a system that can generate traffic data and many related fields will take advantages of the traffic predictions, for example, site evaluation, transportation system, route planning, autonomous car, and data mining. The following paragraphs will discuss it and study some cases in each field.

1.2 Scenarios

Site Evaluation. Site evaluation gives a review of the characteristics and properties of the building or construction and thus improves the site selections. Travel costs have been an essential factor for site evaluation. It's necessary to compute the dynamic traffic changes to examine the impact of travel costs on-site evaluation. For example, an investor may build a new restaurant having minimum distance with the targeted customers. Nevertheless, traffic density is another critical factor to make the site assessment. If the traffic density in a specific area is higher than surrounding regions, the site in this area will have more chance for exposure, but also may suffer from traffic congestion and higher rent cost problem. In other words, if the density is quite low in a particular area, the land prices may be relatively more economical and have a better commute. The investor may have many location options, and he may evaluate the traffic volume in a particular area because traffic enhances the exposing opportunities for customers and bring possible business. Additionally, the site also makes an impact on traffic. For instance, if the government wants to build a light rail or add a new subway line in the road network, they evaluate the current traffic flow and decide the best route. In the construction stage, they may analysis traffic over time and choose the best time to install the infrastructures to minimize the impact on current traffic. With GeoSparkSim, the predicted traffic data helps them make a better decision.

Transportation System. The transportation system includes all the equipment and logistics of transporting objects which are aimed at coordinating the movement of pedestrian, vehicles, and goods to utilize routes most efficiently and safely. Traffic data is considered as the key to designing collaborated transportation rules. For example, customizing traffic signals duration to control and enlarge the traffic throughput.

Besides, traffic data is used to create a model and detect the accident. An accident happens when the reaction distance and responses time is not in a safe range. For example, a truck driver takes brake without having enough distance towards intersection which is very dangerous. The traffic data generated from GeoSparkSim incorporates many traffic models, and GeoSparkSim tries to match it with real GPS data. These generated data helps design a more reliable transportation system.

Route Planning. Route planning defines the optimal path between the source and the destination location. The optimal path can be the route with the shortest distance or the shortest journey time and Section 5.2 will have a more detailed introduction. Traffic data supports the improvements on route planning algorithms, like the shortest path algorithm in Google Map. When you enter your source and destination location, Google Map weights the live traffics and offers alternatives for you. GeoSparkSim helps the developer to implement, test and debug the algorithm.

Data mining. Data mining is a process to find the latent patterns and features behind the data which makes use of a range of techniques like machine learning, statistics, database management system, etc. Researchers and scientist applied data mining technologies in traffic data, such as traffic flow forecasting. The versatile traffic data make benefits on innovative application across different subjects and fields. For example, GasBuddy (2019), a web-based company which lets users search for gas price, makes suggestions for the cheapest gas station nearby. With predicted traffic data, we could improve the gas station finding algorithm to return results with less journey time. Getting the most accurate results for the driver is critical because it helps them to avoid traffic congestion and fill the tank quickly. GeoSparkSim generates traffic data to make more possibilities in novel technologies and emerging fields.

Autonomous Car Testing and Planning. The driverless vehicles with radars or computer vision technologies could automatically avoid surrounding vehicles and the researches indicate countless benefits and significant impacts from these vehicles Litman (2017). One of the provisions in the future is to build a world with all cars without drivers and there are no traffic lights at roads. The researchers from MIT proposed such a smart city for autonomous vehicles MIT (nd). Autonomous car study traffic rules and acts all driving behaviors without having control from the driver. In the process, traffic is a significant part to accommodate autonomous cars' driving behaviors. GeoSparkSim can simulate growing number of autonomous vehicles and collect the trajectories with full driving events. This simulation and collection play an essential role in validation before releasing the car.

1.3 Problems and Solutions

Road network traffic data contains the trajectories of a set of vehicles moving over time. Each path consists of a sequence of Global Positioning System (GPS) coordinates which capture the vehicle locations at every audited time step. These data are classified as historical and live traffic. Historical traffic is the stored data collected by research or scientist, while live traffic is generated from real-time GPS. Unfortunately, although there are millions of vehicles driving in big cities, collecting large-scale high-quality live traffic data requires tremendous efforts since participating vehicles must install GPS receivers and administrators must continuously monitor these devices. Researchers from Microsoft Research spent more than five years on collecting 17621 trajectories over 182 volunteers Zheng *et al.* (2010). Even we get these data, this kind of historical data has many constraints, such as location, volume, and quality. For example, New York taxi trip data NYCTraffic (2018) is a dataset of records including pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances,

itemized fares, rate types, payment types, and driver-reported passenger counts. The dataset is focusing on New York location, taxi type of vehicle and taxi status. In order to get complete mobility, researchers need extraordinary efforts to clean and process it. The disciplines mentioned above are also affected by these problems. For example, it is hard to get a long period of traffic data for site planning and transportation system. Data mining, route planning, and autonomous car planning require a large-scale amount of data to build the model and improve the algorithm.

To remedy that, researchers turn to traffic data simulators which can generate massive synthetic road network traffic data. Traffic simulator is the mathematical modeling of transportation system building on the application of computer software where researchers have spent many years in studying traffic simulators. Chapter 2 is a literature review of existing simulators. These mathematical traffic models are related to many facts which are very computation-intensive. GeoSparkSim is an extensible microscopic traffic simulator to handle these problems that can generate a large amount of traffics in a short period.

Giving this outlook, this rest of this thesis is presented as follows: Chapter 2 studies the related works. An overview and user documents of GeoSparkSim is given in Chapter 3. Chapter 4 introduces road network graph. Chapter 5 demonstrate vehicles generations and VehicleRDD. The vehicle partitioning method and microscopic simulation details are explained in Chapter 6. A comprehensive experimental analysis is given in Chapter 7. Chapter 8 concludes the paper.

Chapter 2

LITERATURE REVIEW

There are many existing traffic simulators with different features. This chapter attempts to give a brief overview of the state-of-art in traffic simulators and lastly provides literature with the comparison.

2.1 Simulators

Traffic simulators are classified as macroscopic and microscopic. A microscopic simulator is targeted at the detailed mobility patterns of individual vehicles. On the other hand, the macroscopic traffic simulator focuses on the whole vehicular traffic flow without taking into account the current traffic pressures, vehicle distributions and road network constraints, such as traffic lights, one road, crossroad, etc. There are many classic traffic simulators proposed in the past two decades, such as BerlinMOD Düntgen *et al.* (2009), Brinkhoff Brinkhoff (2002), SUMO Krajzewicz *et al.* (2002), TRANSIMS Nagel and Rickert (2001), MATSim Waraich *et al.* (2009), Vissim Vissim (2019), ParamGrid Klefstad *et al.* (2005) and Smarts Ramamohanarao *et al.* (2017). This chapter will discuss relevant terminologies and these simulators in details.

Macroscopic traffic simulator. Simulators in this category focus on general vehicular flow in the transportation road network. All vehicles drive similarly and merely move from the sources to the destinations step by step. Brinkhoff proposed a simulator Brinkhoff (2002) that generates moving objects for every single road segment in a simulation period. BerlinMOD Düntgen *et al.* (2009) is a popular moving object benchmark including a set of queries and a data generator which is able to generate road network traffic data for a number of identifiable vehicles. MNTG Mok-

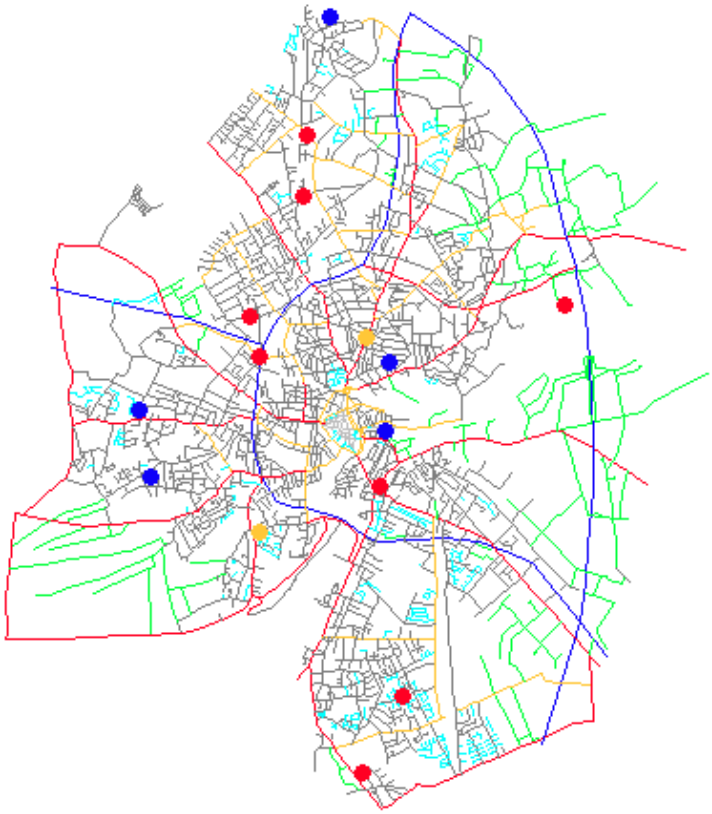


Figure 2.1: Brinkhoff Moving Objects Visualization

bel *et al.* (2013) develops a wrapper of Brinkhoff framework and BerlinMOD and provides a web service with a user-friendly and more accessible interface. Macroscopic simulators can quickly yield a massive amount of data because they are less computation-intensive. But the produced data may not be very useful because the vehicles do not follow real traffic rules and there are many vehicle collisions (e.g., vehicles have the same GPS locations). Figure 2.1 is Brinkhoff moving objects visualization which is a typical example of a macroscopic simulator.

Microscopic traffic simulator. Compare to macroscopic simulators, microscopic traffic simulators pay more attention to the detailed mobility of each vehicle and takes into account many different driving behaviors including lane changing, car

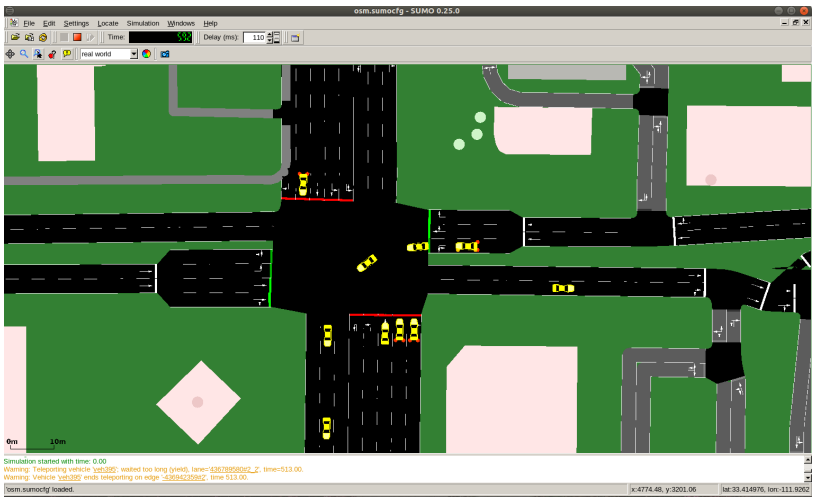


Figure 2.2: Sumo Traffic Visualization

Intersection between South Rural Rd and East Apache Blvd

following, traffic signals and different traffic rules. SUMO Krajzewicz *et al.* (2002) is one of the most popular open source microscopic simulators. It supports many microscopic traffic models such as lane changing, different right-of-way rules, and traffic lights. Besides that, it also provides the user with opportunities to customize simulation data for various objects, such as vehicles, pedestrian, bicycles and railway. Microscopic traffic simulators are too computation-intensive because the driving behavior of a vehicle is affected not only by its specified status but also its surroundings like nearby vehicles and traffic lights. For example, to simulate the next location of a vehicle, the simulator needs to check whether it is in a safe distance to other vehicles. Therefore, although microscopic simulators can generate realistic data, they suffer from the scalability issue. Figure 2.2 is SUMO traffic visualization which is an example of microscopic traffic simulator.

2.2 Related Works

2.2.1 Brinkhoff

Professor Thomas Brinkhoff designed a framework for generating network-based moving objects (Brinkhoff). Network-based moving objects generation means the object distribution is correlated to the density of the network. He adopts uniform distribution to initiate moving object, computing the fastest path and provide the interactive visualization interface for results.

- **Motion Computing.** The lifetime of a moving object is defined from starting, moving by trajectories, arriving and die. The moving object born at a certain point in the road network, change location step by step following identical trajectories and disappear after arriving the last coordinate in trajectory set. By increment the trajectory index, the object moves from time t to t+1. Professor Brinkoff proposed three approaches to generate the start node of a moving object. First is the data-space oriented approach(DSO) which computes a coordinate (x, y) by using a two-dimensional distribution function and do map matching to find the nearest node in the road network. Second is the region-based approach(RB). Each cell has a value describing the probability of the cell to adapt the object into the network and the starting point is based on the value. Last is the network-based approach(NB) adopting in Brinkhoff traffic generator. The distribution of starting nodes is associated with the road network distribution. After that, the generator randomly generates the travel length and compute the destination by the created starting node and the randomized length.

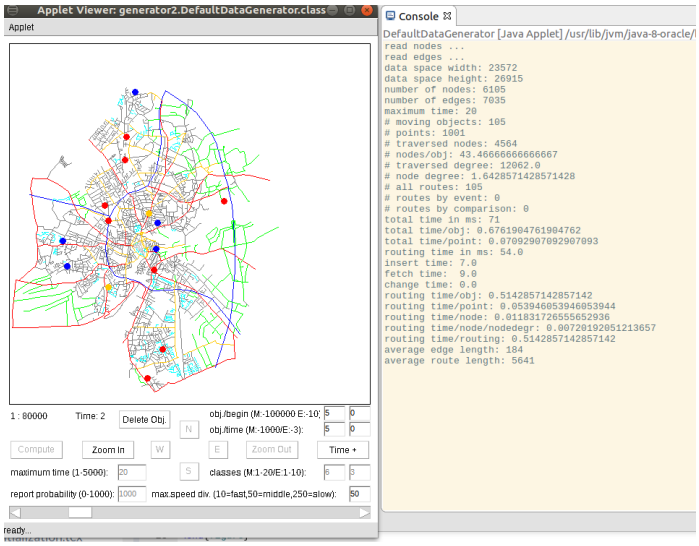


Figure 2.3: Brinkhoff UI Snapshot

- **Framework summary.** The input of Brinkhoff framework is the road network data and some relevant parameters. After triggering generation, source-destination pair will be generated by the motion computing algorithm mentioned in the last paragraph and the shortest path between source and destination computed by Dijkstra and A star pathfinder algorithm. The trajectories will be collected and reported in text or database by Java Database Connectivity (JDBC). These results could be read from the user interface and visualization the mobility in a certain road network. Figure 2.3 is Brinkhoff Moving object UI and the right side is the simulation summary on the console. Brinkhoff moving objects generator is developed in Java and the user can install it in any platform. Brinkhoff generator preprocesses road network to a compatible format, such as edge and node. After imported the road network data, Brinkhoff computes the moving objects and show it in GUI. User can modify the configuration file and the parameters in UI to make personal results.

2.2.2 BerlinMOD

BerlinMOD is a benchmark for moving object databases based on SECONDO DBMS Guting *et al.* (2005), an extensible DBMS architecture and prototype for generating data. BerlinMOD used the approaches from Brinkhoff to create the start and destination node. Travel assumptions are used to clarify the movement of object. For instance, a person leaves home node at 8 am + T1, drive to work node, stay there until 4 pm + T2 and then returns to the home node. T1 and T2 is the variable to define the variation of moving object time frame.

- **Trip Generation.** The input is the start node, destination node and travel starting time. BerlinMOD uses well-known shortest path to compute the route like Dijkstra, and loop all the ongoing visiting node in the road. If the moving object not arrive at the next node in path and the rest distance to the node is larger than 50 meters, apply an acceleration event to the trip when current speed less than the maximum speed, randomly choose either deceleration event or stop activity depending on the distance when current driving speed is larger than speed limits. Otherwise, reducing the velocity. Object move 5 meters each step and apply the possible events in moving steps.

- **Benchmarking Queries.** BerlinMOD provides two sets of queries for benchmark the moving objects. Data model and operations are predefined and the user can do SQL-like queries for moving objects. For example, select distinct LL. License as Licence, C.Model as Model from dataScar C, QueryLicences LL where C.Licence = LL.License. The query returns the license and model of a moving object when the object's license is equal to the license in the license table. The queries include range and point queries, relations queries and nearest neighbor query. Range and point queries perform the necessary operations

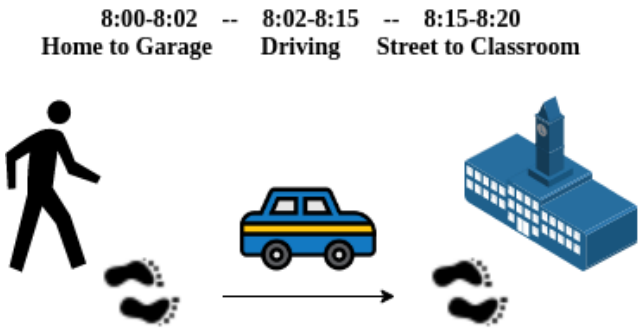


Figure 2.4: Multimodality

to objects, like object identity, dimension, interval, condition, and aggregation. Relations kind of query discovers the relationship between the query elements. NN queries observe the historical moving objects and return distance semantics.

2.2.3 SUMO

Simulation of urban mobility(SUMO) Krajzewicz *et al.* (2002) is an open source continuous, microscopic and multi-modal traffic simulation package developed by the Institute of Transportation Systems at the German Aerospace Center. Multi-modal means the movement model includes not only a car but also pedestrian, public transportation system, etc. For example, a person may leave at home on feet, travel by car, left the main street and walk to class. Figure 2.4 is an example of the compound route.

- **Components.** The car-following model used in SUMO is Gipps-model Wiki Gipps' model (2019) which can display the traffic features. This kind of car driver model moderates the safe velocity and helps to avoid collisions. Traffic lights are deployed in SUMO and plays a vital role in traffic management. SUMO developed a network converter to convert various road network data

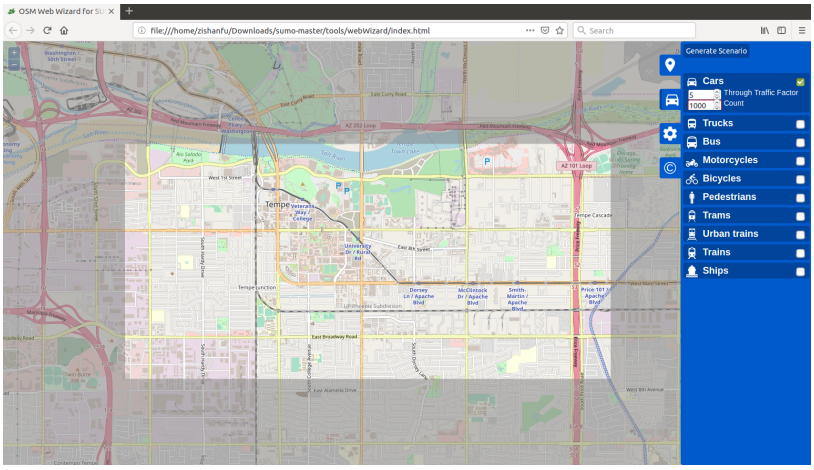


Figure 2.5: SUMO Front-end Interface

into XML-description. Dijkstra shortest path algorithm is used to compute the route.

- **Usability.** SUMO supports different types of moving objects, such as a car, pedestrian, train, subway, ship, etc. By inputting the types of moving objects, simulation numbers and region-rectangle, a visual simulation interface is provided by SUMO. The user could customize data and explore the insights behind the simulation results. Figure 2.5 is a web front-end for SUMO and 2.6 is the simulation GUI and console results. By selecting an arbitrary rectangle in the front-end interface and entering the required parameters, SUMO prepares the road network and simulation results. When it gets ready, the simulation GUI will open and begin to simulate effects.

2.2.4 TRANSIMS

Parallel implementation of the TRansportation ANalysis and SIMulation System (TRANSIMS) is a traffic micro-simulator. TRANSIMS parallelize simulation process by design strategies to utilize multi-threads which called domain decomposition. For

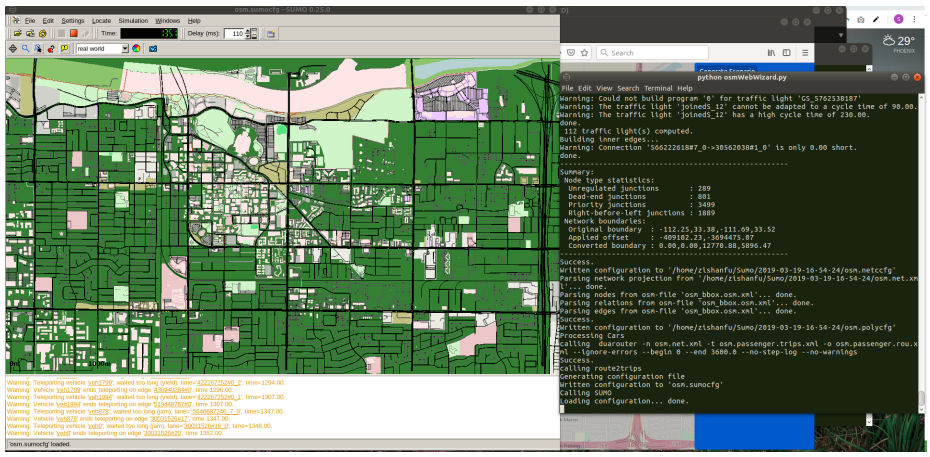


Figure 2.6: SUMO Traffic Simulation GUI

example, a road network graph is partitioned to several domains as the number of CPUs and each CPU simulates the traffic on its domain. Moreover, TRANSIMS proposed the methods to minimize message passing cost and achieve load balancing.

- **Domain Decomposition.** Domain decomposition is a process to cut the geographical region into several similar size domains. TRANSIMS cuts the network streets in the middle of the links rather than intersections. Each CPU computes the local simulation from time t to t+1 and CPUs communicate and exchange the boundaries messages for time step t, then update locally.

- **Graph Partitioning.** Three factors need to be considered to make an efficient graph partitioning. First is to minimize the number of split links. Because more links require more computing capability to perform partition. Second, the number of domains each CPU shares links to should be minimized. After each local simulation, a simulator is asked for exchange messages. Less shared links make benefits to synchronize the boundaries information. TRAMSIMS uses the METIS Library (nd) to cut network graph. METIS is a library to do serial graph partitioning which includes graph coarsening phase, initial partitioning

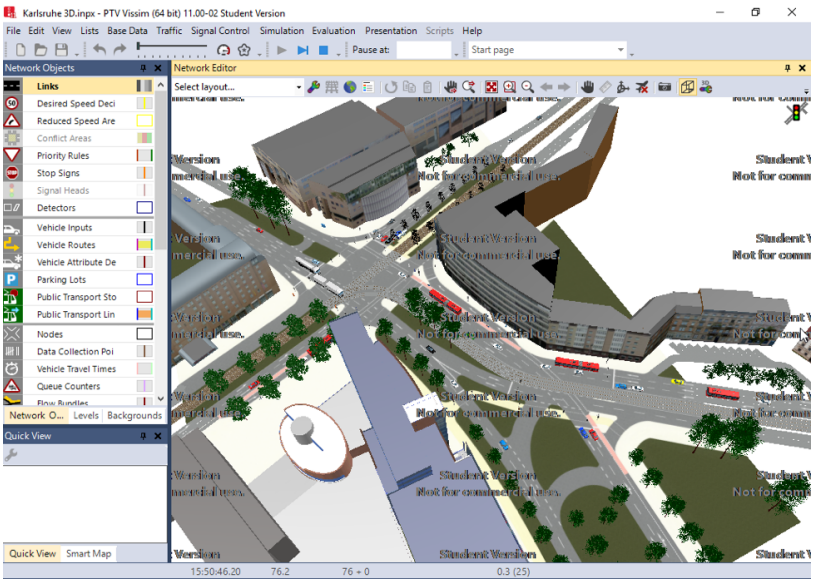


Figure 2.7: Vissim GUI

phase, and uncoarsening phase. During the process, TRANSIMS calculates accumulated computational loads at the nodes and recursively pick domains.

- **Adaptive Load Balancing.** To be efficient, the loads on different CPUs should be as similar as possible while the load depends on the actual vehicle traffic in respective domains. TRAMSIMS adapts the actual execution time of each link and each intersection to partitioning algorithm. Estimated first simulation iteration determines how to partition the road network graph.

2.2.5 Vissim

Vissim (2019) is a microscopic multi-modal traffic simulation software developed by PTV Planung Transport Verkehr AG in Karlsruhe, Germany. Vissim is a widely used microscopic simulation software indicates the detailed driving behaviors and provide plenty of customization solutions. Figure 2.7 is an intersection example in Karlsruhe.

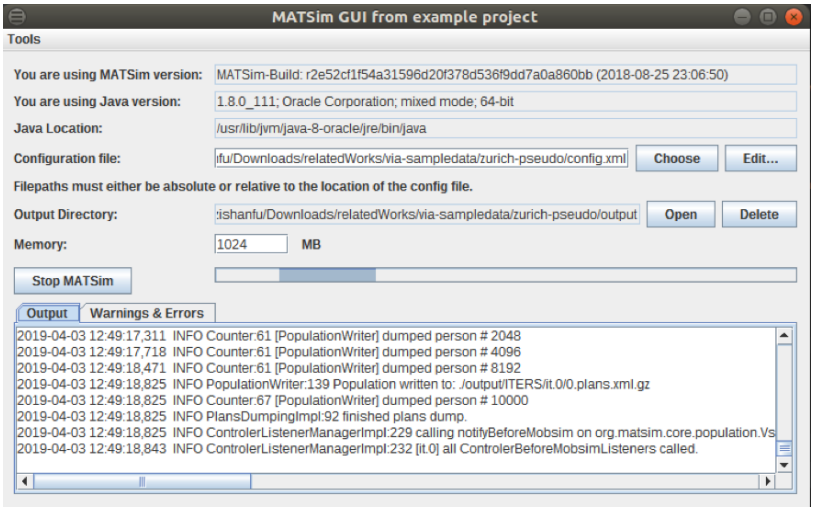


Figure 2.8: MATSim Simulation UI

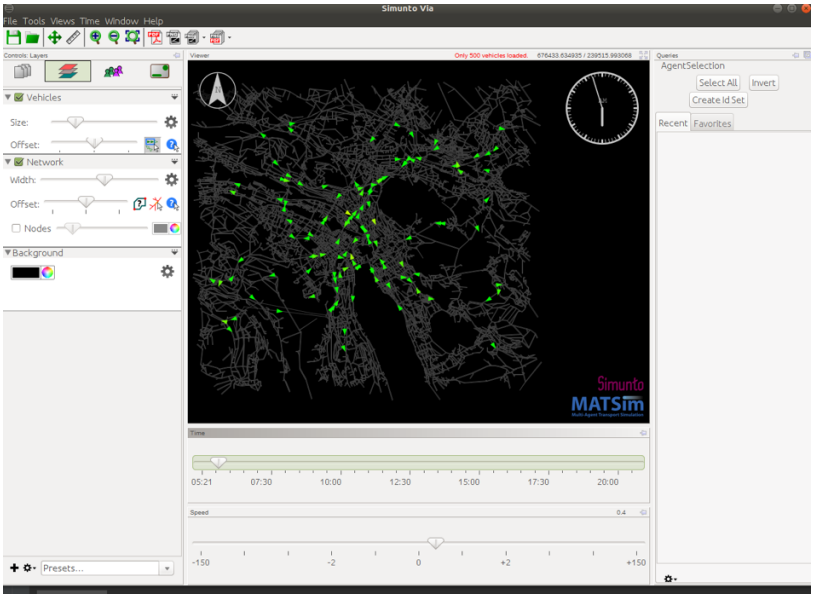


Figure 2.9: MATSim Traffic Visualization GUI

2.2.6 MATSim

Multi-Agent Transport Simulation Toolkit (MATSim) Waraich *et al.* (2009) is an open source toolbox to run and implement large-scale agent-based transport simulations. Multi-agent system consists of agents and their context. In such a system,

it is difficult to consider an individual agent without interacting with other agents. MATSim offers a set of the flexible toolkit that user can customize the modules and do personalized simulation job. MATSim makes the most of the CPUs by multi-threading simulation job. Multithreading can execute multiple processes or thread concurrently. Figure 2.8 is MATSim simulation UI. By configuring the simulation parameters, the user can generate traffic data iteratively and visualize it by the GUI in Figure 2.9 which is an example for Zurich.

- **Simulation Models History.** Generally, the approach to simulate agent in MATSim is to take advantage of the queue. The queue is a collection in computer science that the elements follow Fist-In-First-Out (FIFO) data structure. This means the first element added to the queue will be the first one to be removed. MATSim applied a queue to execute the operations in road network links, like a street. Each link has a queue to report the entry time of agent and adjacent links collaborate and exchange agents and corresponding messages to make sure simulating correctly. Based on this approach, MATSim designed QueueSim in C++, a fixed-increment time advance model that agents move along in fixed time steps of a second. In order to make it faster, MATSim implemented Deterministic Event-Driven Queue-Based Traffic Flow Micro-Simulation (DEQSim). Instead of fixing the time steps, the agent changes state by discrete actions related to the event-based model, such as entering a street, leaving a road street, etc. And MATSim parallelizes DEQSim to improve the simulation performance. In the end, MATSim corporate QueueSim and DEQSim in Java version to make it more maintainable, called JDEQSim.

- **Parallel JDEQSim.** JDEQSim contains three main components, simulation units, messages and scheduler. Simulation units in MATSim are mainly about

vehicles and links. Messages are the information exchange medium including vehicle simulation knowledge, such as a vehicle leave a street and enter another street. The schedule is a message priority queue which sorts message time and message type that controls the agent's actions. For example, the first vehicle in the link queue will be the first vehicle to leave the link because the vehicle has the earliest entering time. The road network is divided into two parts in the vertical direction and the roughly the same number of events assigned to each subset. By round robin to assign the same amount of event handlers to each thread, MATSim handles simulation events in parallel. The messages of the vehicle passing across different parts are synchronized between the threads. For example, if one thread schedules a message from another thread, synchronized access to the queue object is required. By the end, in order to make sure synchronization correctness, a small time delta is defined to fetch message and process to message executor thread.

2.2.7 ParamGrid

ParamGrid Klefstad *et al.* (2005) is a distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation.

- **Network Division.** In ParamGrid, a vast traffic network is divided into tiles of equal size area with the number of rows and columns. In the dividing process, boundary zones are generated when source and destination zones are in different network. Cutter determines the bounding box, calculates rows and columns, partitions the network items, splits link and add the boundary zones. ParamGrid develops a demand divider to generate a set of the origin-destination matrix which stores the approximate simulation workload of the vehicle. The matrix will be applied to the cutting process and proximate balance the work-

load. When a routing vehicle travels across tiles, with knowledge of all boundary zones, global route services will synchronize these vehicles.

- **Components.** ParamGrid uses Pramics Website (2019), a suite of microscopic simulation modeling tools, to do traffic simulation job. Paramics serves as the single-CPU simulator for ParamGrid. Common Object Request Broker Architecture (CORBA) Vinoski (1997) is a standard designed to facilitate the collaboration between systems. The ACE ORB (TAO) Wiki TAO (nd) is an open source c++ implementation of CORBA based on Adaptive Communication Environment(ACE) which acts as middleware to CORBA communication standards. TAO provides distributed communication between machines.

- **Architecture.** ParamGrid follows master-slave distributed computing architecture. The master controller manages three services, global routing service, CORBA naming service and event service. Master cuts the network, globally assign the name and location to each tile, handles cross tiles vehicles and provide a broadcast channel to synchronize simulation time frame. The slaves take charge of three plug-ins, object request broker, vehicle movement handler, simulation synchronization handler. A slave is responsible for the sub-network tile from manage vehicle movement, receive transferred vehicles and synchronize the simulation.

2.2.8 Smarts

Scalable Microscopic Adaptive Road Traffic Simulator (SMARTS) Ramamohanarao *et al.* (2017) is a distributed large-scale microscopic simulator which is able to utilize multiple processes in parallel.

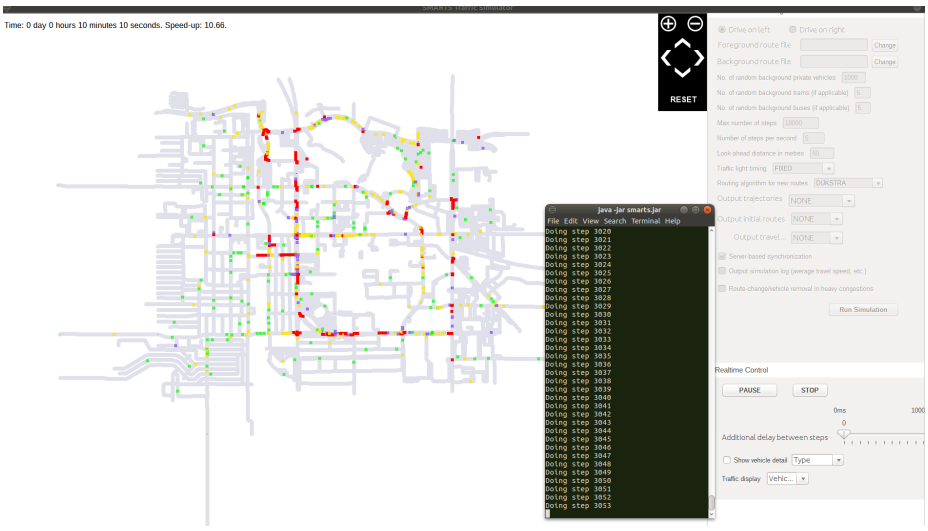


Figure 2.10: SMARTS Simulation Interface

- **Architecture.** SMARTS provides a comprehensive set of simulation features and it is mainly in three categories: input, simulation, and output. Road network, routes, and a setup script is the input for SMARTS. The road network data extracted from OpenStreetMap OSM (2019) by loading the external OSM file. The vehicle trajectories generated by the standard shortest path algorithm, Dijkstra and a route contains the vehicle's ID, start time, type and a sequence of nodes that the vehicle will visit. The setup script is a simulator configuration description of simulation parameters, such as maximum number of steps, number of internally generated random vehicles, maximum impeding distance, number of updates per second, etc. After preparation, the simulation will immediately run and SMARTS uses several microscopic simulation models to make more close traffic prediction, such as car-following, lane-changing, traffic light, route changing, traffic rules, and calibrations. The simulation results will be stored in the disk and visualize in a graphical user interface.

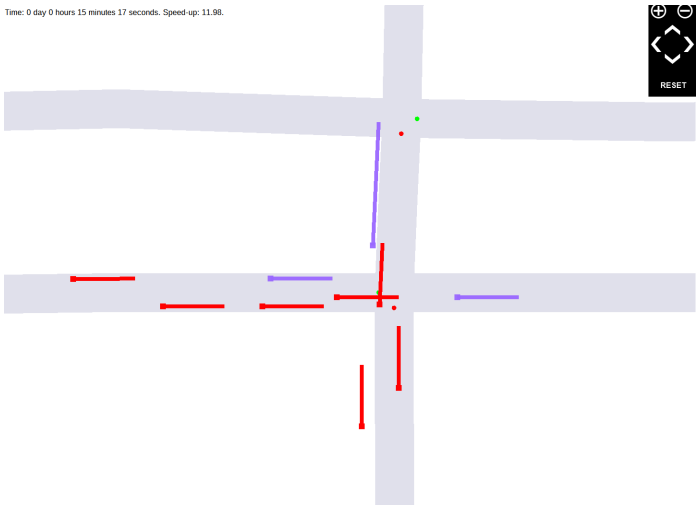


Figure 2.11: SMARTS Snapshot

- **Spatial Workload balancing.** The start coordinate of vehicles are generated following network distribution; thus they assume the simulation workload is approximately equal to network distribution. A set of grids divided from road network data and each grid is assigned to an ID. SMARTS adopts a typical master-slave distributed computing model in which the master machine manages the simulation configuration jobs, such as assigning workload to slave and a slave is an executor to run the local simulation.

- **Synchronization.** SMARTS proposed two simulation synchronization strategies: centralized synchronization (CS) and decentralized synchronization (DS). In CS, the server asks all the workers to exchange information and simulate traffic at each time step. For instance, in each time step, the master will schedule all the job and workers will respond when they received the command. Compared to CS, the master doesn't play such a critical role and hold fewer pressures in DS. Master only assign the first simulation job, and the worker will automatically run the simulation step by step after that.

2.3 Related Works Comparison

There are several classic traffic simulators proposed in the past two decades including Brinkhoff (2002) and BerlinMod Düntgen *et al.* (2009). The caveat of using these approaches is that they do not consider microscopic traffic models Krajzewicz *et al.* (2002), and hence cannot simulate individual vehicle driving behaviors and do not consider a variety of road simulation scenarios such as traffic signals, intersections, traffic rules, speed limitation and so on. For example, a vehicle will not stop in red light and moderate speed on the distance to heading vehicle. Most moving objects overlap in the simulator and this kind of data is not useful when the researches need traffic data in a certain quality. Microscopic traffic models are beneficial in practice since they can generate and simulate data matching the real-time traffic. A vital fact is that whether it is a pedestrian, bicycle or a car, all the moving objects in the world maintain a safety provision. These objects pay attention to the environment, check possible dangers, communicate with other moving objects and make corresponding changes. Computing these actions in simulation need to check front and back objects in the current lane and desired road lane. The more objects simulator to test, the more safety actions objects have. However, a simulation involving many characteristics is computation-intensive. For example, traditional microscopic simulators such as SUMO Krajzewicz *et al.* (2002) are only able to simulate a limited number of vehicles over a small size of the road network.

Recently, there have been several research works that proposed scalable microscopic simulators which can horizontally parallelize the simulation workload by adding more machines. However, performing microscopic traffic simulation in a distributed environment is very challenging because:

- **Workload balance.** A scalable simulator needs to partition the workload

to small chunks and assign them to different machines in a cluster. However, whenever a vehicle tries to change lane or accelerate, it has to check the surroundings, like the distance to nearby vehicles, traffic lights, etc. A proper partitioning method should take into account the spatial proximity of vehicles and minimize cross-partition data exchange, communications costs, and simulation synchronization.

- **Dynamic distribution.** The spatial distribution of moving vehicles will dynamic changes over time. Nearby vehicles in last moment may soon become far from each other. Simulators have to employ proper mechanisms to handle dynamic events.

To deal with the challenges, TRANSIMS Nagel and Rickert (2001) opts to use graph partitioning approaches to partition road networks but does not consider their spatial distribution. The road network based partitioning methods may not accurately balance the vehicle simulation workload because most roads in a road network are idle and only major streets are full of vehicles. ParamGrid Klefstad *et al.* (2005) proposes to partition the geographical space to uniform grids which do not work well if the cars and road network have skewed distribution. SMARTS Ramamohanarao *et al.* (2017) comes up with an approach that partitions the area into small chunks numbered in a Z-curve Zhang *et al.* (2003) like order. It then assigns nearby pieces to the same machine. However, it makes an unrealistic assumption of the fixed spatial distribution of moving vehicles.

Besides, most existing scalable simulators are designed upon inefficient distributed computing models. For instance, Parallel BerlinMod Lu and Guting (2012) uses Hadoop MapReduce Hadoop (nd) and SMARTS Ramamohanarao *et al.* (2017) leverages simple TCP sockets. Apache Spark, on the other hand, provides a novel data

Table 2.1: Comparison Among Different Traffic Simulators

Feature	Brinkhoff	SUMO	BerlinMOD	TRANSIMS	MATSim	ParamGrid	SMARTS	GeoSparkSim
Simulation model	Macroscopic	Microscopic	Macroscopic	Microscopic	Microscopic	Microscopic	Microscopic	Microscopic
Scalability	Single node	Single node	Distributed	Distributed	Multi-thread	Distributed	Distributed	Distributed
Workload partitioning	-	-	Hash	Graph cut	Uniform grids	Uniform grids	Z-curve	Quad-Tree
Partition organization	-	-	Fixed	Fixed	Fixed	Fixed	Fixed	Dynamic
Distribution model	-	-	MapReduce	MPI	Thread sync.	CORBA	TCP sockets	RDD

Brinkhoff Brinkhoff (2002), SUMO Krajzewicz *et al.* (2002), BerlinMOD Düntgen *et al.* (2009), Distributed BerlinMOD Lu and Guting (2012), TRANSIMS Nagel and Rickert (2001), MATSim Waraich *et al.* (2009), ParamGrid Klefstad *et al.* (2005) and SMARTS Ramamohanarao *et al.* (2017)

Distribution highlights: MapReduce Hadoop (nd), MPI Gabriel *et al.* (2004), CORBA Vinoski (1997) and RDD Zaharia *et al.* (2012)

abstraction called Resilient Distributed Datasets (RDDs) Zaharia *et al.* (2012) that are collections of objects partitioned across a cluster of machines. Each RDD is built using parallelized transformations (filter, join or groupBy) that could be traced back to recover the RDD data. In memory RDDs allow Spark to outperform existing models.

It requires tremendous efforts to develop a scalable traffic simulator that fits a distributed environment because the simulator has to deal with a new problem that is how to balance the spatial workload to minimize data shuffle. Researchers have come up with many different approaches, explained below (see Table 2.1). A study of distributed execution models used in these simulators is given in Appendix A.

2.4 Simulator Scalability

Non-spatial partitioning approach. Some existing solutions partition the workload without taking into account the spatial proximity of the moving vehicles. Parallel-BerlinMOD Lu and Guting (2012) integrates BerlinMOD with a distributed DBMS called Parallel-Secondo Lu and Guting (2012) to deliver a scalable solution. It partitions the vehicles using generic partitioners such as hash partitioner and round-robin partitioner and parallelizes the computation to a set of Hadoop MapReduce operations Hadoop (nd). This approach is easy yet inappropriate for microscopic simulators because vehicles running on the same road segment are simulated by different machines. On the other hand, a microscopic simulator TRANSIMS Nagel and Rickert (2001) proposes to use graph cuts to partition the large road network then apply the same partitions to vehicles. It leverages message passing interface MPI Gabriel *et al.* (2004) to coordinate different machines in a cluster. TRANSIMS may yield balanced network partitions such that each partition has a similar number of road nodes and segments but ignores an important fact: most road networks are

idle and only major streets are full of vehicles because most traffic are gathered in the center of the city.

Spatial partitioning approach. Most scalable microscopic simulators use spatial partitioning methods to strike balanced workloads. MATSim Waraich *et al.* (2009) comes up with a technique that splits the space to uniform grids (say, 5km*5km) then uses these grids to partition road networks and vehicles. It uses multi-threads to parallelize the computation. ParamGrid Klefstad *et al.* (2005) uses a partitioning method similar to MATSim but utilizes CORBA Vinoski (1997) framework which internally uses RPC. SMARTS Ramamohanarao *et al.* (2017) partitions the space to some small cells and order them into a curve close to Z-curve. Cells that have the same ID are assigned to the same machine. Although these approaches take into account spatial proximity, GeoSparkSim still outperforms because (1) their partitioners cannot well balance vehicles due to their skewed spatial distribution. GeoSpark Yu *et al.* (2018) and SpatialHadoop Eldawy *et al.* (2015) both show that KDB-Tree and Quad-Tree partitioning approaches are better. (2) The spatial distribution of moving vehicles keeps changing during the simulation. Instead of using fixed partitions, GeoSparkSim uses a spatial-temporal partitioning approach to dynamic automatically repartition vehicles over time.

Distributed computing models. Most existing solutions are designed upon inefficient distributed models. Many of them still use message passing services and do not employ advanced computation models and job schedulers. SMARTS Ramamohanarao *et al.* (2017) leverages simple TCP sockets, TRANSIMS Nagel and Rickert (2001) uses MPI Gabriel *et al.* (2004), and MATSim Waraich *et al.* (2009) only utilizes multi-thread synchronization. On the other hand, Parallel BerlinMod Lu and Guting (2012) uses Hadoop MapReduce Hadoop (nd). Although the Hadoop-based approach achieves high scalability, it still exhibits slow run time performance since it

persists all intermediate data on disk. Apache Spark provides a novel data abstraction called Resilient Distributed Datasets (RDDs) Zaharia *et al.* (2012) that are collections of objects partitioned across the node in a cluster of machines. Each RDD is built using parallelized transformations (filter, join or groupBy) that could be traced back to recover the RDD data. In memory RDDs allow Spark to outperform existing models.

Chapter 3

SYSTEM ARCHITECTURE

GeoSparkSim has developed in Java and Scala programming language which can be deployed in standalone or distributed mode to achieve traffic prediction in high-performance. User can easily install it in any platform and customize their simulation. This chapter will discuss the application architecture, user interface, contribution and how to use it.

3.1 System Overview

3.1.1 Apache Spark

Apache Spark Zaharia *et al.* (2010a) is an open source distributed computing engine for large-scale real-time data processing and analytics. Figure 3.1 shows Spark architecture. Spark provides a set of libraries, Spark SQL, Spark streaming, Machine learning, and GraphX that user can query, construct and streaming data, run machine learning models and graph algorithms. Spark uses a master/slave architecture. Master is the entrance of driver program and manages the application, while slaves parallel computes jobs assigned by the master. Resilient Distributed Datasets Za-

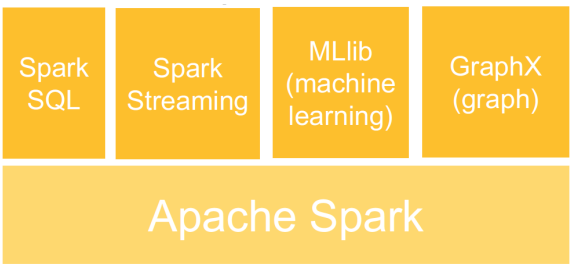


Figure 3.1: Apache Spark

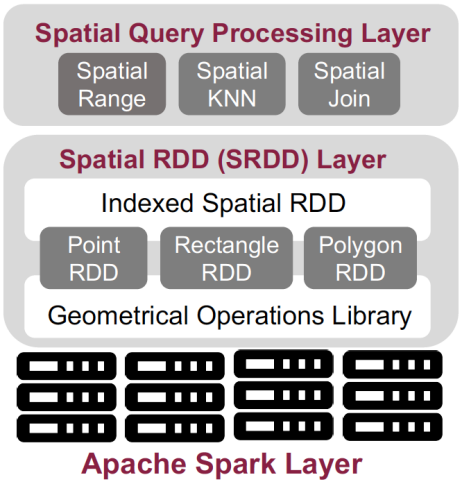


Figure 3.2: Apache GeoSpark

haria *et al.* (2010b) (RDDs) is a collection of data partitioned across the nodes of the cluster that can perform operations in parallel. GeoSparkSim builds models in RDDs and runs the simulation based on Apache Spark.

3.1.2 Apache GeoSpark

Apache GeoSpark Yu *et al.* (2018) is an in-memory cluster computing framework for processing large-scale spatial data. Figure 3.2 is GeoSpark architecture. The framework is based on Spark and provides two layers, spatial RDDs (SRDD) layer, and spatial query processing layer. SRDD layer offers a wrapper for spatial data in RDD and provides spatial partition mechanism on Spark RDDs, such as R-tree, Quad-tree, etc. Spatial query processing layer designed many spatial operations on RDDs, such as spatial range, join, k-nearest neighbors algorithm (KNN) that user can directly use these algorithms in the distributed environment. GeoSparkSim creates simulation models on general SRDD in GeoSparkSim and utilizes the spatial partition algorithm in GeoSpark.

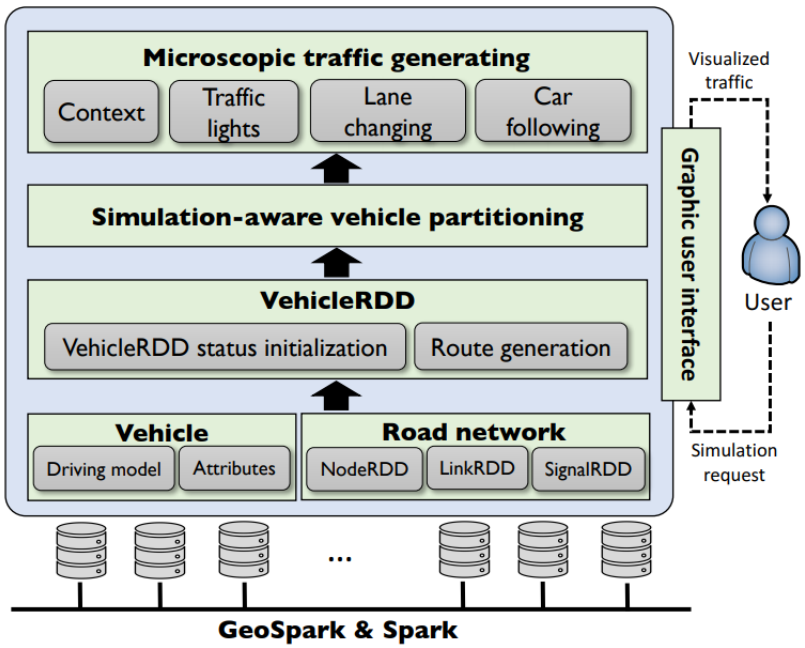


Figure 3.3: GeoSparkSim Architecture

3.1.3 Hadoop Distributed File System (HDFS)

Apache Hadoop (nd) is an open source framework that allows large-scale offline data processing across a cluster of computers by utilizing the MapReduce model. Hadoop ecosystem consists of Hadoop Common, Hadoop Distributed File System, Hadoop YARN and Hadoop MapReduce. The user can process, manage and store data in Hadoop. GeoSparkSim stores data in HDFS and distributes data to computers.

3.1.4 GeoSparkSim Architecture

GeoSparkSim consists of a Graphic User Interface (GUI) and four layers: (1) Vehicle and road network layer (2) VehicleRDD layer (3) simulation-aware route partitioning layer (4) microscopic traffic computing layer. GeoSparkSim works in concert with GeoSpark Spatial RDDs and Spark to deliver a holistic approach that allows data

scientists to simulate, analyze and visualize large-scale urban traffic data.

Graphic user interface (GUI). Users can interact with GeoSparkSim by the front-end map interface which provides two functions: (1) it takes input parameters from users including the number of to-be-simulated vehicles, simulation region, vehicle context, time step, simulation period and so on. A user can directly draw a rectangular window on the map and fill in necessary parameters. Then GeoSparkSim backend will download the road network of the specified region, generate simulated traffic data and visualize back to GUI.

Vehicle and road network. Each vehicle has several driving behaviors and attributes such as acceleration/deceleration, velocity, safe distance and so on. The values of these attributes are randomized in a specific reasonable range, so each vehicle has its personalized behavior. The user can also control attribute values via a vehicle configuration file. Besides that, each vehicle also has its status to record its current simulated speed, GPS locations, and acceleration state. Road network describes the road situation of the specified simulation region and consists of three RDDs, NodeRDD, LinkRDD, and SignalRDD. NodeRDD contains all road junctions, and LinkRDD contains all road segments. Besides that, there is a SignalRDD to describe specific scenarios in the road network.

VehicleRDD. VehicleRDD is a specialized Spark RDD which consists of millions of individual vehicle records. GeoSparkSim first creates the initial status vehicles in this layer. Then it randomly generates sources, destinations and route length in a particular range for every vehicle following on the specified distribution. It leverages an open source library to build an index over the static road network. This index contains lots of pre-computed shortest paths, so GeoSparkSim parallel computes routes for every source and destination pair on top of it, gather the results and convert to VehicleRDD.

Simulation-aware vehicle partitioning. After route planning, every vehicle in VehicleRDD has a planned route. These vehicles will precisely follow the expected path, but each of them will show different microscopic driving behaviors. To simulate the microscopic model of a single vehicle, GeoSparkSim needs to know the status of nearby vehicles and road network information. To scale out such simulation to millions of vehicle in VehicleRDD, GeoSparkSim repartition the VehicleRDD and road network according to their spatial proximity such that it can perform local microscopic simulation inside each VehicleRDD partition. The repartitioning occurs periodically to reflect the vehicle distribution because vehicles may move to different locations on the road network after a while.

Microscopic traffic computing. Given a VehicleRDD and the road network partitioned by the vehicle partitioner, GeoSparkSim will then run the microscopic simulation in each VehicleRDD partition and its corresponding road network partition. This local simulation generates traffic with individual object mobility pattern which consists of vehicle status at each time step. Each vehicle has a safe distance to avoid collisions. A vehicle will moderate the speed if its next movement invades the safe distance to nearby vehicles or objects. Traffic signals at road intersections also affect the traffic.

Distributed Computing. GeoSparkSim distributes the simulation work in master-worker mode. Master is responsible for handle simulation requests from the user, fetch road network data, vehicle generation, and visualization. After assigning data preparation and simulation task, the cluster will run tasks in parallel. Figure 3.4 is the distributed GeoSparkSim Architecture.

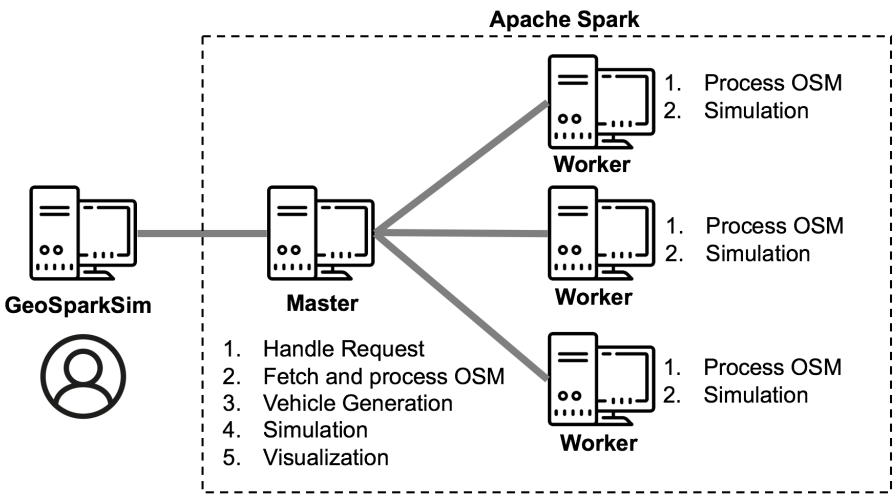


Figure 3.4: Distributed GeoSparkSim Architecture

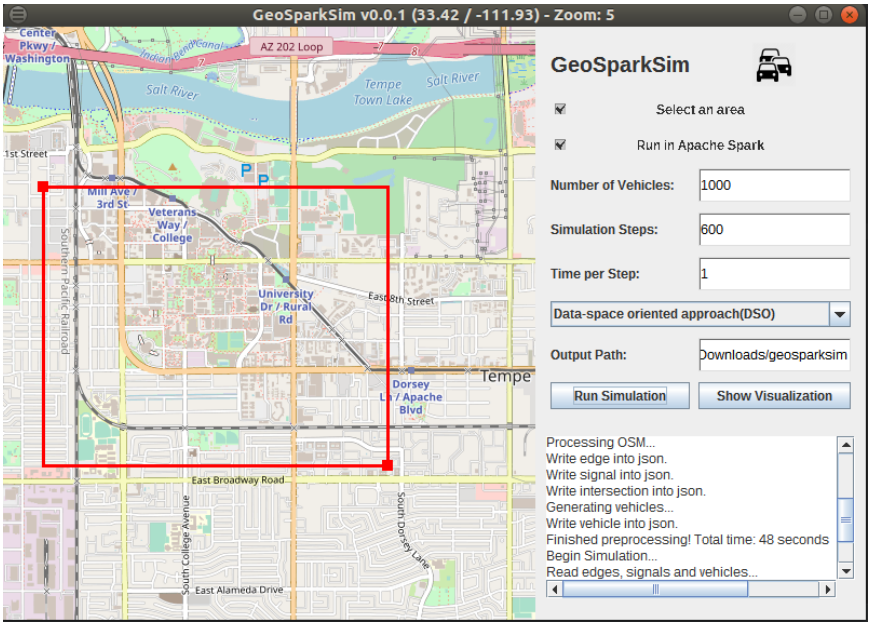


Figure 3.5: GeoSparkSim GUI

3.2 Graphic User Interface (GUI)

GeoSparkSim provides a graphic user interface that allows users to interact with the system. The user can issue simulation requests and see visualized simulation results via this interface. Figure 3.5 is a GUI example.

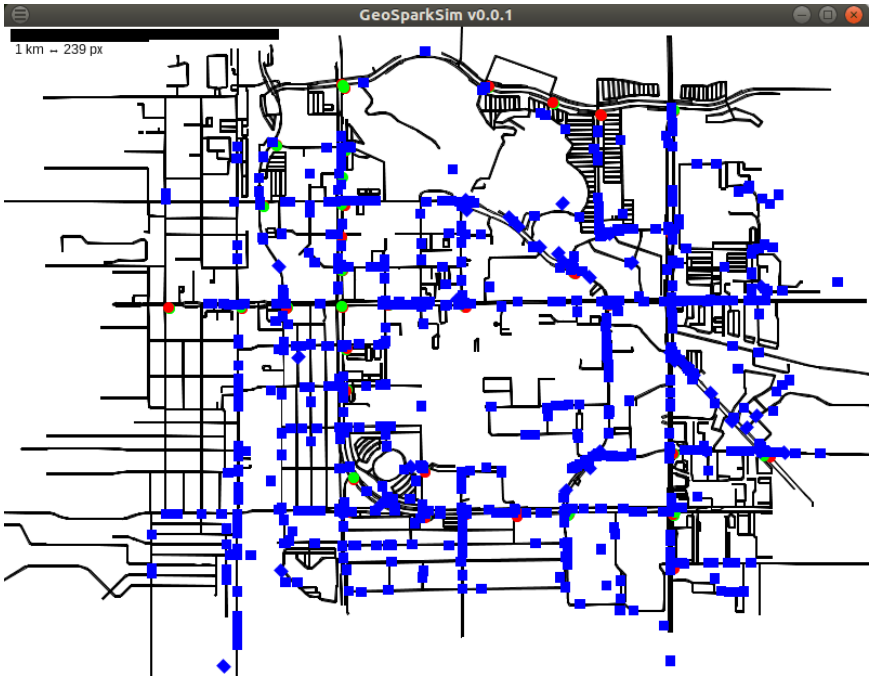


Figure 3.6: GeoSparkSim Traffic Visualization

Interface components. GeoSparkSim user interface contains three main parts: input panel, map panel and report panel. The input panel on the right-top is the place where the user can describe their personalized simulation request by checking several options and fill the parameters. The map panel on the left shows the viewport of a road network with map background. Users can zoom in/out and pan on this panel to see different regions. The report panel on the right-bottom designed to show the description of each processing step, like completion time, and keep track of the simulating process.

Issue a simulation request. The user can begin to enter the number of moving object, select VehicleRDD initialization approach and the simulation period on the input panel. Then the user is required to draw a rectangle on the map panel to specify the simulation region.

Visualize the simulation result. Once the simulation is done, the user can opt to ask for visualized simulation results. The GUI will create a new simulation graphical frame in order to render all road network elements and simulated vehicle locations to segments and points by every simulation time step. In each time step, the graphical painter will update and redraw the points from simulation results to show the latest simulation locations and update all road network objects events. The simulation panel is created by Java Swing which can run in any platform, and the simulator renders the real road network at scale. The GUI keeps listening to the user events from the mouse wheel and mouse motion. If the user zooms in or zooms out the simulation, the panel will repaint all the elements coordinate projection in the simulators panel. If the user drags and moves the center of the simulation, the panel will make corresponding coordinates projection changes. Figure 3.6 is an traffic visualization example for Arizona State University.

3.3 User Documentation

GeoSparkSim can be packaged as a Java ARchive (JAR) file which is a package file format used for aggregate java class file, needed dependencies, resources, and metadata into one file for use and distribution. The user can run the following command to start GeoSparkSim.

- **Command setting.** GeoSparkSim provides option fields with which user can interactively play with GeoSparkSim and personalize their requests. For example, $-o$ means showing the user interface. By default, GeoSparkSim configures some default values to provide a sample simulation. Figure 3.7 is the GeoSparkSim help command interface. It shows all the available optional parameters. By default, GeoSparkSim will not show the visualization if the number of vehicles is larger than 5000 or the area size is larger than 8,000,000 square meters.

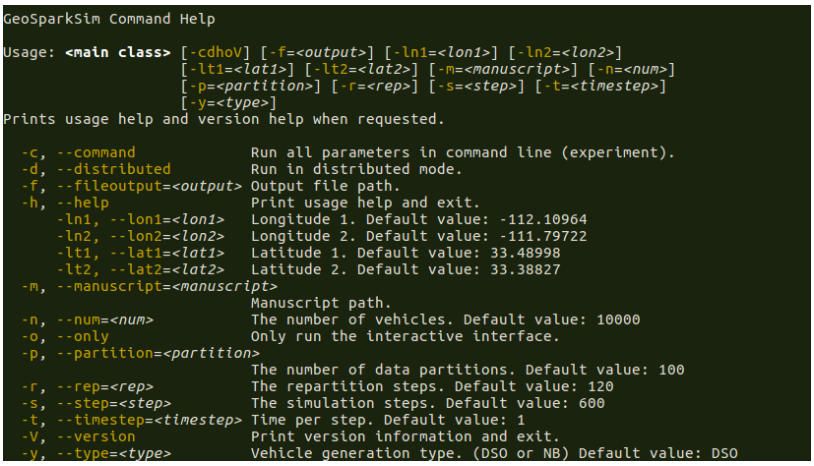


Figure 3.7: GeoSparkSim Command Line Tools

Output	Description
map.osm	Raw OpenStreetMap map data by user selected rectangle
node.parquet	Node data processed from map.osm
way.parquet	Way data processed from map.osm
map-gh	Route planning map index
edges.json	Structured edges data
signals.json	Structured traffic signals data
intersections.json	Structured uncontrolled intersections data. (exclude signal intersection)
vehicles.json	Structured vehicle data
reports	Simulation results by step

Table 3.1: Outputs Description

- **Output.** User can set the output path and GeoSparkSim will save all the road network, vehicle route and simulation results all in the path.

- **User-defined traffic model.** GeoSparkSim by default uses intelligent driving model Kesting *et al.* (2010a) and MOBIL Kesting *et al.* (2007a) (minimizing overall braking induced by lane change) models to moderate velocity and perform lane changing. GeoSparkSim allows the user to plug in his or her traffic model such the simulation result can fit in any specific scenario.

GeoSparkSim provides two abstract classes, *car follow* and *movement control*. The user can easily extend them and implement abstract methods such as *safe distance check* and *movement control*. *Safe distance check* takes as input a vehicle and road network information and returns vehicles or traffic lights ahead of the input vehicle. The user can define the checking mechanism for different vehicles and assign priorities to different vehicles. Given the current status of a vehicle, *movement control* computes the next movement of this vehicle, such as acceleration, velocity and lane change.

3.4 Contribution

This paper presents GeoSparkSim, a scalable microscopic traffic simulator, which extends Apache Spark to generate large-scale road network traffic data with various microscopic traffic models. The proposed system seamlessly integrates with a Spark-based spatial data management system, GeoSpark, to deliver a holistic approach that allows data scientists to simulate, analyze and visualize large-scale traffic data. Specifically, the proposed system has the following contributions:

• GeoSparkSim converts road networks to Spark graphs and simulated vehicles to VehicleRDDs. Then it parallelizes each step in traffic simulation into a set of RDD transformations. Such transformation efficiently distributes the computation-intensive simulation workload to every machine in a cluster.

• GeoSparkSim takes into account microscopic traffic models such as traffic lights, lane changing, and car following. To achieve that, it employs a simulation-aware vehicle partitioning method to partition vehicles among different machines such that each machine takes a roughly similar amount of simulation workload to achieve load balance. This partition mechanism intuitively considers both temporal attribute and spatial attribute of vehicles to handle the dynamic spatial distribution.

• A full-fledged prototype of GeoSparkSim is implemented in Apache Spark. Our experimental analysis shows that GeoSparkSim can simulate the movements of 200 thousand vehicles over a very large road network (250 thousand road junctions and 300 thousand road segments).

Chapter 4

ROAD NETWORK

The road network is a system interconnecting streets segments and holding vehicles and pedestrian traffic. Figure 4.1 shows a road network map example for metropolitan Phoenix area with live traffic. Green to red or black lines show the speed changes from fast to slow. Road network is the fundamental base of traffic simulator. The quality and complexity of these data make a huge difference in the simulation. In order to achieve high performance in the processing road network, GeoSparkSim handles all road network elements in Spark and convert it to serializable and big data framework compatible structure that user can do various large-scale of analysis on it. Figure 4.2 is the road network converter architecture. After a user selects the region, road network OpenStreetMap will be downloaded and a sinker is used for split and process node and way from XML. Node and way are saved as parquet format in HDFS or local disk. GeoSparkSim read node and way into data frames and do some relational operations to get ideal data. Lastly, these data will be converted to RDDs, do a set of transformation and convert to road network graph. GeoSparkSim possesses three road network data structures, NodeRDD, LinkRDD, and SignalRDD generated from OpenStreetMap. This chapter presents its internals and explains how to process and structure them in Spark.

4.1 OpenStreetMap

OpenStreetMap(OSM) is an open source user-generated world map database. As Google Map, OSM gives sophisticated and continuously updated street information in high quality. There are a large number of frameworks related to or support OSM in

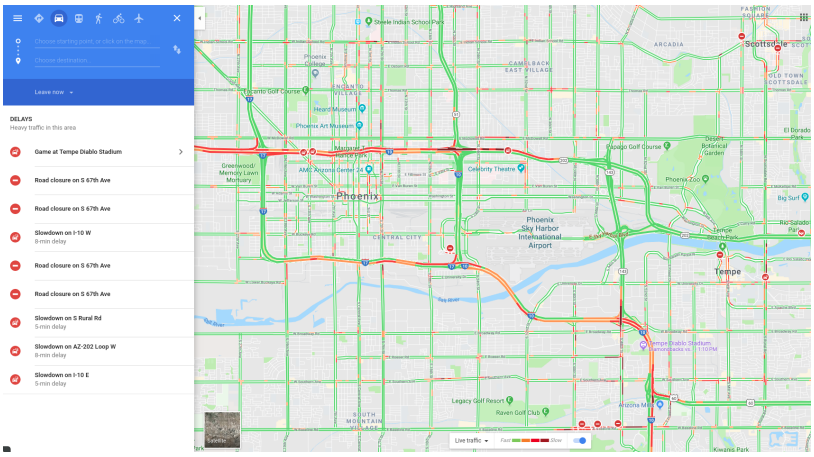


Figure 4.1: Phoenix Road Network with Traffic

Roads and Freeways in Metropolitan Phoenix with Live Traffic in Google Map

(March 15 2019 Friday 13:10)

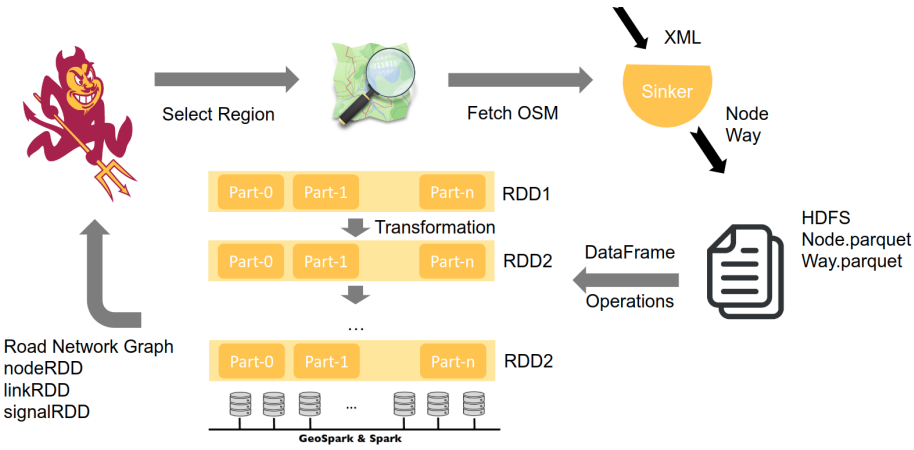


Figure 4.2: Road Network Process Architecture

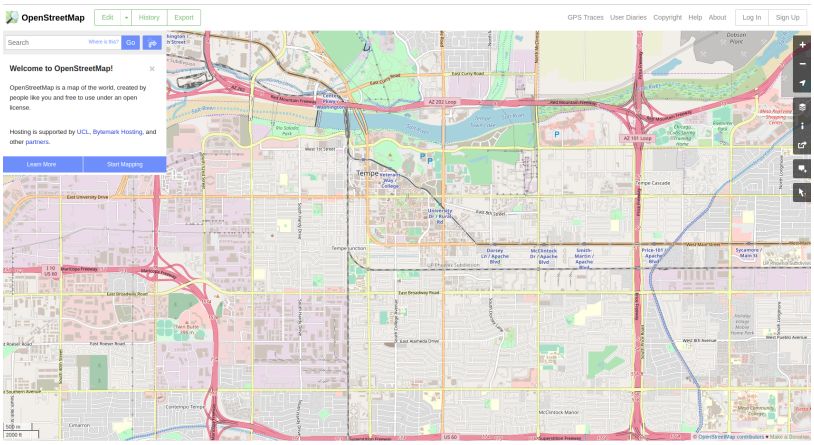


Figure 4.3: Tempe City Portrait in OpenStreetMap

different disciplines, programming language and platform supports. These framework functions from accessing, processing, generating, displaying to navigating, such as Mapbox, a great interactive map. Figure 4.3 shows the Tempe City map portrait in OSM. There are three main elements in OSM: nodes, ways, and relations.

- **Node.** Nodes define all the points in the space which includes node id, coordinate and tags. Node ids are a unique identification of the node in the area, coordinate in degrees uses the standard WGS84 projection GeoSystem (nd), and tags represent the map features for nodes, for example, $Tag : traffic\_signals = signal$.

- **Way.** Ways define the linear features and area boundaries which includes way id, tags, and nds. An ID is the unique key of way, tags describe the way segment's features and nds is a sequential collection of the nodes Id in the way.

- **Relation.** Relations are used to explain how elements work together, like a bus route is composed of many ways.

OSM uses tags to label node, way or relation's characters. The tagging system describes specific map features by utilizing the key-value pair. Key is attribute name

and value is associated with the attribute, such as highway=footway, maxspeed=50. Following is an example showing a street way with id 5090250 and characteristics when the contributor sets up the data for the street. Nds are the nodes along the way in sequence and ref is the unique id for the node which is same to the node id. Tags are the features for the road and the example way is a residential highway, name Clipstone Street and a one direction way. GeoSparkSim cut ways in following segments and each segment consists of two nodes. All ways features are attached to those segments. Listing 4.1 is an example of OSM XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="Overpass API 0.7.55.5 2ca3f387">
<note>The data included in this document is from www.openstreetmap.org. The data is made available
     under ODbL.</note>
<meta osm_base="2019-03-17T08:11:02Z"/>
  <bounds minlat="33.4108960" minlon="-111.9506450" maxlat="33.4331220" maxlon="-111.9225800"/>
  <node id="41459438" lat="33.4245016" lon="-111.9275146" version="5" timestamp="2017-02-04T18:03:38Z"
        changeset="45811624" uid="292665" user="Dr Kludge"/>
  <node id="41520505" lat="33.4307934" lon="-111.9442680" version="4" timestamp="2014-09-28T23:30:45Z"
        changeset="25735513" uid="227972" user="Your Village Maps">
    <tag k="noexit" v="yes"/>
    <tag k="source" v="Bing"/>
  </node>
  <node id="41529691" lat="33.4231268" lon="-111.9304264" version="6" timestamp="2017-02-04T18:03:38Z"
        changeset="45811624" uid="292665" user="Dr Kludge">
    <tag k="highway" v="traffic_signals"/>
  </node>
  <way id="544156292" version="1" timestamp="2017-12-02T15:35:03Z" changeset="54273258" uid="1378289"
       user="ParagonPrime">
    <nd ref="5260201424"/>
    <nd ref="5260201425"/>
    <nd ref="5260201426"/>
    <nd ref="5260201427"/>
    <tag k="highway" v="service"/>
    <tag k="service" v="parking_aisle"/>
  </way>
  <way id="597596576" version="2" timestamp="2019-03-15T06:49:28Z" changeset="68163203" uid="8407158"
```

```
    user="shadty">
  <nd ref="41830285"/>
  <nd ref="41885773"/>
  <nd ref="3983483036"/>
  <nd ref="41604298"/>
  <nd ref="3983483052"/>
  <nd ref="5341183673"/>
  <nd ref="6339179807"/>
  <nd ref="5341183648"/>
  <nd ref="5341183677"/>
  <tag k="bicycle" v="yes"/>
  <tag k="cycleway" v="lane"/>
  <tag k="highway" v="tertiary"/>
  <tag k="lanes" v="2"/>
  <tag k="lanes:backward" v="1"/>
  <tag k="lanes:forward" v="1"/>
  <tag k="name" v="South College Avenue"/>
  <tag k="smoothness" v="good"/>
  <tag k="surface" v="asphalt"/>
</way>
<relation id="56412" version="6" timestamp="2017-03-25T17:05:53Z" changeset="47157684" uid="665748"
    user="sebastic">
  <member type="way" ref="28822157" role="outer"/>
  <member type="way" ref="28822679" role="inner"/>
  <tag k="addr:city" v="Tempe"/>
  <tag k="addr:country" v="US"/>
  <tag k="addr:housenumber" v="1050"/>
  <tag k="addr:state" v="AZ"/>
  <tag k="addr:street" v="South Forest Mall"/>
  <tag k="alt_name" v="ED"/>
  <tag k="building" v="yes"/>
  <tag k="name" v="H B Farmer Education Building"/>
  <tag k="ref" v="ED"/>
  <tag k="type" v="multipolygon"/>
</relation>
</osm>
```

Listing 4.1: OSM XML example

4.2 Data Importing

Even for the small size of the region, it is common to have a large amount of OSM data because the size of the road network is mainly depended on the complexity. Geofabrik is a company having a close collaborative relationship with OSM. In their website GeoFabrik (nd), the compressed Europe street data takes more than 19 gigabytes and more than 30 gigabytes for non-compressed data. Handling a large amount of OSM data is also a challenging task in GeoSparkSim.

A user can select any region in the world and GeoSparkSim will identify the bounding box of the area, for example, top-left coordinates and bottom-right coordinates. Then construct the URL, download arbitrary region in XML format by overpass API and extracts all OSM elements. The Overpass API Overpass (nd) is an interface to select customized OSM map data which acts as a web database. A user can customize their request and send it through API and get results back from the server database. Following is the example of querying OSM data from Overpass.

```
http://overpass-api.de/api/map?bbox=left,bottom,right,top
```

Osmosis (nd) is a Java library for processing OSM data. GeoSparkSim leverages Osmosis XML sinker to parser each OSM element and output OSM in the desired format. First, it filters the unallowable and irrelevant ways, formats the data to the schema shown in Figure 4.2. In order to save storage and encode the schema, GeoSparkSim processes the road network data in parquet format. Apache Parquet (nd) is a columnar storage format which is compatible with any projects in the Hadoop ecosystem. The OSM data comes in the nested data structure and each data field has the same format. The most data processing operation in GeoSparkSim is based on the column instead of the whole role. For instance, GeoSparkSim converts the nodes array in the way to segments in the graph which involves the flatten transformation

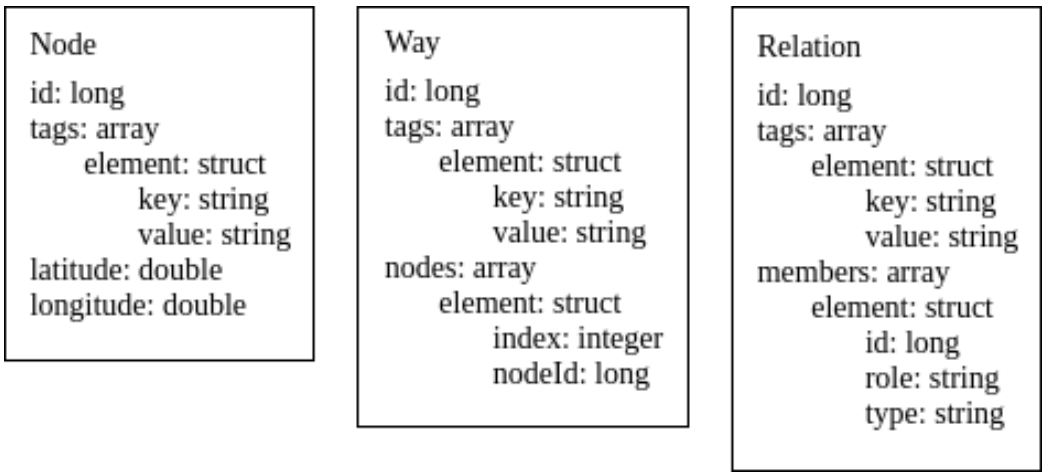


Figure 4.4: OpenStreetMap Schema

operations. Store data in parquet format could take advantages of efficient schema encoding and data compression. For example, the raw data takes more than 100 megabytes and takes less than 700 kilobytes after convert into parquet format which reduces size more than 100 times. GeoSparkSim decomposes the OSM data into node parquet and way parquet files, and store it in the Hadoop Distributed File System (HDFS (nd)) among the cluster.

4.3 Network Converter

GeoSparkSim creates three serializable classes, lane, segment link, and segment node to parse data and perform a set of operations in Spark. Serialization is the process to translate data structure and status to a storable, transmittable and reconstructable structure. In order to build an object in Spark RDD and distribute given operations, the objects and functions used in the operations should be serialized. Everything needed and referenced for the operations should be packaged, serialized and send to executors to run. Executor in Spark is regarded as a managing agent responsible for executing tasks, and it will send the serialized objects and operations for

Attribute	Type
ID	Int
HeadLine	Coordinate
TailLine	Coordinate
Head	Coordinate
Tail	Coordinate

Table 4.1: Lane Class

workers to run. When the worker finished tasks, gathering the results from workers back and deserialize it to origin format. It is a widely held view that serialization is another challenge for computing simulation in Spark because not all the objects or data structure could be easily serialized. This section will introduce network structure and how GeoSparkSim converts the network to these classes.

Lane contains id, boundary coordinates, and central coordinates. Lane id is the unique identifier generated when we compute lane geometries from way's information. In this case, lane id is in the range from 0 to a total number of the lane. Boundary coordinates determine the envelope of the road lane and central coordinates are used for compute moving object trajectories. Table 4.1 shows the lane class schema.

SegmentNode contains node id, coordinate, signal and uncontrolled intersection. Table 4.2 is SegmentNode class schema. **SegmentLink** class represents processed road network segments which include way id, head segmentNode, tail segmentNode, distance in meter, speed in meter per seconds or mile per hour, driving direction and the number of lanes. Table 4.3 is SegmentLink class schema.

GeoSparkSim builds a road network graph on Spark GraphX Xin *et al.* (2013) by parsing nodes into vertices and ways into edges. We filter the nodes never showing in ways which means the useless nodes and split the nodes into two sets. One is

Attribute	Type
ID	Long
Coordinate	Coordinate
Signal	Boolean
Intersect	Boolean

Table 4.2: SegmentNode Class

Attribute	Type
ID	Long
Head	SegmentNode
Tail	SegmentNode
Distance	Double
Speed	Int
DriveDirection	Int
Lane	Int

Table 4.3: SegmentLink Class

intersection nodes set, and another is non-intersection nodes which lay down along the segments between two vertices. The raw OSM data contains nodes and ways, but a necessary representation of the directed graph needs vertices and edges. Vertices are used to represent intersections along the road and edges represent the link that connects two intersections. The development of road network includes three primary operations, find the intersection nodes among the nodes array in the way, split way to segment edge links and build the graph by the intersection node vertexes and segment link edges.

1. Process Nodes

A node usually is the junction, turning point or end point of the way. The node schema in the last figure shows id, latitude, longitude, and tags. Read node parquet to the dataset in Spark, convert key-value pair from tag column to a map and filter the nodes which contain the key of highway and value of traffic_signal. Format the traffic signal datasets, join it with node dataset and mark signal node in the signal column.

2. Process Nodes from Ways

   A way consists of a sequential collection of nodes, and the intersection is a junction where two or more roads meet or cross which can be classified by the segment ways. Identify the intersection nodes by aggregate and filter the nodes showing more than once in the ways. Label these nodes as intersection nodes which will represent the vertex in the graph. In order to gather all road network relation, the nodes processed from the last step are fully joined with the intersection nodes.

3. Process Ways to Segment Links

   After identifying all way nodes, we need to convert the road relation to simulation information. For example, the non-directional way geometries should be converted to directional divided lane geometries. In the way transformation, we extract way id, way tag map and node sequence. In the tag map, format speed limits, report lanes, select driving direction, identify one-way road and roundabout road and label the bidirectional way. Since the nodes come in sequence in the way, the segment link is generated by sliding the nodes array with windows size 2. The sequential identifier labels the head node and the tail node in the segment. The detailed information of the road is stored in the segment link, such as maximum speed, tail node, head node, distance cost, drive direction,

and total lanes. The distance cost is the distance in the mile between the head node and the tail node which combines the maximum speed to calculate the route. The maximum speed in mile per hours is the maximum allowed speed for a vehicle running in the road which will be used to not only to calculate the route but also compute the simulation objects because all the way has the driving direction and lane information. We split way and corresponding lane count by the direction label.

4. Format Segment Links to Simulation Links

   In this step, we use the segment link direction angle to make a distance bias and thus calculate the lane coordinates. The direction angle means the angle between the edge and x positive axis.

5. Build the Road Network Graph

   Formatting the vertices from node dataset and structuring the link to the edge, apply these vertices and edges to graph build API from GraphX. The graph will be used to generate trip data, computing the trips and run the simulation.

   Figure 4.5 shows how to convert way into segment link. In the diagram, way1 and way2 come across, and there are five nodes. Node2 is the shared node and intersection for way1 and way2. The way1 has three nodes in the diagram, node1, node2 and node3 with four lanes and 45 miles per hour maximum speed. We slice way1 into 4 segments, s1 from node1 to node2, s2 from node2 to node3, s3 from node3 to node2 and s4 from node2 to node1, mark each segment with 2 lanes and 45 mph maximum allowed speed, calculate distance between the node and linear interpolate the points between the head node and tail node.

If the number of nodes is N and the number of ways is M, the time complexity

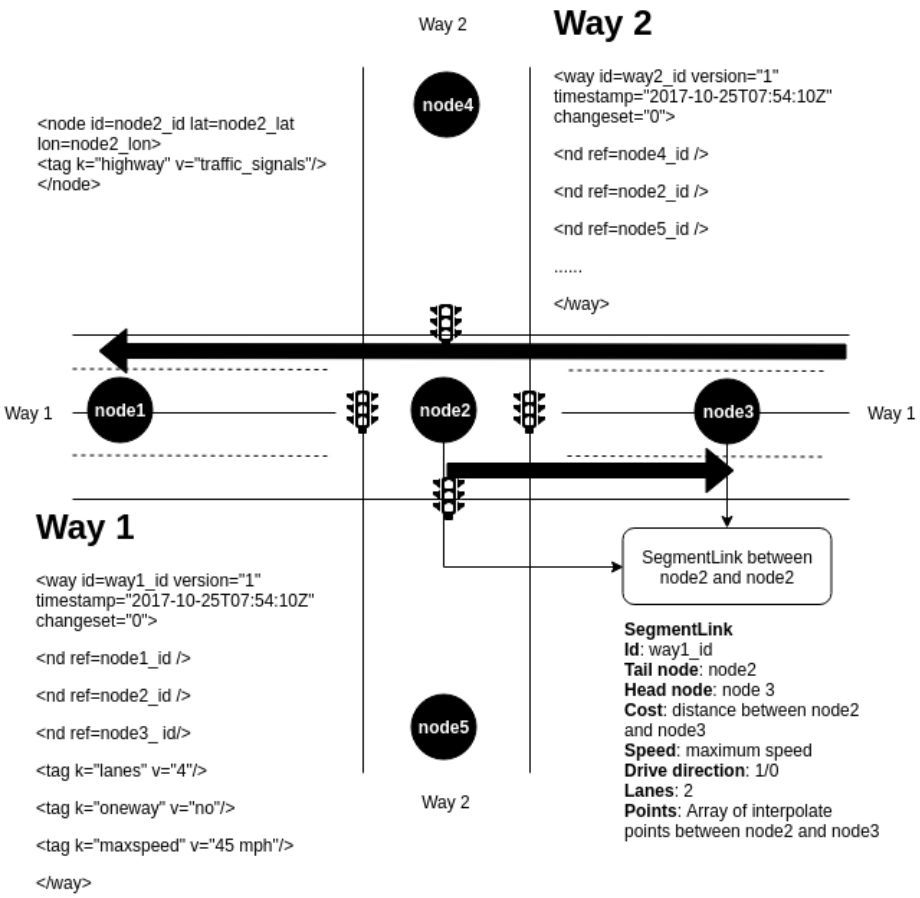


Figure 4.5: Convert Ways to Segments

Algorithm 1: Road Network Converter

Data: rawnodes, rawways

Result: nodes, links, lights

1 Filter light nodes from rawnodes;

2 Explore and flat a sequence of nodes from rawways and format into ways;

3 Join light nodes with explored ways by node ID and format into wayDF;

4 Convert wayDF to segmentRDD by segment node sequence;

5 Build lane coordinate reference system and convert segmentRDD into linkRDD;

6 Filter light node from linkRDD and format into signalRDD;

7 **return** *Road Network(nodeRDD, linkRDD, lightRDD)*

of the Algorithm 1 is $\mathcal{O}(N * M)$ because of the join operations between nodes and ways. If the number of partition is P, the time complexity is $\mathcal{O}(N * M/P)$ because the operations run in parallel.

NodeRDD. NodeRDD contains all needed vertexes from an arbitrarily selected region, and each vertex is a road junction that connects two road segment links. Each vertex has three attributes (1) ID: the unique ID of this vertex (2) location: the spatial coordinate of this vertex (3) type: a vertex can be on a highway or residential street. It may have traffic lights or being an uncontrolled intersection.

LinkRDD. LinkRDD accommodates all necessary edges, and each edge is a road segment which is a straight way between two vertexes. Each edge consists nine attributes (1) the ID of the way (2) way source vertex (3) way destination vertex (4) total length of the edge (5) speed limit (6) drive direction (7) lane count (8) direction angle (9) lane geometries.

SignalRDD. SignalRDD includes all signal vertexes in the simulation region. Each signal contains three main attributes (1) the node ID (2) the controlled way ID (3) coordinate.

Chapter 5

VEHICLE

The moving agent in GeoSparkSim is a different type of vehicles, such as truck and car which has different characteristics and moving attributes. For instance, the truck has longer length and higher speed limits than the compact car. These vehicles play a fundamental role in construct traffic in the road network. This chapter will introduce the algorithm to initialize vehicles, compute the shortest path, seed up it in multithreading and convert to vehicleRDD.

5.1 Initialization

The vehicle model in GeoSparkSim is time-continuous coordinates list. A vehicle leaves a place (coordinate) at a particular time (step), follows the path step by step and disappears when reaching the destination (coordinate). The first step is to initialize these parameters.

Source Coordinate. There are some existing approaches to generate the moving objects source node, such as the data-space oriented approach (DSO) and network-based approach (NB) proposed by professor Brinkhoff (2002). The DSO approach generates source points based on a specific spatial distribution and run map matching to match points to their nearest nodes in the road network. This spatial distribution can be the density of the human population or the thickness of buildings. Regions with high density will produce more sources node. The NB approach randomly selects road junctions as sources. GeoSparkSim provides these two options for the user.

Route Length. For observation purpose, the route length should not be too short or very long. We define a minimum route length 1 kilometer, and the maximum route

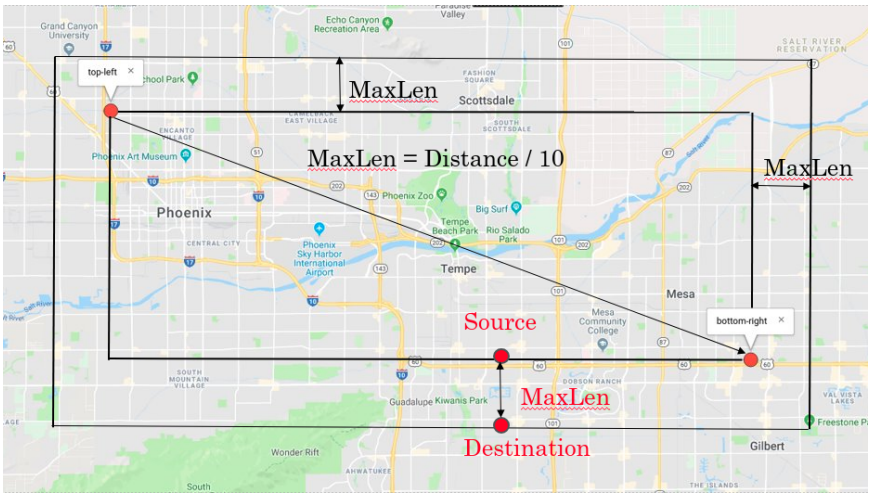


Figure 5.1: Vehicle Initialization

length is the length of the diagonal line from the selected region divide by 10. If the region is smaller than the defined width, the minimum and maximum value will scale as well to meet the requirements. Figure 5.1 is an example of the relationship between the region and vehicle initialization. The region is bounded by top-left and bottom-right coordinates and all the source node is generated in this region. GeoSparkSim will create a region buffer with the maximum length in the following equation to hold the destination node.

$$maximum\ length = distance(top - left, bottom - right)/10$$

Destination Coordinate. Destination Coordinate is computed by source coordinates, route length and a randomly generated angle [0, 360]. Route length is random generate in the range mentioned in the last paragraph. Assume GeoSparkSim in Cartesian Coordinate System, like Figure 5.2, the source is the center of the circle and the radius is the route length. Randomly generate the center angle and calculate the destination by the equation.

$$longitude = source.longitude + length * sin(angle)$$

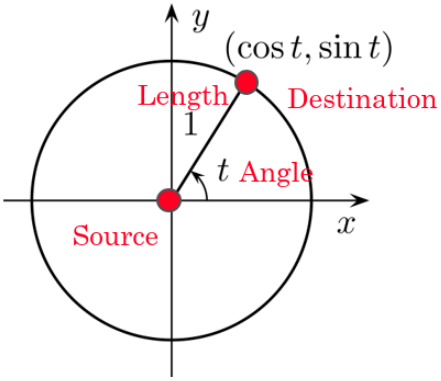


Figure 5.2: Souce Destination Cartesian Coordinate System

$$latitude = source.latitude + length * cos(angle)$$

Vehicle Attributes. Vehicle extends JTS (nd) LineString and each vehicle has three components: vehicle, intelligent driving mode and lane-changing model. The JTS is a Java Library for creating and manipulating geometry.

1. The vehicle contains attributes with type in Table 5.1. The ID is the unique identification of vehicle created by a random combination with length 5 of uppercase characters and number 0-9. The source and target coordinates are generated in the method mentioned above. Edgepath is an array of edge id (OSM way id) where the vehicle will stop by. The cost for each edge is stored in the cost array. The full path is the trajectories of the vehicle which expected path is the temporal path storing the LineString coordinates. Front coordinate represents the vehicle head while rear labels the vehicle back position. Edge index indicates the current edge from edge array where the vehicle is running. Vehicle attributes is a representation of the vehicle's basic information. Vehicle encapsulates as an individual class and the user could customize it to represent a different kind of vehicle.

Attribute	Type
ID	String
Source	Coordinate
Target	Coordinate
Edge Path	Long Array
Costs	Double Array
Full Path	Coordinate List
Front	Coordinate
Rear	Coordinate
Edge Index	Integer
Car Length	Double
Current Lane	Integer
Is Arrive	Boolean
Current Link	Link

Table 5.1: Vehicle Attributes

2. Intelligent Driving Model (IDMVehicle) incorporates several parameters to control the driving behavior of a vehicle showing in Table 3.2. Start position is the position in the current lane. Head signal keep tracks of the nearest signal position in the vehicle's path. During the simulation, every vehicle moves along its planned route over and over, but each time it may stop at different traffic lights, run in separate lanes and generate various acceleration/deceleration events based on IDM. The detailed movement illustration will be the next chapter.

3. Lane-changing Model (MOBILVehicle) consists of several attributes required

Attributes	Value or Type
Acceleration	2.5 (m/s^2)
Brake Deceleration	3 (-m/s^2)
Start Position	0 (meter)
Safe Distance	3 (meter)
Default Speed Limit	17.88 (m/s)
Current Speed	Double
Head Signal	Traffic Light

Table 5.2: IDMVehicle Attributes

Attributes	Value or Type
Politeness Factor	0.3
Maximum Safe Deceleration	4 (-m/s^2)
Threshold Acceleration	0.4

Table 5.3: MOBILVehicle Attributes

Table 5.3 to calculate the lane change criterion. The details of MOBIL will be discussed in the next chapter.

GeoSparkSim first initializes the status of vehicles which will generate a trip source and a destination for every vehicle such that the vehicles will move from their sources to destinations during the simulation. Their specific routes will be generated in the next step.

5.2 Route Planning

After vehicle initialization, route planning layer read road network, process it with graph structure and compute the shortest path of each vehicle. GeoSparkSim explores three engines to computes the path and decides to leverage Graphhopper to do route planning. This sections will discuss route planning approach on these three engines and discuss the pros and cons for these three approaches.

5.2.1 Open Source Routing Machine

Open Source Routing Machine OSRM (nd) is an open source application with solutions to find the shortest path in the OSM road network. OSRM is implemented in C++ and developed two preprocessing pipelines: contraction hierarchies (CH) Geisberger *et al.* (2008) and multi-level Dijkstra (MLD) Holzer *et al.* (2009). Contraction hierarchies is a technique to speed up the shortest path finding process by precomputing the short cut between graph connection. Multi-level Dijkstra is based on the standard Dijkstra algorithm with preprocessed multiply graph layers, such as highway layer. OSRM provides back-end, front-end and docker image for the user to utilize the routing engine. Before querying the path, OSRM requires the user to do road network data preprocessing. For example, if you want to calculate the shortest path from ASU to Phoenix, you need to download OSM data and run preprocessing command from OSRM. Unfortunately, this preprocessing takes a huge chunk of time. For instance, the Arizona state OSM road network data takes around two hours to do preprocessing. Even the routing performance is close to Graphhopper Section 5.2.3, OSRM unable to do faster preprocessing from raw data and thus is unsuitable for real-time traffic simulation.

5.2.2 Apache GraphX

GraphX implements many classic graph-related algorithms, wrap it with API and thus provides a set of Apache Spark APIs for parallel graph computation. Rather than the link cut from TRANSIMS mentioned in Chapter 2, GraphX partitions graph by intersection. TRANSIMS do graph decomposition more from simulation side since they think the simulation scenarios are more complicated in the intersection than links. But GraphX considers it more from graph algorithm side, like message passing. Graph traversal is a typical process in the graph, and it will visit the node in the graph in specific ways, such as Depth First Search(DFS) and Breath First Search(DFS). The message need to be computed will pass from node to node by the connecting link, like path message.

Chapter 4 discussed the conversion from the road network to vertexRDD and edgeRDD. We build the graph in Spark GraphX and develop the Dijkstra algorithm in GraphX pregel Malewicz *et al.* (2010). Figure 5.3 shows an example of the road network graph on GraphX. Vertex table is the table representing vertex information in which GraphX uses the ID to identify and make communication. Edge table is regarded as the table to maintain the directional graph relationship by pointing srcId, DstId and corresponding property. Pregel API is an implementation of bulk-synchronous message-passing API, especially for graph computing. Pregel API enables message computation over edges, read computation message both from a vertex and its attributes and iteratively passing estimate from one vertex to all neighbor edges and vertex. Algorithm 2 is the pseudocode of Dijkstra implementation in GraphX. Initiate message with destination ID, infinite costs and an empty list to store the ongoing visite node. Utilized pregel API, graph passing message from a source to all neighbor vertex, calculate and update costs. In the end, merge the results in

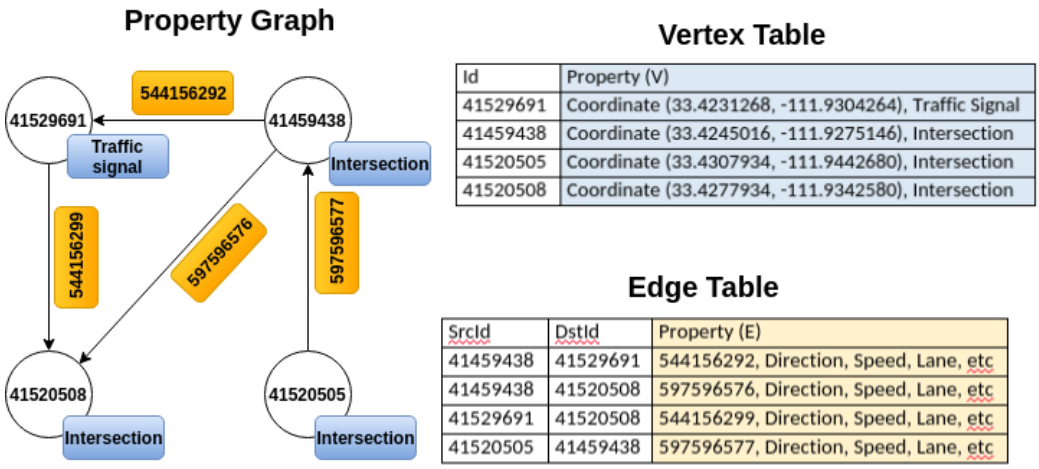


Figure 5.3: A Road Network Graph Example in Apache GraphX

the graph. Unfortunately, without advanced graph preprocessing and indexing, the Dijkstra algorithm implementation in Pregel GraphX over a large road network is very slow. For example, compute the shortest path from ASU brickyard to Walmart in E Southern Ave over Tempe, AZ area takes around 1 minute to complete. Because of the limitations, GeoSparkSim requires real-time quick route planning and GraphX is not a good idea.

5.2.3 GraphHopper

GraphHopper Karich and Schröder (2014) is an open source route planning library to compute the shortest path for every source and destination pair. Graphhopper supports various routing algorithms, such as Dijkstra and A star that provides three modes for the user depending on different purposes, such as speed mode, flexible mode, and hybrid mode. Speed mode preprocesses road network to contraction hierarchies, and the flexible mode comes without CH. Graphhopper supports many different types of input, such as OSM, Shapefile, Navteq, etc. We use Graphhopper to preprocess OSM and overwrite the API to return the data GeoSparkSim needs, such

Algorithm 2: SSSP Dikstra Implementation in Pregel GraphX

Data: Road network graph, source Id, destination Id

Result: Shortest Path

```
load road network graph;
val initialMsg = Route(destId, Infinity, List())
val spGraph = graph.mapVertices{ (vid, vertex) =>route }
//Execute Dijkstra
val pregel = Pregel(spGraph,
    initialMsg
    spGraph.vertices.count(), //Maximum Iterations
    EdgeDirection.Out)(
    vprogf = (id, msg) =>route with minimum cost //Update
    sendMsgf = triplet =>compare costs //Compute Msg
    combinef = (m1, m2) =>msg with minimum cost //Combine msg
    )
    pregel.filter(destination == last Id in list).map(route)
return Route from source id and destination id
```

as edge path. Compared to Open Street Routing Machine, Graphhopper has much less preprocessing time. For example, processing Arizona OSM data takes around two hours in OSRM while Graphhopper takes a couple of minutes to complete. Moreover, Graphhopper takes less route computing time than the implementation in GraphX.

For the sake of routing speed, GeoSparkSim leverages GraphHopper Karich and Schröder (2014). GeoSparkSim first uses GraphHopper to build an index over the imported road network. This index pre-computes short paths among common road junctions. Then GeoSparkSim queries the pre-built index to calculate the route for every vehicle.

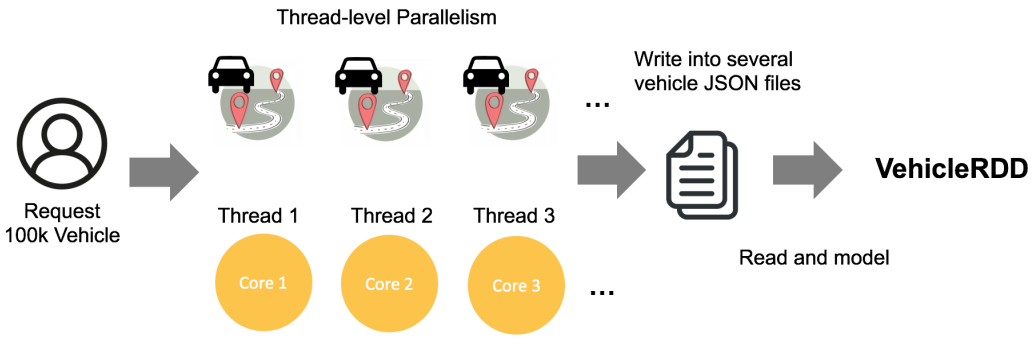


Figure 5.4: Vehicle to VehicleRDD

5.3 Parallel Vehicle Generation

Graphhopper is a routing machine focusing single shortest path algorithm. In order to speed up the shortest path computing process and take most of the CPU computing resources, GeoSparkSim parallel process the requests in multi-threads.

By default, GeoSparkSim uses a thread pool with eight threads to generate the vehicle and compute the shortest path in parallel. Each thread contains a spatial coordinate generator and a routing engine. Figure 5.4 is an example to generate vehicles in parallel. User requests simulation with 100,000 vehicles. GeoSparkSim will divide the number of the vehicle to 8 threads. Each thread sequentially produces vehicle status and compute the shortest path. When the thread finished route requests, GeoSparkSim will gather results in a list of vehicle and convert to vehicleRDD. After all the requests finished, the thread pool will be shut down immediately.

5.4 VehicleRDD

GeoSparkSim VehicleRDDs are in-memory distributed datasets that extend traditional RDD to accommodate vehicle objects in Apache Spark. VehicleRDD is constructed by a set of vehicles with its randomized driving models and status such

that yields arbitrary trajectories. Each VehicleRDD consists of many partitions and each partition contain thousands of vehicles. GeoSparkSim performs simulation in each partition and these partitions compute simulation in parallel by distributing the VehicleRDDs across the cluster.

VehicleRDD transformation. To simulate the traffic of numerous vehicles in a specific period, GeoSparkSim generates GPS locations of these vehicles for every simulation time step. For instance, if the period is one day and the simulation time step is 1 hour, GeoSparkSim will take a snapshot of the traffic every hour from 0:00 am to midnight. To achieve that, GeoSparkSim first creates an initial VehicleRDD, and all vehicles stay at the origins of their routes. Then it keeps transforming the VehicleRDD via a map operation. Each RDD transformation will compute the new running status of vehicles according to their driving models. Every transformation produces a new VehicleRDD based on its ancestor VehicleRDD. The running status computation uses microscopic simulation models and will be detailed in Chapter 6. In other words, a VehicleRDD is a snapshot of current vehicle movements over the road network.

Chapter 6

SIMULATION

This chapter will explain the driving behaviors and driving models being considered in GeoSparkSim. Spatial distribution and workload balancing have a significant impact on simulation which will be discussed in this chapter. Also, the simulation algorithm, partition strategy, and parallel simulation computation will be explained in this chapter.

6.1 Driving Behaviors

Driving behaviors describes the moving objects in the simulation. GeoSparkSim is a microscopic traffic simulator, in which the single vehicle-driver behaviors are characterized by all the simulated vehicle in a given time. For example, the vehicle change lane where there is no obstacles side by side, accelerate when there is no close vehicle ahead and stop when the light is red. GeoSparkSim is a microscopic traffic simulator simulating single vehicle-driver units, so the dynamic variables for the model represent the moving object behaviors, like the position, change lane, velocity change, brake, stop, etc. GeoSparkSim is a time-continuous simulator following the car-following Gipps (1981) and lane-changing Kesting *et al.* (2007a) model that describes the dynamic driving behaviors in ordinary differential equation(ODE).

- **Keep Safe Distance to Surrounding Objects.** Based on the vehicle trajectory, GeoSparkSim will check next following road to determine the nearest front object, vehicle or traffic signal. Get the relative speed; the vehicle will keep a safe distance between the object and make corresponding actions. Because sometime the vehicle may have randomized driving behaviors, a checking buffer

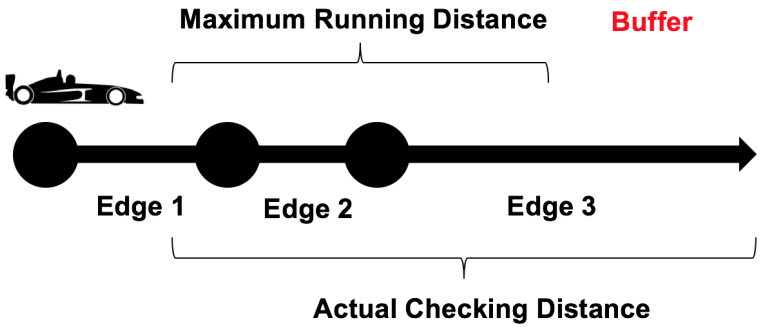


Figure 6.1: Safe Distance Check

is added to a safe distance check. Figure 6.1 shows the example of checking buffer.

- **Traffic Signal Intersection.** In the intersection with a traffic signal, GeoSparkSim assigns the same signal when two roads are in the same angle. During the simulation, the simulator will update the signals, and the vehicle will respond to corresponding lights. Vehicles keep the same passing speed at a green light, decelerate at a red light and check the safe distance to pass at a yellow light.

- **Moderate Speed in Every Step.** In each simulation steps, the vehicle will check the heading objects and moderate speed to keep a safe distance. Next section will have a detailed introduction of the model.

- **Change to More Safety Lane.** GeoSparkSim simulate vehicle in the multi-lane road network. The simulator will switch the vehicle to less traffic lane to distribute traffic and simulate the real scenarios. Next section will introduce the lane change mode.

On each partition, GeoSpark runs a generic simulation algorithm which allows pluggable microscopic traffic models.

Car-following. GeoSparkSim adopts the Intelligent-Driver Model(IDM) Kesting *et al.* (2010b)to update the speed, position, and gap. The bumper-to-bumper gap is the distance between the vehicle and the front vehicle. IDM is the model decides the acceleration and moderate speed every timestamp in simulation, like accelerations and braking deceleration of the drivers. The influencing factors of IDM include desired speed(v0) referring to the maximum speed of the street, headway time(T), acceleration factor(a), braking deceleration factor(b), minimum bumper-to-bumper distance(s0) to the front vehicle and acceleration exponent(delta). The following is the IDM model equation. The car-following model helps GeoSparkSim better describe the microscopic driving behaviors and define the vehicle's location more precisely without overlaying other objects. If nearby vehicles are within the safe distance buffer, this model will make the vehicle decelerate. If there are no nearby vehicles, the model may accelerate the vehicle.

$$\frac{dv}{dt} = a[1 - (\frac{v}{v0})^{\delta} - (\frac{s^*(v, \triangle v)}{s})^2] \tag{6.1}$$

where

$$s^*(v, \triangle v) = s_0 + max[0, (vT + \frac{(v\triangle v)}{2\sqrt{ab}})] \tag{6.2}$$

Lane-changing. GeoSparkSim uses a general lane-changing model called MOBIL Kesting *et al.* (2007b). Safety criterion and a specific probability will trigger lane change. For instance, if the vehicle changes the lane and can get a more safety

context to surrounding objects, the lane change will happen. Otherwise, it will stay in the same lane.

Some lanes information is from OpenStreetMap and default lane is bidirectional with one lane each direction. Taking the acceleration decision from IDM, the MOBIL calculates if the targeted new lane is allowed. The following equation is MOBIL which includes politeness factor (p), maximum safe braking deceleration(b_{save}), threshold(a_{thr}) and bias to the right lane Delta b.

$$acc'(M') - acc(M) > p[acc(B') - acc'(B')] + a_{thr} \tag{6.3}$$

where

$$acc'(B') > -b_{save}$$

Traffic lights. GeoSparkSim assigns initial signal randomly, and it exactly follows the green-yellow-red sequence. The time duration for green light is 55 seconds, yellow light 5 seconds and red light 60 seconds. When a traffic light appears in the safe distance of a vehicle, GeoSparkSim will check the status of this light. The speed of this vehicle will be changed to 0 right away if the signal is red. If the light becomes green in the next time steps, the vehicle will start to accelerate.

6.3 Simulation Architecture

Figure 6.2 is the general simulation architecture in GeoSparkSim. Chapter 4 and chapter 5 explained the generation of LinkRDD, SignalRDD and VehicleRDD. Considering a user issues a simulation request for 10 minute and 1 second per step, 600 simulation steps will be computed. In each simulation step, GeoSparkSim first calculates signal timing and update the next signal and then loop all the vehicle calculating the next movement. For each vehicle, GeoSparkSim will check nearby objects

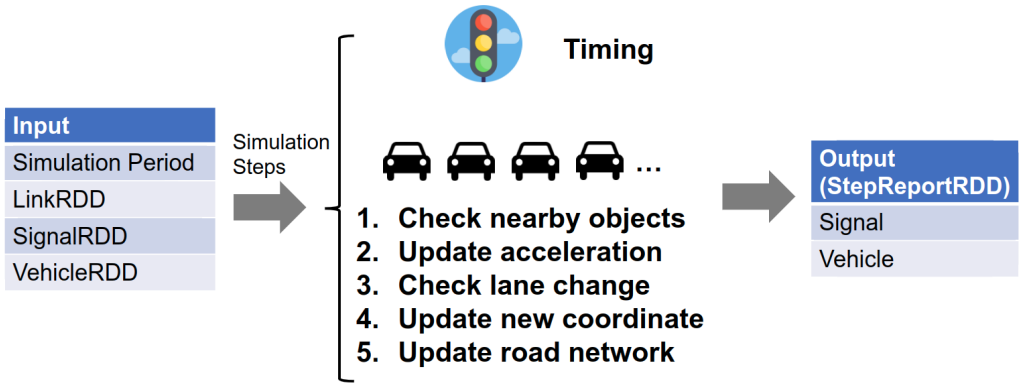


Figure 6.2: Simulation Architecture

following the trajectory. Based on these objects and their relative status, update new acceleration event and moderate speed. Calculate the possible lane to change and compute new coordinate by the speed, simulation interval, and lane coordinate reference. Finally, remove the vehicle from the last position and add it to the new road. Each Simulation step will generate new signal and vehicle events, and these events will be reported as the StepReportRDD output. A user can use the result for visualization or analytic.

6.4 Distributed Simulation

The simulator will iteratively run simulation steps calculated from period and time per step and compute results. GeoSparkSim is developed as a distributed system and implemented on Apache Spark following a master/slave architecture that requires one master process and one or more slave processes, referred to as workers. The master is responsible for data preparation, simulation management, partition, and visualization. The worker is responsible for process data, compute driving models and update simulation objects. The master and worker can be run in the same machine and serve the corresponding jobs. Master and workers can be configured in

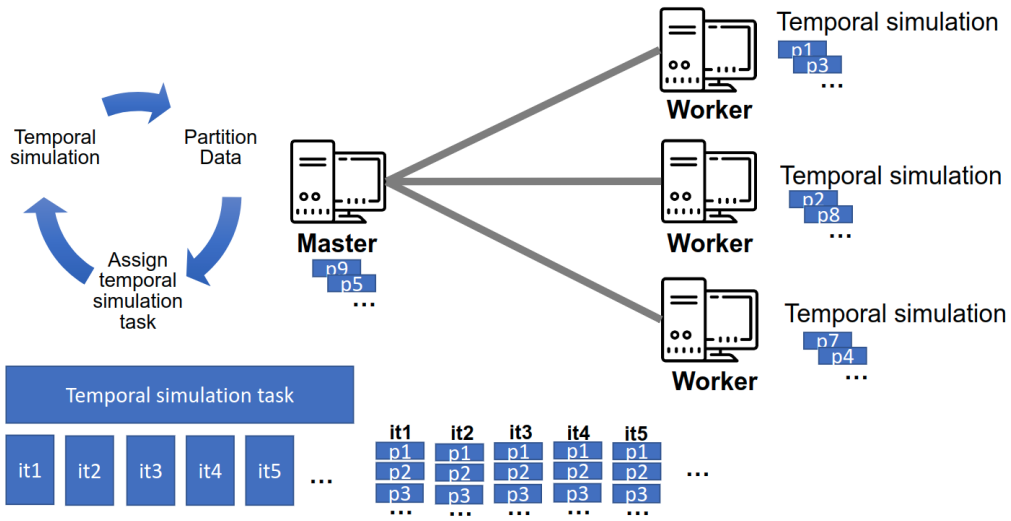


Figure 6.3: Distributed Simulation

internet connected computers across the cluster by Apache Spark. The master is able to communicate with all the slaves in cluster and workers only follow the master's commands.

Figure 6.3 is the simulation work distribution in GeoSparkSim. Temporal simulation is a partial period of simulation and will be explained in Section 6.7. General speaking, a simulation task has many iterations. For example, ten minutes simulation and one second per steps have 600 iterations. Current iteration simulation computation is depended on the last iteration results. In each iteration, GeoSparkSim distributes works to workers and each worker-run simulation in parallel. Since workers parallel compute the traffics, the simulation time can be significantly reduced compared to run it in a single worker. How to distributes workload will be discussed in Section 6.5 and the workload balancing approach will be introduced in Section 6.7.

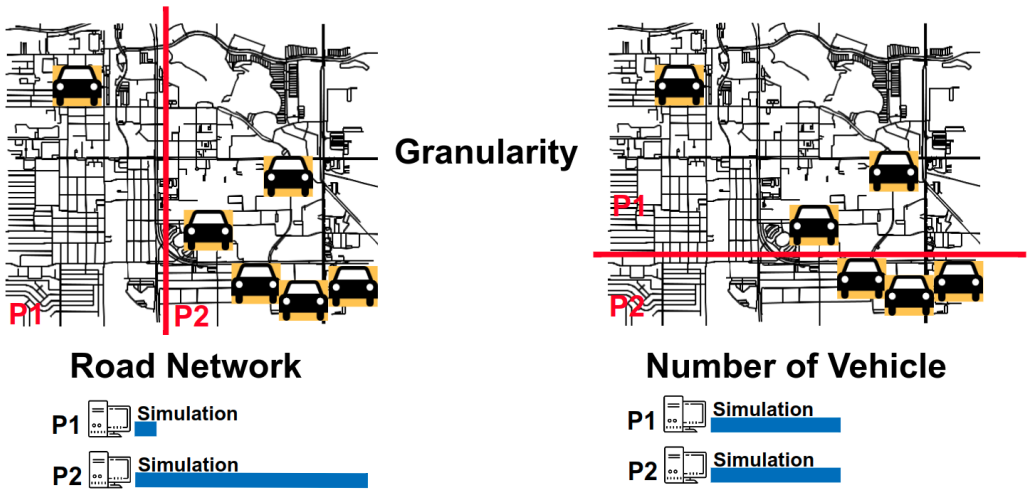


Figure 6.4: Spatial Workload Distribution

6.5 Spatial Workload Distribution

GeoSparkSim is a distributed simulator, and the workload distribution in the cluster has a significant impact on performance. For example, if two workers are running the simulation at the same time, the total simulation time depends on the worker who takes the longest time to complete. Balanced workload helps system to make full use of computation resources. If the workload is unbalanced, all the workers need to wait to process the next steps, and at that time these workers do nothing and perform idle.

GeoSparkSim partitions simulation works by the spatial attributes, like coordinate and trajectory. Granularity describes the simulation details and density of a partitioning grid, such as streets, traffic lights, vehicle movement and lane change which is a representation of simulation workload. Figure 6.4 is a comparison between partition simulation by the road network and the vehicle density. The background network describes the road network granularity, and the yellow vehicle represents the vehicle granularity. These two figures are on the same road network and have equal

vehicle distribution. The left diagram shows the road network partition plan. Partition 1 (P1) and partition 2 (P2) have the same road network density. However, the number of vehicles in P2 is larger than P1. Simulator computes vehicle movement in each step, and the major workload is related to the vehicle; thus P2 have heavier simulation workload than P1, and when P1 finished works, it will cause an idle time for P1 to wait for the completion of P2. It is inefficient to partition simulation by road network because of the unbalanced workload issue. The right figure shows the approach partition simulation by vehicle. P1 has a larger region area size than P2, but the vehicle density is the same in two grids. This approach tries to balance simulation workload which will have benefits on computation resources.

6.6 Spatial Resilient Distributed Dataset (SRDD)

Spatial partition is the process to divide a space into several regions. This section will introduce three types of spatial partition tree, Quad-tree, KDB-tree, and R-tree. Spatial Resilient Distributed Dataset extends Spark RDDs with spatial objects that efficiently partition spatial data across Apache Spark cluster, and GeoSparkSim wraps all simulation objects into SRDDs in the cluster that can efficiently build a spatial index and distribute the simulation work in the cluster by the spatial proximity.

6.6.1 Spatial Partitioning Tree

Spatial partition system partitions space into many parts by the spatial proximity and this subspace will be organized into a tree, called a space-partition tree. A Quad-tree Finkel and Bentley (1974) is a tree data structure in which each internal node has exactly four children. A KDB-tree Robinson (1981) (k-dimensional B-tree) is a tree data structure for subdividing a k-dimensional search space and provide the search efficiency of a balanced k-d tree. An R-tree Guttman (1984) is a tree data structure

building multi-dimensional index for spatial objects.

6.6.2 SRDD Partitioning

GeoSparkSim creates three simulation objects, vehicle referred to as Point, link referred to as LineString and signal as Point. Coordinate represents the Point in the spatial system, and LineString represents the line segment with at least two coordinates. GeoSparkSim loads raw data into memory physically splits its in-memory (hashing partitioning) and distributes an equal number of partitions to each worker node. This kind of partitioning doesn't consider the spatial proximity which is essential for simulation-related analytic. For example, a vehicle trajectory is a sequence coordinates following spatial proximity. Also, all the simulation objects combined by their spatial proximity. With the increasing complexity of simulation and transportation system, spatial proximity preserves the spatial relationship which is crucial for zipping spatial objects together.

The partition algorithm first builds a global grid file at the master node. Spatial RDDs represents a vastly distributed dataset, and it is very time-consuming to construct a partitioning tree on the original RDDs. The algorithm takes a sample from the original RDDs and builds the selected spatial structure on the collected sample on the master. The grids are retrieved from the built spatial structure and if the spatial object intersects

1. **Build a global spatial grid file.** The spatial objects are loaded and split by spatial distribution to achieve load-balancing. In these steps, the sample is taken from each subset and the sample keeps the spatial proximity. After that, a spatial partitioning tree is used to split the sampled data into partitions at the Spark master node.

2. **Assigning a grid cell ID to each object.** After building a global grid file, the original Spatial RDDs need to be repartitioned by the spatial structure. The global grid file will be broadcast to each Spatial RDD with grid ID and each Spatial RDD will begin to check the internal objects against the grid file. A new Spatial RDD will be created to store the results with key-value pair schema. If the object intersects or spans across with several grids, the duplicated object will be created in each grid.

3. **Re-partitioning SRDD across the cluster.** The key is the grid ID and the value is the object. In this step, Spatial RDD is repartitioned by key and the object with the same key are grouped into the same partition. The data will be shuffled and partitioned across the nodes in the cluster.

6.7 Spatial Workload Balancing

Simulation measures the dynamic object events in transportation system over time. The last section discussed the importance of vehicle for simulation and this section would discuss the spatial simulation-aware partitioning approach in GeoSpark-Sim which try to make equal simulation workload for each partition in the cluster over time. The vehicle life cycle is beginning with source coordinate, travel path coordinates and destination coordinate. Initially, the workload could be distributed by the vehicles' source coordinate. But the vehicle movement may cause spatial distribution different than the beginning. Conclusively, the spatial distribution in simulation dynamic changes over time.

6.7.1 Partition by Vehicle Source Coordinate

Consider if simulation workload is spliced by the distribution of vehicle source coordinate, Figure 6.5 shows the partitioned road network consisting three grids, par-

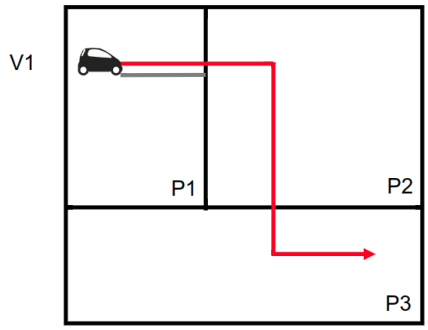


Figure 6.5: Partition by Vehicle Source Coordinate

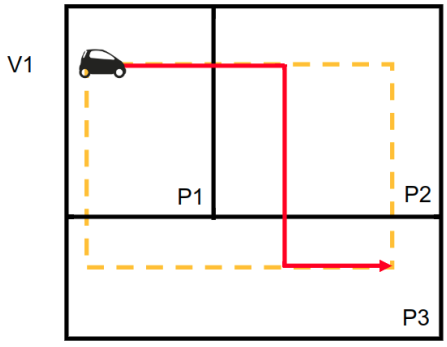


Figure 6.6: Partition by Vehicle Trajectory

tition 1 (P1), partition 2 (P2) and partition 3 (P3). As the partition is based on vehicle, the area size of these partitions is different. Vehicle 1 (V1) will move following the red line and dynamic update the locations over the simulation. If the vehicle reaches the boundary between P1 and P2, a message exchange process is required to continue to simulate V1 which will lead to additional message exchange costs. In addition, this approach doesn't consider the dynamic changed spatial distribution since it maintains the same partition all the time which will cause unbalanced workload with increased simulation steps.

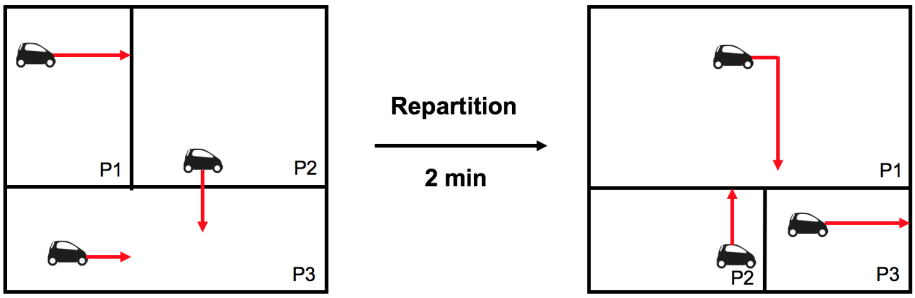


Figure 6.7: Simulation-aware Partitioning

6.7.2 Partition by Vehicle Trajectory

Assume the simulation workload is partitioned by the vehicle trajectory and Figure 6.6 displays the partitions. The trajectory is a collection of coordinates vehicle will follow in simulation and a minimum bounding rectangle (MBR) will be created by the coordinates, the yellow dash rectangle in the figure. GeoSpark provides an approach to do spatial partitioning with a spatial partitioning tree. The trajectory MBR may go across several partitions and will have a copy in each of them. This means the vehicle will compute three times in total if the simulator computes traffic in parallel. This approaches may have significant overhead and a waste of computation resources in the cluster.

6.7.3 Simulation-aware Partition

GeoSparkSim proposed a simulation-aware partition (SP) approach trying to balance the workload distribution over time. SP partitions initial simulation workload by the source coordinates and periodically repartition by the new vehicle location. Figure 6.7 is a SP illustration. Redline is the temporal simulation path. For example, if simulator repartition workload in every 2 minutes, the temporal simulation is 2 minutes. In the figure, GeoSparkSim dynamic update partitions over the time to

achieve spatial workload balancing and make full use of computation resources.

The repartition period is dynamic determined by the simulation precomputation. GeoSparkSim takes samples from VehicleRDD, LinkRDD and SignalRDD, distributed simulate samples in different repartition period, and records the period with the minimum execution time. The sample keeps the same ratio against the original data, and the spatial distribution of the samples is exactly same as the original data. By doing that, the repartition period is more flexible to the simulation period, and GeoSparkSim always tries to give the best repartition strategy with the minimum overhead.

6.8 Simulation Algorithm

In repartition criterion Algorithm 3, GeoSparkSim takes 1% samples from vehicleRDD, LinkRDD and SignalRDD. Segmenting five repartition periods from the simulation period and check the best repartition period with best workload balancing and minimum simulation time. Algorithm 3 flexible determines the best repartition period with minimum computation overhead.

GeoSparkSim set a repartition time and run temporal simulation iteratively which will shuffle the vehicleRDD, LinkRDD and SignalRDD in the simulation. If the simulation period is too short, like 2 minutes, GeoSparkSim will initially partition the vehicle once by coordinates, otherwise, periodically do repartition.

In simulation Algorithm 4, GeoSparkSim periodically do repartition for vehicleRDD and apply same partition mechanism to LinkRDD and SignalRDD. Because an entire road network contains vehicles, links, and signals, VehicleRDD, LinkRDD, and SignalRDD are zipped by objects' spatial proximity. After partitioning the RDDs, GeoSparkSim is ready to run the local microscopic simulation on each RDD partition. All vehicles will follow their trajectory in this temporal partition. Every route

starts from the last location of the vehicle. This algorithm first calculates the number of GPS locations(step) needed to be simulated for every vehicle in this temporal partition. This number can be easily computed via the following equation:

$$locations\ per\ vehicle = \frac{temporal\ partition\ size}{time\ step\ size}$$

where time step is the granularity of simulated trajectories (say, 1 second). It also indicates the number of simulation iterations needed to run by GeoSparkSim. The algorithm then runs a set of iterations, and in each iteration, it first computes all signal next timing and update light and calculate every vehicle movement. The vehicle will search the nearest objects, like vehicle or traffic light ahead from links. In each local simulation, links will be initialized to a Map with way ID key and link object value. Vehicle trajectory contains the way information, such as coordinates, distance costs, and way ID. The vehicles are able to get link object by way ID from links map. In order to avoid collisions, GeoSparkSim computes the relative speed to nearest objects and update the next acceleration. Calculate the following position in the lane by time step and acceleration and remove the vehicle from the last place. GeoSparkSim also considers multi-lanes roads and will check the possible lane change opportunities and update vehicle to the new lane. The new coordinate calculated by the lane coordinate reference and the position in the road lane. Finally, update the vehicle in the link map. After the local simulation on each RDD partition, GeoSparkSim will update VehicleRDD and SignalRDD status and persist the simulation results on HDFS.

If the number of simulation iterations is N, E street links, V vehicles, S traffic signals and considering the microscopic simulation model computation is a constant time cost C, the time complexity of Algorithm 4 is $\mathcal{O}(N * (S + CV))$. Because GeoSparkSim distributed run traffic simulation, if the number of spatial partition is

P, the time complexity of the algorithm is reduced by P which shows in the following equation.

$$time\ complexity = \frac{N * (S + V * C)}{P}$$

Algorithm 3: GeoSparkSim Repartition Algorithm

Data: Iterations, VehicleRDD, LinkRDD and SignalRDD

Result: BestRepartition

```
Initialize minTime and bestRepartition;
for repartition ← iterations/10 to iterations/2 do
    Initialize time;
    Run GeoSparkSim With VehicleRDD, LinkRDD and SignalRDD Samples;
    if time <minTime then
        bestRepartition = repartition ;
return Best repartition period
```

Algorithm 4: GeoSparkSim Simulation Algorithm

Data: Iterations, VehicleRDD, LinkRDD and SignalRDD

Result: StepReportRDD

```
Initialize SignalRDD and LinkRDD;
foreach temporalpartition in iterations do
    Partition VehicleRDD by current location;
    Apply same partition to LinkRDD and SignalRDD;
    Zip VehicleRDD, LinkRDD and SignalRDD by spatial proximity;
    foreach partition in zipped RDDs do
        foreach iteration in temporal simulation do
            Initialize vehicles in links;
            foreach Signal S do
                Update S timing and light;
            foreach vehicle V do
                if V not arrive destination then
                    Search the closest vehicle and traffic signal ahead;
                    Compute V acceleration and velocity;
                    Remove V from links;
                    Compute V position P in lane;
                    Check possible lane change;
                    Compute coordinate by P and new lane;
                    Update V and add to links;
                else
                    reborn V;
    Update vehicleRDD and SignalRDD by the latest status;
    Write steps reports;
return Simulation results
```

Chapter 7

EXPERIMENT

7.1 Preprocessing

This section focuses on the data preparation experiments, and it contains two parts, road network graph preprocessing and vehicle generation. When we do traffic simulation, a user will issue the request from UI; the corresponding road network will be download and then do vehicle generation and simulation.

Experiment Setting. Because SUMO is not scalable, the preprocessing experiments are on Apache standalone mode. Compared to cluster mode, standalone mode runs the program in a single machine. The machine has a Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz (8 cores), 32 GB memory, and 500 GB HDD. Apache Hadoop 2.6 and Apache Spark 2.3.2.

Road Network Graph. In this experiment, we study the impact of road network graph converting. The approaches to handle the OSM road network was introduced in Chapter 4. We conduct the experiments with different road network area and comparison with SUMO road network converting process. Figure 7.1 shows the road network experiment settings and Table 7.1 performance companion for SUMO and GeoSparkSim. The area increase, the execution time will increase. By taking advan-

Road Network	Area (square mile)	GeoSparkSim Execution Time	SUMO Execution Time
Rectangle 1	325	32 seconds	1080 seconds
Rectangle 2	226	28 seconds	196 seconds
Rectangle 3	22	20 seconds	22 seconds

Table 7.1: Road Network Graph Computing Comparison

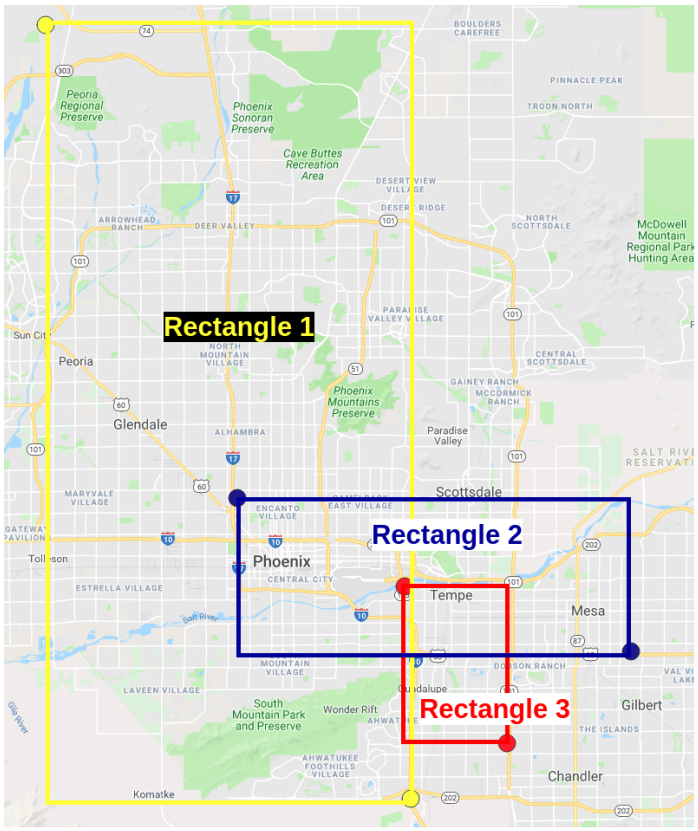


Figure 7.1: Experiment Rectangle

tages of parallel computing, the preparation process takes less time than SUMO.

Vehicle Generation. After the road network being processed, vehicles will be initialized and GeoSparkSim will generate the individual shortest path for them. The increasing number of vehicles will have a huge impact on the real-time simulation performance. In table 7.2, with more vehicles, the execution time increases. Since the application has launch time, the execution time is not a linear grow. By taking advantages of multi-threading approach introduced in Chapter 5, the vehicle generation will not have a big influence on the simulator.

Number of Vehicle	Execution Time(seconds)
10k	11
20k	16
30k	23
40k	26
50k	27

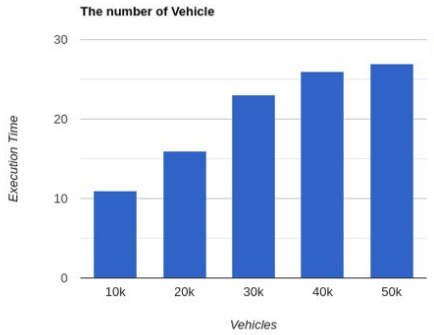
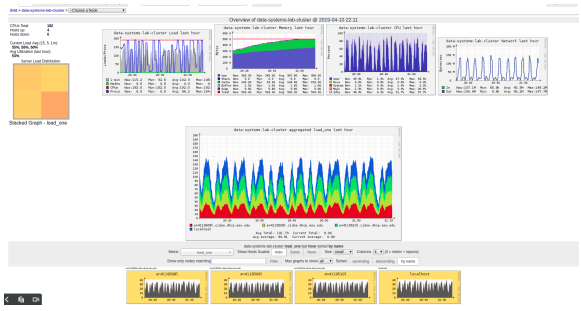
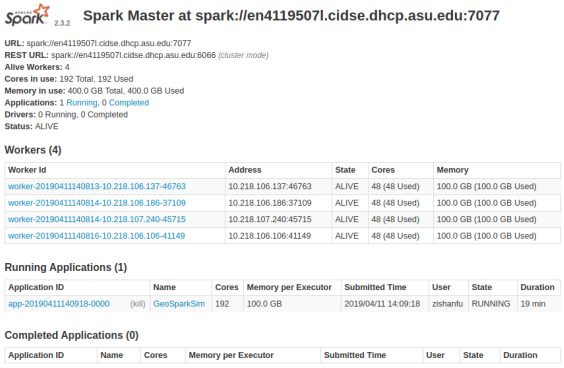


Figure 7.2: Vehicle Generation Results



(a) Ganglia



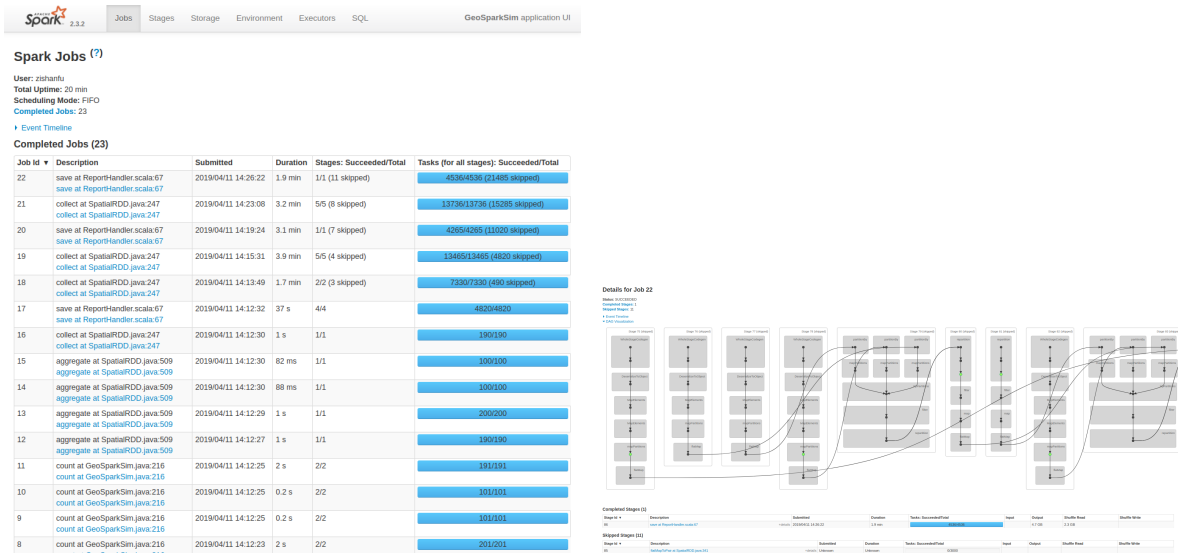
(b) Spark Cluster

Figure 7.3: Ganglia and Spark Master UI

7.2 Cluster Setting

All compared approaches are implemented with Apache Spark. We conduct the experiments on a cluster which has one master node and four worker nodes. Each machine has an Intel Xeon E5-2687WV4 CPU (12 cores, 3.0 GHz per core), 100 GB memory, and 4 TB HDD. We also install Apache Hadoop 2.6 and Apache Spark 2.3.2. We assign 10 GB memory to the Spark driver program that runs on the master machine, which is quite enough to handle any necessary global computation.

In the evaluation phase, several monitors are using to measure the experiments



(a) Spark Backend UI (b) Spark DAG

Figure 7.4: Spark Back-end Monitor

and cluster status, such as Spark master UI, Spark context web interface and Ganglia. Spark master UI can be accessed from 8080 port in master IP address in which listed nodes across the cluster, along with its number of CPUs(Cores) and memory. Spark context web interface can be accessed by opening $http://<driver-node>:4040$ in the web browser that displays scheduler stages and tasks, a summary of RDD sizes and memory usage, environmental and executors information for running application. Ganglia is a distributed monitoring system that can check the current CPU, memory, network utilization of the cluster. Figure 7.3a shows the running GeoSparkSim status in cluster. Four machines are deployed and left grids show the workload distribution among machines. Top four diagrams show metrics loads, memory, CPU, and network. GeoSparkSim uses it to evaluate workload distribution among machines. Figure 7.3b shows GeoSparkSim master UI and GeoSparkSim uses the monitoring to know launched workers and the setting for Spark programs, such as the number of CPUs and memory. Figure 7.4 shows Spark backend UI and GeoSparkSim uses it to

monitor application status, such as preprocessing and simulating.

7.3 Simulation

7.3.1 Experiment Setting

Parameters. We change the following parameters throughout the experiments (values listed in Table 7.2): (1) Number of vehicles: the number of vehicles that need to be simulated. (2) Time step: the time interval between two generated GPS locations. It is the simulation granularity. Time step has a significant influence on simulation time. For example, if the period is 10 minutes and 1 second per step, it requires 600 simulation iterations. If it is 0.8 second per step, 10 minutes need 750 steps. (3) The number of partitions: the number of partitions applied to RDDs (4) Repartition period: the period to recut RDDs. For example, the repartition period is 1 minute means that GeoSparkSim will run the temporal simulation and divide RDDs for every 1 minute. (5) Simulation period: the overall period that GeoSparkSim wants to simulate. By default, GeoSparkSim sets the temporary partition period to 2 minutes. In other words, it will invoke the vehicle partitioning layer to repartition the VehicleRDD after simulating every 2-minute traffic. For instance, assume a simulation workload (time step = 1 second, temporal partition size = 2 minute, simulation period = 8:00 to 8:15), GeoSparkSim will simulate the vehicle GPS locations from 8:00 to 8:15 at the granularity of 1 second. GeoSparkSim will repartition VehicleRDD Eight times (8:00, 8:02, 8:04, 8:06, 8:08, 8:10, 8:12, 8:14, 8:15). Besides, GeoSparkSim uses the KDB-tree partitioning method from GeoSpark Yu *et al.* (2018) in its spatial partitioning step. If the simulation period is less than the repartition period, GeoSparkSim will not invoke the repartition layer.

Evaluation metrics. We use execution time to measure the performance of each

Parameter	Range
Number of vehicles (thousand)	100, 200, 300
Time step (second)	1, 0.8, 0.6, 0.4, 0.2
Number of partitions	1000, 1500, 3000
Repartition period (minute)	1, 2, 4, 8, 10
Simulation period (minute)	10, 30, 60, 120

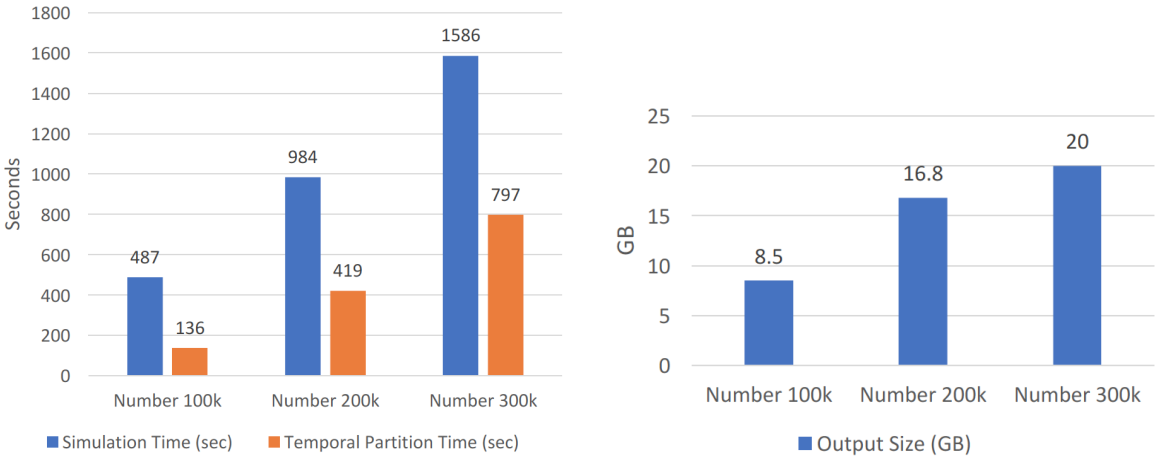
Table 7.2: Parameters

approach. Execution time is the time to finish a program. In the simulation, execution time is the total time to complete a simulation. In graph processing, execution time is the whole time to process the road network data in Apache Spark. In general, the execution time is the total time to generate vehicles and compute each vehicle's specific routes.

Tested data. We use the full road network of the Phoenix metropolitan area in the experiment. It consists of Maricopa and Pinal counties, comprising a total area of about 325 square miles. Rectangle 2 in Figure 7.1 and Table 7.1. The entire road network contains 250 thousand road junctions and 300 thousand road segments.

7.3.2 Number of Vehicle

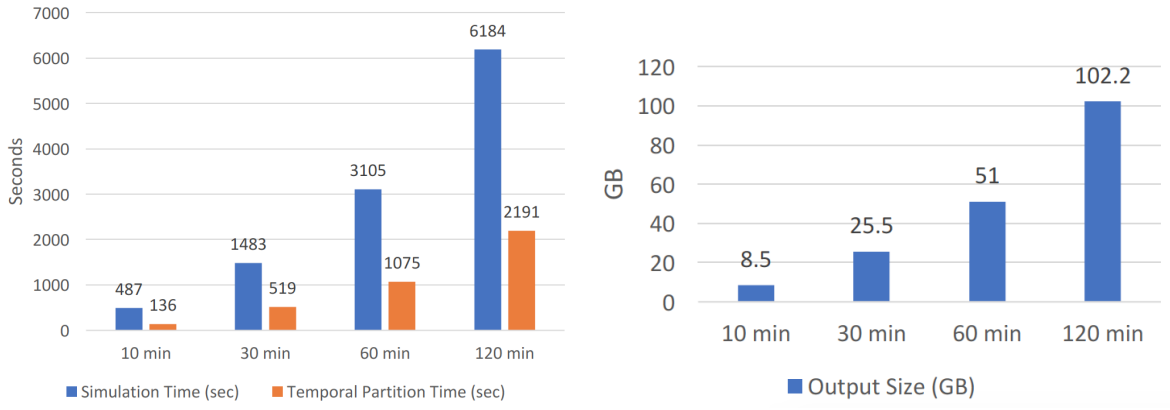
In this experiment, we study the impact of different numbers of vehicles and partitions combination. We vary the vehicle number from 100 thousand to 300 thousand and measure the execution time. 10 minute are simulated, the temporal partition size is 2 minute, the number of partition is 1500 and the time step is 1 second. Figure 7.5a shows that simulation and repartition time linear increases with more vehicles. Figure 7.5b It shows the simulation results size increases with more vehicle. These



(a) Execution time

(b) Data size

Figure 7.5: The Impact of the Number of Vehicles



(a) Execution time

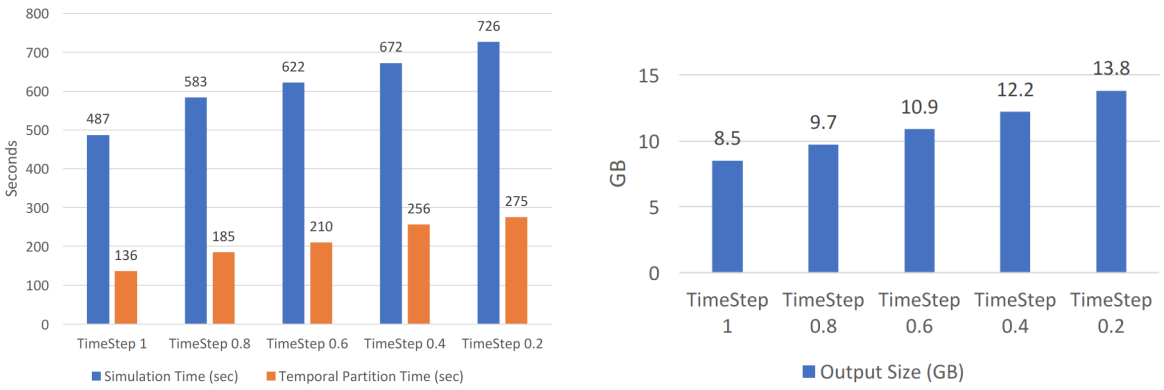
(b) Data size

Figure 7.6: The Impact of Simulation Period

make sense because more vehicles lead to more computation and more results.

7.3.3 Simulation Period

In this experiment, we further examine the impact of different simulation periods. We vary the simulation period from 10 minutes to 120 minutes. One hundred thousand vehicles are simulated, the temporal partition size is 2 minutes, the number of



(a) Execution time

(b) Data size

Figure 7.7: The Impact of Time Steps

partition is 1500 and the time step is 1 second. We report the results in Figure 7.6.

As shown in Figure 7.6a, as the simulation period increases, GeoSparkSim spends more time on simulating the traffic which makes sense because the system has to calculate the vehicle movements for more time steps. The vehicle partitioning time is also longer for the more extended simulation period. This happens because GeoSparkSim repartitions the VehicleRDD, 5, 10, and 15 times for different periods. It is worth noting that, the simulation period can be considerable because it will only increase the execution time linearly. GeoSparkSim will always partition the period to temporal partitions and simulate them one by one. Figure 7.6b shows more steps leading to a larger output size since it computes a more extended period.

7.3.4 Time Steps

In this experiment, we explore the impact of the different simulation time step. We vary the time step from 1 second to 0.2 seconds. One hundred thousand vehicles are simulated, the temporal partition size is 2 minutes, the number of partition is 1500 and the simulation period is 10 minutes. We report the results in Figure 7.7. The results show with the decreasing time step, the simulation execution time increases and larger

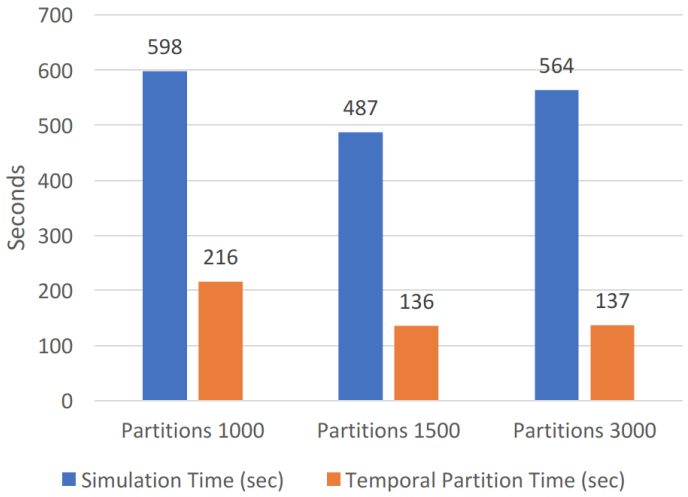


Figure 7.8: The Impact of Number of Partitions

output file. Time step means the period per updating steps. It is reasonable because the smaller time steps cause more steps and therefore more simulation computation. For example, 10 min simulation period and 1 second will have 600 times simulating step. However, 0.8-time step and the same simulation period will have 750 steps. Smaller time step leads to more computation and more results.

7.3.5 Number of Partitions

In this experiment, we seek the impact of a different number of partitions. We vary the numbers from 1000 to 3000. One hundred thousand vehicles are simulated, the time step is 1 second, the repartition period is 1 minute and the simulation period is 10 minutes. We report the results in Figure 7.8. In the figure, it displays a decreasing trend and then increasing with the rising number of partitions. It is supposed to have faster simulation time with more partitions since more partitions mean more simulation works run in parallel. However, more partitions in the same simulation will cause more overlapping situations and more data shuffle time. Because each partition will have a smaller size, the vehicle has a higher probability of going across

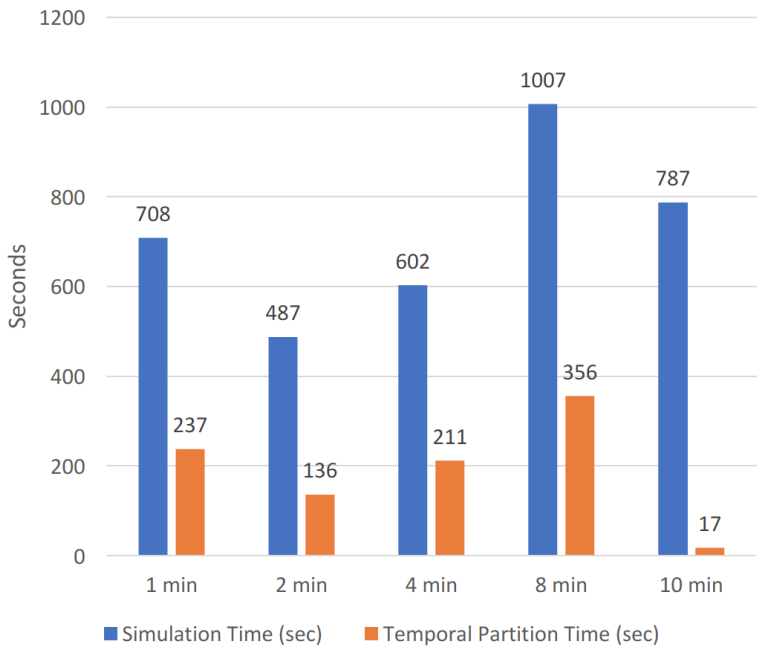


Figure 7.9: The Impact of Repartition Period

one partition to another. This also indicates the importance of simulation-aware repartition strategy.

7.3.6 Repartition Period

In this experiment, we analyze the impact of different repartition period. We vary the repartition period from 1 minute to 10 minutes, 100 thousand vehicles are simulated, the time step is 1 second, the number of partition is 1500 and the simulation period is 10 minutes. We report the results in Figure 7.9. In the figure, it shows a decreasing trend from 1 minute to 2 minutes, increases between 2 to 8 minutes and then decrease after that. Shorter length leads to less computation in temporal simulation and more possible partitions. However, this procedure will trigger new computation for all vehicles and signals and more repartition time. But longer repartition period may cause unbalanced workload, thus increasing the simulation time and repartition

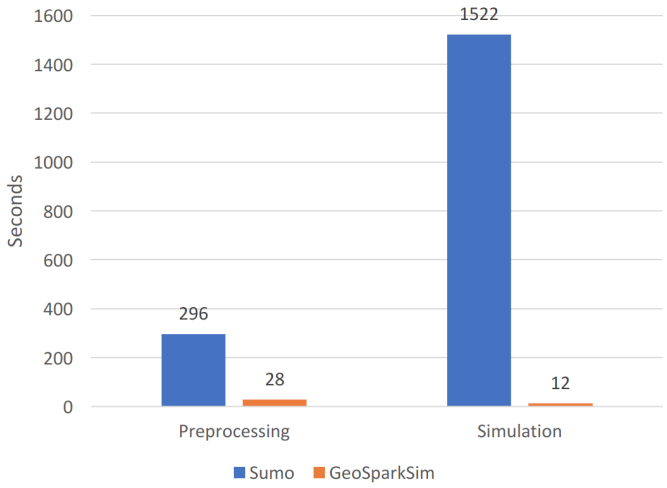
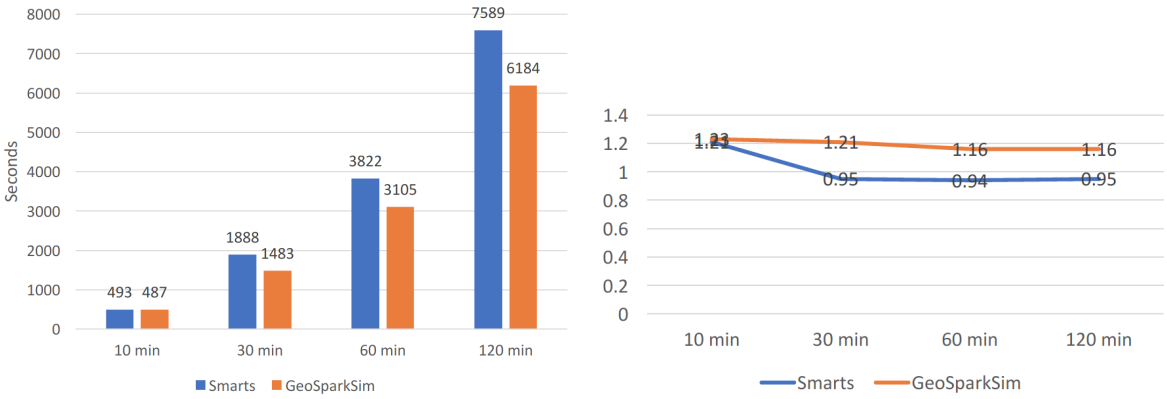


Figure 7.10: SUMO and GeoSparkSim

time as well. The decreasing trend from 8 minutes to 10 minutes is because 10 minutes just do a primary partition for data and the repartition time is very low. There is a trade-off in repartition period.

7.3.7 SUMO and GeoSparkSim

In this experiment, we compare the preprocessing time and simulation time between GeoSparkSim and SUMO in one thousand vehicles; the simulation period is 1 minute and 1-second time step. The simulation region is showing in rectangle 2 in Table 7.1 and Picture 7.1. SUMO and GeoSparkSim adopt same driving models, intelligent driving model, and MOBIL lane change model, but SUMO is not scalable. In Figure 7.10, GeoSparkSim completes road network preprocessing 10X faster and finish simulation more than 100X faster than SUMO. Traffic simulation is a very computing-intensive work, and GeoSparkSim makes a great contribution to distribute the simulation works among cluster and scale the simulation workload.



(a) Simulation Time

(b) Speed Up

Figure 7.11: Smarts and GeoSparkSim

7.3.8 SMARTS and GeoSparkSim

$$speedup = \frac{simulation\ period}{time\ cost} \tag{7.1}$$

In this experiment, we compare the simulation time and speed up factor between GeoSparkSim and SMARTS in 10 thousand vehicles, simulation period from 10 minutes to 120 minutes, 1-second time step with the same environment. Speed up is defined by the requested simulation period divided by simulation time cost. For example, if the user requests 20 minutes of simulation and simulator take 10 minutes to generate traffic, the speedup factor is 2, and it evaluates the traffic generation performance. SMARTS and GeoSparkSim use the same traffic models, intelligent driving model, and MOBIL lane change model. The simulation output for SMARTS is a collection of vehicle simulation GPS coordinate by step, while GeoSparkSim contains not only the GPS trajectories but also vehicle events in each step, such as acceleration and velocity. In Figure 7.11a, the simulation time is very close in 10 minutes, but GeoSparkSim has better performance with more extended simulation period. This

is because GeoSparkSim considers the dynamic vehicle spatial distribution over time and try to balance workload by periodically perform repartition. Figure 7.11b also indicate that GeoSparkSim can speed up simulation even when the request period is very long.

Chapter 8

CONCLUSION AND FUTURE WORK

In this paper, we presented GeoSparkSim, a scalable traffic simulator which extends Apache Spark to generate large-scale road network traffic data with microscopic traffic models. The proposed system seamlessly integrates with a Spark-based spatial data management system, GeoSpark, to deliver a holistic approach that allows data scientists to simulate, analyze and visualize large-scale traffic data. Moreover, GeoSparkSim equips VehicleRDD and parallelizes the simulation workload to a set of VehicleRDD transformations. The proposed system also employs a simulation-aware vehicle partitioning method to partition the workload among different machines. The experimental analysis shows that GeoSparkSim can simulate the movements of 200 thousand vehicles over a very large road network (250 thousand road junctions and 300 thousand road segments). GeoSparkSim source can be retrieved GeoSparkSim Source (2019) and a video demo can be viewed GeoSparkSim Demo (2019)

In the future, GeoSparkSim will include more driving models and perform more driving behaviors. Moreover, designing more APIs to connect the functions and libraries in Apache Spark and GeoSpark to do traffic analytic. In addition, GeoSparkSim currently unable to visualize large-scale data and may use graphics processing unit (GPU) to improve it in the future.

REFERENCES

Brinkhoff, T., "A framework for generating network-based moving objects", GeoInformatica **6**, 2, 153–180 (2002).

Düntgen, C., T. Behr and R. H. Güting, "Berlinmod: a benchmark for moving object databases", The VLDB Journal **18**, 6, 1335–1368, URL `https://doi.org/10.1007/s00778-009-0142-5` (2009).

Eldawy, A., L. Alarabi and M. F. Mokbel, "Spatial partitioning techniques in spatial-hadoop", Proceedings of the International Conference on Very Large Data Bases, PVLDB **8**, 12, 1602–1605 (2015).

Finkel, R. A. and J. L. Bentley, "Quad trees a data structure for retrieval on composite keys", Acta informatica **4**, 1, 1–9 (1974).

Gabriel, E., G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham and T. S. Woodall, "Open MPI: goals, concept, and design of a next generation MPI implementation", in "Recent Advances in Parallel Virtual Machine and Message Passing Interface, 11th European PVM/MPI Users' Group Meeting, Budapest, Hungary, September 19-22, 2004, Proceedings", pp. 97–104 (2004).

GasBuddy, "GasBuddy Website Link", Https://www.gasbuddy.com/ (2019).

Geisberger, R., P. Sanders, D. Schultes and D. Delling, "Contraction hierarchies: Faster and simpler hierarchical routing in road networks", in "International Workshop on Experimental and Efficient Algorithms", pp. 319–333 (Springer, 2008).

GeoFabrik, "GeoFabrik", `https://www.geofabrik.de/` (n.d.).

GeoSparkSim Demo, "GeoSparkSim video demo link", Http://www.public.asu.edu/ jiayu2/video/geosparksim-demo.mp4 (2019).

GeoSparkSim Source, "GeoSparkSim source github Website", Https://github.com/zishanfu/GeoSparkSim (2019).

GeoSystem, "World Geodetic System", `https://en.wikipedia.org/wiki/World_Geodetic_System` (n.d.).

Gipps, P. G., "A behavioural car-following model for computer simulation", Transportation Research Part B: Methodological **15**, 2, 105–111 (1981).

Greenshields, B., W. Channing, H. Miller *et al.*, "A study of traffic capacity", in "Highway research board proceedings", vol. 1935 (National Research Council (USA), Highway Research Board, 1935).

Guting, R. H., V. Almeida, D. Ansorge, T. Behr, Z. Ding, T. Hose, F. Hoffmann, M. Spiekermann and U. Telle, "Secondo: An extensible dbms platform for research prototyping and teaching", in "Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on", pp. 1115–1116 (IEEE, 2005).

Guttman, A., *R-trees: a dynamic index structure for spatial searching*, vol. 14 (ACM, 1984).

Hadoop, "Hadoop", `https://hadoop.apache.org/` (n.d.).

HDFS, "HDFS", `https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html` (n.d.).

Holzer, M., F. Schulz and D. Wagner, "Engineering multilevel overlay graphs for shortest-path queries", Journal of Experimental Algorithmics (JEA) **13**, 5 (2009).

JTS, "JTS", `https://github.com/locationtech/jts` (n.d.).

Karich, P. and S. Schröder, "Graphhopper", http://www. graphhopper. com, last accessed **4**, 2, 15 (2014).

Kesting, A., M. Treiber and D. Helbing, "General lane-changing model mobil for car-following models", Transportation Research Record **1999**, 1, 86–94 (2007a).

Kesting, A., M. Treiber and D. Helbing, "General lane-changing model mobil for car-following models", Transportation Research Record **1999**, 1, 86–94 (2007b).

Kesting, A., M. Treiber and D. Helbing, "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity", Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **368**, 1928, 4585–4605 (2010a).

Kesting, A., M. Treiber and D. Helbing, "Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity", Philosophical Transactions of the Royal Society of London A: Mathematical, Physical and Engineering Sciences **368**, 1928, 4585–4605 (2010b).

Klefstad, R., Y. Zhang, M. Lai, R. Jayakrishnan and R. Lavanya, "A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation", in "Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE", pp. 813–818 (IEEE, 2005).

Krajzewicz, D., G. Hertkorn, C. Rössel and P. Wagner, "Sumo (simulation of urban mobility)-an open-source traffic simulation", in "Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)", pp. 183–187 (2002).

Litman, T., *Autonomous vehicle implementation predictions* (Victoria Transport Policy Institute Victoria, Canada, 2017).

Lu, J. and R. H. Guting, "Parallel Secondo: Boosting Database Engines with Hadoop", in "International Conference on Parallel and Distributed Systems", pp. 738 –743 (2012).

Malewicz, G., M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser and G. Czajkowski, "Pregel: a system for large-scale graph processing", in "Proceedings of the 2010 ACM SIGMOD International Conference on Management of data", pp. 135–146 (ACM, 2010).

METIS Library, "METIS Library", http://glaros.dtc.umn.edu/gkhome/metis/metis/overview (n.d.).

MIT, "Senseable City", http://senseable.mit.edu/wave/, [Online; accessed 15-March-2019] (n.d.).

Mokbel, M. F., L. Alarabi, J. Bao, A. Eldawy, A. Magdy, M. Sarwat, E. Waytas and S. Yackel, "Mntg: an extensible web-based traffic generator", in "International Symposium on Spatial and Temporal Databases", pp. 38–55 (Springer, 2013).

Nagel, K. and M. Rickert, "Parallel implementation of the transims micro-simulation", Parallel Computing **27**, 12, 1611–1639 (2001).

NYCTraffic, "Yellow Taxi Trip Data", Https://data.cityofnewyork.us (2018).

OSM, "OpenStreetMap", http://www.openstreetmap.org/ (2019).

Osmosis, "Osmosis", https://github.com/openstreetmap/osmosis (n.d.).

OSRM, "Open Street Routing Machine", http://project-osrm.org/ (n.d.).

Overpass, "Overpass", http://overpass-api.de/ (n.d.).

Parquet, "Parquet", https://parquet.apache.org/ (n.d.).

Pramics Website, "Paramics Microsimulation", https://www.paramics.co.uk/en/ (2019).

Ramamohanarao, K., H. Xie, L. Kulik, S. Karunasekera, E. Tanin, R. Zhang and E. B. Khunayn, "Smarts: Scalable microscopic adaptive road traffic simulator", ACM Transactions on Intelligent Systems and Technology (TIST) **8**, 2, 26 (2017).

Robinson, J. T., "The kdb-tree: a search structure for large multidimensional dynamic indexes", in "Proceedings of the 1981 ACM SIGMOD international conference on Management of data", pp. 10–18 (ACM, 1981).

Vinoski, S., "Corba: integrating diverse applications within distributed heterogeneous environments", IEEE Communications magazine **35**, 2, 46–55 (1997).

Vissim, "PTV Vissim", Http://vision-traffic.ptvgroup.com/en-us/products/ptv-vissim/ (2019).

Waraich, R. A., D. Charypar, M. Balmer and K. W. Axhausen, "Performance improvements for large scale traffic simulation in matsim", in "9th STRC Swiss Transport Research Conference: Proceedings", vol. 565 (Swiss Transport Research Conference, 2009).

Wiki Gipps' model, "Gipps' model", https://en.wikipedia.org/wiki/Gipps\%27_model (2019).

Wiki TAO, "Wiki TAO", https://en.wikipedia.org/wiki/TAO_(software) (n.d).

Xin, R. S., J. E. Gonzalez, M. J. Franklin and I. Stoica, "Graphx: A resilient distributed graph system on spark", in "First International Workshop on Graph Data Management Experiences and Systems", p. 2 (ACM, 2013).

Yu, J., Z. Zhang and M. Sarwat, "Spatial data management in apache spark: The geospark perspective and beyond", Geoinformatica (2018).

Zaharia, M., M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker and I. Stoica, "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing", in "Proceedings of the USENIX Symposium on Networked Systems Design and Implementation, NSDI", pp. 15–28 (2012).

Zaharia, M., M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster computing with working sets.", HotCloud **10**, 10-10, 95 (2010a).

Zaharia, M., M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, "Spark: Cluster computing with working sets", in "USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010", (2010b).

Zhang, C.-T., R. Zhang and H.-Y. Ou, "The z curve database: a graphic representation of genome sequences", Bioinformatics **19**, 5, 593–599 (2003).

Zheng, Y., X. Xie and W.-Y. Ma, "Geolife: A collaborative social networking service among user, location and trajectory.", IEEE Data Engineering Bulletin **33**, 2, 32–39 (2010).

APPENDIX A

MACROSCOPIC SIMULATORS AND DISTRIBUTED MODELS

Macroscopic traffic simulator. Simulators in this category focus on general vehicular flow in the transportation road network. All vehicles drive similarly and merely move from the sources to the destinations step by step. Brinkhoff proposed a simulator Brinkhoff (2002) that generates moving objects for every single road segment in a simulation period. BerlinMOD Düntgen *et al.* (2009) is a popular moving object benchmark including a set of queries and a data generator which can generate road network traffic data for a number of identifiable vehicles. MNTG Mokbel *et al.* (2013) extends the functions, encapsulates Brinkhoff framework and BerlinMOD generators and provides a web service with a user-friendly and more accessible interface. Macroscopic simulators can quickly yield a massive amount of data because they are less computation-intensive. But the produced data may not be realistic and contain many vehicle collisions (e.g., vehicles have the same GPS locations).

Non-spatial partitioning approach. Some existing solutions partition the workload without taking into account the spatial proximity of the moving vehicles. Parallel-BerlinMOD Lu and Guting (2012) integrates BerlinMOD with a distributed DBMS called Parallel-Secondo Lu and Guting (2012) to deliver a scalable solution. It partitions the vehicles using generic partitioners such as hash partitioner and round-robin partitioner and parallelizes the computation to a set of Hadoop MapReduce operations Hadoop (nd). This approach is easy yet inappropriate for microscopic simulators because vehicles running on the same road segment are simulated by different machines. On the other hand, a microscopic simulator TRANSIMS Nagel and Rickert (2001) proposes to use graph cuts to partition the large road network then apply the same partitions to vehicles. It leverages MPI Gabriel *et al.* (2004) to coordinate different machines in a cluster. However, TRANSIMS may yield balanced network partitions such that each partition has a similar number of road nodes and segments but ignores an important fact: most road networks are idle and only major streets are full of vehicles.

Distributed computing models. All existing solutions are designed upon inefficient distributed models. Many of them still use message passaging services and do not employ advanced computation models and job schedulers. SMARTS Ramamohanarao *et al.* (2017) leverages simple TCP sockets, TRANSIMS Nagel and Rickert (2001) uses MPI Gabriel *et al.* (2004), and MATSim Waraich *et al.* (2009) only utilizes multi-thread synchronization. On the other hand, Parallel BerlinMod Lu and Guting (2012) uses Hadoop MapReduce Hadoop (nd). Although the Hadoop-based approach achieves high scalability, it still exhibits slow run time performance since it persists all intermediate data on disk. Apache Spark, on the other hand, provides a novel data abstraction called Resilient Distributed Datasets (RDDs) Zaharia *et al.* (2012) that are collections of objects partitioned across a cluster of machines. Each RDD is built using parallelized transformations (filter, join or groupBy) that could be traced back to recover the RDD data. In memory RDDs allow Spark to outperform existing models.