Building a Large-Scale Microscopic Road Network Traffic Simulator in Apache Spark

Zishan Fu
Computer Science department
Arizona State University
Tempe, Arizona
zishanfu@asu.edu

Jia Yu
Computer Science department
Arizona State University
Tempe, Arizona
jiayu2@asu.edu

Mohamed Sarwat
Computer Science department
Arizona State University
Tempe, Arizona
msarwat@asu.edu

***Abstract*—Road network traffic data has been widely studied by researchers and practitioners 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 GPS receivers and administrators must continuously monitor these devices. There has been a number of 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. In the paper, we propose 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 vehicles over a very large road network (250 thousand road junctions and 300 thousand road segments).**

***Index Terms*—Spatio-temporal Data, Apache Spark, Traffic Model, Microscopic Traffic Simulation**

I. Introduction

Road network traffic data contains the trajectories of a set of vehicles moving over a road network. Each trajectory consists of a number of GPS points which capture the vehicle locations at every audited time step. Such traffic data has been widely studied by researchers and practitioners in different disciplines that include urban planning, traffic prediction and spatial-temporal databases. For instance, researchers use traffic data to evaluate the impact of road network changes. Unfortunately, although there are millions of vehicles driving in big cities, collecting large-scale high-quality traffic data requires tremendous efforts since participating vehicles must install GPS receivers and administrators must continuously monitor these devices.

There are several classic traffic simulators proposed in the past two decades including Brinkhoff [2] and BerlinMod [3]. The caveat of using these approaches is that they do not consider microscopic traffic models [12], and hence cannot simulate individual vehicle driving behaviors and do not consider different road situations such as traffic signals. Microscopic traffic models are very useful in practice since they can generate data close to reality. However, a simulation involving so many characteristics is computation-intensive and traditional microscopic simulators such as SUMO [12] are only able to simulate limited vehicles over a small 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 into small chunks and assign them to different machines in a cluster. However, every time when a vehicle tries to change lane or accelerate, it has to check its distance to nearby vehicles. A proper partitioning method should take into account the spatial proximity of vehicles and minimize cross-partition data exchange.
- **Dynamic distribution.** The spatial distribution of moving vehicles changes over time. Nearby vehicles may soon become far from each other. Simulators have to employ proper mechanisms to handle this change.

To deal with the challenges, TRANSIMS [15] 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 [11] proposes to partition the geographical space to uniform grids. This does not work well if the vehicles and road network have a skewed distribution. SMARTS [16] comes up with an approach that partitions the space into small chunks numbered in a Z-curve like order. It then assigns nearby chunks to the same machine. However, it makes an unrealistic

assumption that the spatial distribution of moving vehicles is fixed.

In addition, most existing solutions are designed upon inefficient distributed models. For instance, Parallel BerlinMod [13] uses Hadoop MapReduce [7] and SMARTS [16] leverages simple TCP sockets. Apache Spark, on the other hand, provides a novel data abstraction called Resilient Distributed Datasets (RDDs) [24] 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.

This paper presents 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. 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).

Giving this outlook, this rest of this paper is presented as follows: Section II studies the related work. An overview of GeoSparkSim is given in Section III. Section IV details the internals of VehicleRDD and road network graph. Section V illustrates how to determine the routes of numerous vehicles. The vehicle partitioning method is explained in Section VI. Section VII describes the microscopic simulation models in GeoSparkSim. Section VIII shows the graphic user interface. A comprehensive experimental analysis is given in Section IX. Section X concludes the paper.

II. Related Works

Macroscopic traffic simulator. Simulators in this category focus on general vehicular flow in transportation road network. All vehicles drive in a similar manner and simply move from the sources to the destinations step by step. Brinkhoff proposed a simulator [2] that generates moving objects for every single road segment in a simulation period. BerlinMOD [3] 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 [14] 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 realistic and contain many vehicle collisions (e.g., vehicles have the same GPS locations).

Microscopic traffic simulator. Compare to macroscopic simulators, microscopic traffic simulators pay more attention to the detailed mobility of each individual vehicles and takes into account many different traffic events including lane changing, car following and traffic signals. SUMO [12] is one of the most popular 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 custom simulation data for different objects, such as vehicles, pedestrian, bicycles and railway. Such simulators are too computation-intensive because the driving behavior of a vehicle is affected by not only its own specification but also its nearby vehicles. 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 are able to generate realistic data, they suffer from the scalability issue.

It requires tremendous efforts to develop a scalable traffic simulator that fits a distributed environment because the simulator has to balance the workload to minimize data shuffle. Researchers have come up with many different approaches, explained below (see Table I).

Non-spatial partitioning approach. Some existing solutions partition the workload without taking into account the spatial proximity of the moving vehicles. Parallel-BerlinMOD [13] integrates BerlinMOD with a distributed DBMS called Parallel-Secondo [13] 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 [7]. 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 [15] proposes to use graph cuts to partition the large road network then apply the same partitions to vehicles. It leverages MPI [6] 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.

Spatial partitioning approach. Most scalable microscopic simulators use spatial partitioning methods to strike balanced workloads. MATSim [19] comes up with a method that splits the space to uniform grids (say, 5km*5km) then

TABLE I: Comparison among different traffic simulators

Feature	Brinkhoff [2]	SUMO [12]	BerlinMOD [3]	TRANSIMS [15]	MATSim [19]	ParamGrid [11]	SMARTS [16]	GeoSparkSim
Simulation model	Macroscopic	Microscopic	Macroscopic	Microscopic	Microscopic	Microscopic	Microscopic	Microscopic
Scalability	Single node	Single node	Distributed [13]	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 [7]	MPI [6]	Thread sync.	CORBA [18]	TCP sockets	RDD [24]

uses these grids to partition road networks and vehicles. It uses multi-threads to parallelize the computation. ParamGrid [11] uses a partitioning method similar to MATSim but utilizes CORBA [18] framework which internally uses RPC. SMARTS [16] partitions the space to very small cells and orders them into a curve similar 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 them because (1) their partitioners cannot well-balance vehicles due to their skewed spatial distribution. GeoSpark [23] and SpatialHadoop [4] 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 automatically repartition vehicles over time.

Distributed computing models. Most 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 [16] leverages simple TCP sockets, TRANSIMS [15] uses MPI [6], and MATSim [19] only utilizes multi-thread synchronization. On the other hand, Parallel BerlinMod [13] uses Hadoop MapReduce [7]. Although the Hadoop-based approach achieves high scalability, it still exhibits slow run time performance since it persists all the intermediate data on disk. Apache Spark provides a novel data abstraction called Resilient Distributed Datasets (RDDs) [24] 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.

III. System Overview

GeoSparkSim consists of a Graphic User Interface (GUI) and four layers: (1) Vehicle RDD and road network layer (2) route planning layer (3) simulation-aware route partitioning layer (4) microscopic traffic generating layer. GeoSparkSim works in concert with GeoSpark Spatial RDDs and Spark GraphX to deliver a holistic approach that allows data scientists to simulate, analyze and visualize large-scale urban traffic data.

Graphic user interface (GUI). The GUI of GeoSparkSim is a front-end map interface that users mainly interact with. It provides two functionalities: (1) it takes input parameters from users including the number of to-be-simulated vehicles, simulation region, vehicle setting, time step, simulation period and so on. A user can simply draw a rectangular window on the map and fill in necessary parameters. Then GeoSparkSim backend will download the road network of the specified region and start the simulation. (2) Once the simulation is completed, the interface can animate the generated vehicle movements on the selected road network upon the user's request.

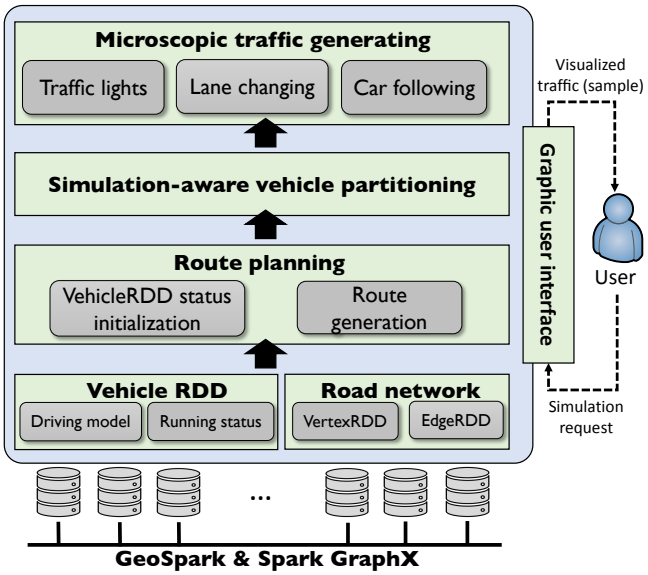


Fig. 1: GeoSparkSim architecture

VehicleRDD and road network. VehicleRDD is a specialized Spark RDD which consists of millions of individual vehicle records. Each vehicle has several attributes such as acceleration/deceleration, velocity, safe distance and so on. The values of these attributes are randomized such that 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 own status to record its current simulated speed, GPS locations and acceleration state. Road network describes road situation of the specified simulation region and is stored as a Spark GraphX graph which consists of a VertexRDD and an EdgeRDD. VertexRDD contains all road junctions and EdgeRDD contains all road segments.

Route planning. GeoSparkSim first creates the initial status vehicles in this layer. Then it randomly generates sources and destinations for every vehicle following the population 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 that GeoSparkSim can quickly compute routes for every source and destination pair on top of it.

Simulation-aware vehicle partitioning. After route planning, every vehicle in VehicleRDD has a planned route. These vehicles will exactly follow the planned routes but each of them may show different microscopic behaviors such as accelerating and lane changing. 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 a VehicleRDD, GeoSparkSim repartitions 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 period of time.

Microscopic traffic generating. 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 microscopic traffic which consists of vehicle status at each time step. Each vehicle has a safe distance buffer to avoid collisions. A vehicle will adjust its speed if its next movement will invade the safe distance buffer of its nearby vehicles. Traffic signals at road intersections also affect the traffic.

IV. VehicleRDD and road network

GeoSparkSim possesses two data structures, VehicleRDD and road network. This section presents their internals and explains how to create them from scratch. Later on, GeoSparkSim will simulate vehicle movements by performing a set of VehicleRDD transformations.

A. VehicleRDD

GeoSparkSim VehicleRDDs are in-memory distributed datasets that extend traditional RDD to accommodate vehicle objects in Apache Spark. Each VehicleRDD consists of many partitions and each partition contains thousands of vehicles. This way, each vehicle in this RDD can have its own randomized driving model and status such that the system yields arbitrary trajectories. In other words, a VehicleRDD is a snapshot of current vehicle movements over the road network.

Vehicle object. Each vehicle in VehicleRDD has two components (1) driving model: it incorporates several parameters to control the driving behavior of a vehicle. They are acceleration(m/s^2), deceleration(-m/s^2), steady speed (km/h), lane changing probability (%), safe distance (meter), and planned route. The route is generated by the route planning layer. 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 different lanes and generate different acceleration/deceleration events. (2) running status: it includes several attributes to indicate the current status of a running vehicle. They are driving event (acceleration/deceleration/steady), speed, and GPS location.

VehicleRDD transformation. To simulate the traffic of numerous vehicles in a specific time period, GeoSparkSim generates GPS locations of these vehicles for every simulation time step. For instance, if the time period is 1 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 individual 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 Section VII.

B. Road Network

Road network in GeoSparkSim is a Spark GraphX graph which consists of a VertexRDD and an EdgeRDD. GeoSparkSim supports the widely used OpenStreetMap XML format so that users can easily load different road networks into GeoSparkSim.

VertexRDD. VertexRDD contains millions of vertexes and each vertex is a road junction that connects two road segments. 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.

EdgeRDD. EdgeRDD accommodates millions of edges and each edge is a road segment which is a straight way between two vertexes. Each edge consists of two attributes (1) the IDs of source and destination vertexes (2) type: an edge can be a part of the highway or residential street. It may also have a one-way restriction.

C. Data importing

GeoSparkSim creates random driving models for all vehicles in VehicleRDD and also allows the user to provide custom rules via a configuration file, such as higher steady speed and shorter safe distance. GeoSparkSim can load OSM XML format data to build the road network. Two parts of OSM XML data format are needed by GeoSparkSim, nodes and ways. Nodes define all the junctions and each of them has id, coordinate and type. Ways define shapes and type of streets. The shape of a street is a sequence of junction IDs which details individual road segments. GeoSparkSim decomposes streets to road segments and stores them in EdgeRDD.

V. Route planning

Before starting to simulate the traffic, GeoSparkSim's route planning layer will create the initial status of a VehicleRDD and generate individual routes for each vehicle in this RDD.

A. VehicleRDD status initialization

GeoSparkSim's route planning layer first initializes the status of vehicles in this RDD. It 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.

There are some existing approaches to generate the sources and destinations of moving objects such as the data-space oriented approach (DSO) and network-based approach (NB). The data-space oriented approach generates source and destination points based on a certain spatial distribution and runs map matching to match points to their nearest nodes in the road network. This spatial distribution can be the density of human population or the density of buildings. Regions with high density will produce more sources and destinations. The network-based approach randomly selects road junctions as sources and destinations. GeoSparkSim uses DSO approach to make the data more realistic.

B. Route generation

Now, GeoSparkSim will generate the shortest path for every pair of source and destination. There are many algorithms to compute routes based on networks, such as Dijkstra, A-star, D-star, and Multi-Level Dijkstra, etc.

For the sake of routing speed, GeoSparkSim leverages GraphHopper [8], an open source route planning library, to generate routes for every source and destination pair. 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.

VI. Simulation-aware vehicle partitioning

To achieve load balance in a Spark cluster, GeoSparkSim needs to split the VehicleRDD to approximately equal-size partitions. The default data partitioning method in Spark such as hash partitioner exhibits good performance for regular ETL queries but cannot handle microscopic traffic simulation because it does not take into account the spatial proximity of simulated trajectories. Spatial partitioning approaches, e.g., Quad-Tree and R-Tree, in GeoSpark [22], SpatialHadoop [5] , BinJoin [20], ST-Hadoop [1], work well for points, polygons and short trajectories but cannot handle long and tangled trajectories. DST [21] and DITA [17] propose partitioning methods for trajectories but cannot tackle the temporal attribute in microscopic traffic simulation.

The vehicle partitioning layer in GeoSparkSim partitions the workload temporally and spatially. It takes as input a VehicleRDD and spatially partitions the vehicles according to their planned routes in the upcoming temporal partition. GeoSparkSim periodically invokes this layer to repartition the VehicleRDD to make sure that the RDD always carries out balanced partitions. To be precise, the simulation-aware vehicle partitioning layer has the following advantages: (1) partition by short-term routes instead of planned long routes to avoid cross-partition routes as many as possible (2) allow local microscopic traffic simulation inside each partition (3) support dynamic vehicle movement distribution.

Step 1: Temporal partitioning. Throughout the simulation, all vehicles will follow specific routes planned by the route planning layer. However, these routes generally span many blocks and tangle with others (depicted in Figure 2). It is very hard to do spatial partitioning on the VehicleRDD according to their overall routes. To remedy that, this step first partitions the simulation period into several equal-width temporal partitions. Then it will simulate these partitions one by one. This way, GeoSparkSim can easily do spatial partitioning over the temporal partition routes of these vehicles which are much shorter.

Step 2: Estimate routes in the temporal partition. Before simulating the trajectories in each temporal partition, GeoSparkSim first needs to estimate the routes in this partition. Given the overall planned route of a vehicle and its ending spatial location in the last temporal partition, this step calculates its farthest route using its steady speed. During the simulation of this temporal partition, although each vehicle always follows the estimated route, it does not necessarily finish the planned route because it may run into random delays caused by red signals and traffic jams.

Step 3: Spatial partitioning. This step utilizes the default spatial partitioning methods in GeoSpark, Quad-Tree, to partition the VehicleRDD. It includes the following substeps: (1) create sample: draw a random sample over the VehicleRDD to represent the spatial data distribution of its temporal partition routes (2) calculate boundaries: create a Quad-Tree structure on the sample's temporal partition routes and use the boundaries of leaf nodes as geometrical boundaries of new RDD partitions (3) repartition VehicleRDD: vehicles whose estimated routes fall into the same boundary are sent to the same partition. Vehicles whose estimated routes intersect several partition boundaries are duplicated to all intersected partitions. Note that, the geometrical boundaries in this step can produce roughly balanced partitions because this step builds balanced tree structures on a real sample of estimated routes.

Step 4: Local microscopic traffic simulation. This step performs the local microscopic traffic simulation on each VehicleRDD partition. Since all vehicles are partitioned according to the spatial proximity of their estimated temporal partition routes, this step does not have to communicate with other partitions for nearby vehicle statuses via data shuffle. This step will be detailed in the next section.

After simulating each temporal partition, GeoSparkSim will invoke Step 2-4 in this layer to repartition the VehicleRDD for the upcoming temporal partition. This is to maintain the workload balance of this RDD because these vehicles keep moving in the simulated region and their spatial distribution varies in different temporal partitions.

Figure 2 is an example of GeoSparkSim workflow. The user may ask GeoSparkSim to simulate the traffic in Tempe, Arizona from 8:00 am to 9:00 am. GeoSparkSim will first plan the routes for the VehicleRDD and do temporal partitioning to partition this 1-hour period into 4 x 15-minute temporal partitions. Then, for temporal partition 1 (8:00 am to 8:15 am), GeoSparkSim runs Step 2-4 to partition VehicleRDD (see the middle part in Figure 2) and then performs local simulation in each RDD partition. GeoSparkSim will execute the same

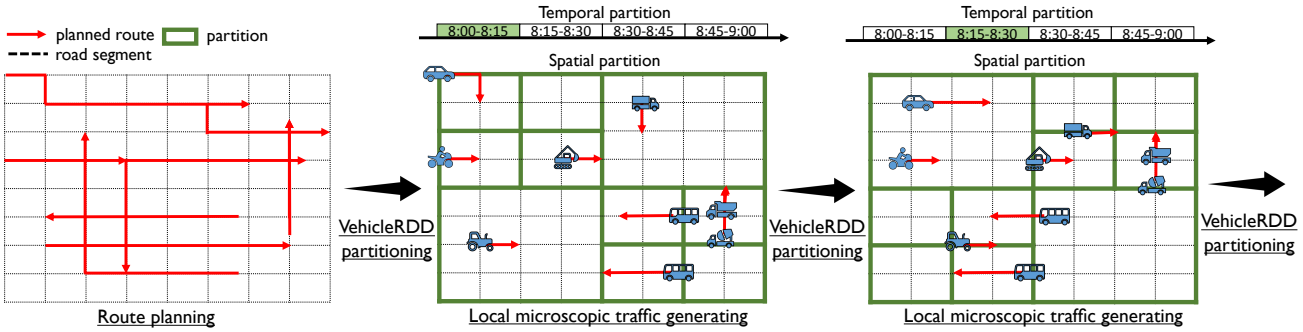


Fig. 2: GeoSparkSim workflow

steps for the rest of the temporal partitions one by one.

VII. Microscopic traffic simulation

A. Simulation Algorithm

After partitioning the VehicleRDD, GeoSparkSim is now ready to run a local microscopic simulation on each RDD partition. All vehicles will follow their planned temporal partition routes in this temporal partition. Every route starts from the last location of the vehicle. GeoSparkSim equips a generic microscopic simulation algorithm (see Algorithm 1) which can plug in many common traffic simulation models. This algorithm first calculates the number of GPS locations 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 be run by GeoSparkSim. The algorithm then runs a set of iterations and in each iteration, it first asks every vehicle to check whether its nearby vehicles invade its safe distance buffer. A safe distance buffer is a small rectangle centered at the vehicle itself. It describes the minimum safe distance between two vehicles to avoid collisions. Then the algorithm will generate individual behavior for every vehicle based on the distance check result. After the local simulation on each RDD partition, GeoSparkSim will persist the simulation results on HDFS.

B. Microscopic simulation models

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

1) Car-following: The car-following model uses the Intelligent-Driver Model (IDM) [10] to update the current vehicle's speed based on its distance to the vehicle ahead of it. IDM decides the acceleration and deceleration for every time step in the simulation. The parameters of IDM include the steady speed of this vehicle, acceleration factor, and deceleration factor. If nearby vehicles are within the safe distance buffer, this model will make the vehicle decelerate.

Algorithm 1: GeoSparkSim microscopic simulation

```
Data: iterations
Result: VehicleRDD and road network
Update the planned temporal partition routes of VehicleRDD
  and perform repartition;
foreach partition in VehicelRDD do
    foreach iteration do
        foreach vehicle V do
            Compute the safe distance buffer;
            Create an empty list L;
            foreach vehicle NearbyV do
                // Assume NearbyV will move
                   forward as usual
                if NearbyV's next movement will be in
                  V's buffer then
                    Add NearbyV to L;
            Generate the next movement of V;
return Simulation results
```

If there are no nearby vehicles, the model may accelerate the vehicle.

2) Lane-changing: GeoSparkSim uses a general lane-changing model called MOBIL [9]. A vehicle changes the lane based on a specific probability. This model avoids the collisions by using the safe distance buffer. If there are no other vehicles inside the safe distance buffer, the vehicle might change its lane. Otherwise, it will stay in the same lane.

3) Traffic lights: The greed-red intervals of traffic lights are generated by GeoSparkSim randomly. When a traffic light appears in the safe buffer 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 light is red. If the light becomes green in the next time steps, the vehicle will start to accelerate.

VIII. Graphical User Interface (GUI)

GeoSparkSim provides a graphical user interface that allows users to interact with the system. The user can issue simulation requests and see visualized simulation results via this interface.

Interface components. GeoSparkSim user interface contains three parts: input panel, map panel and report panel. The input panel on the top is the place where the user can describe his / her request by checking several options. The map panel

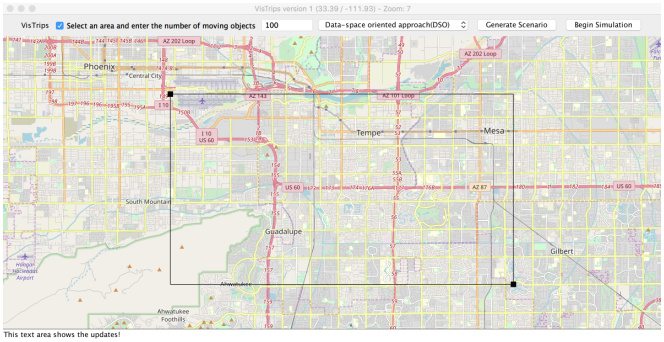
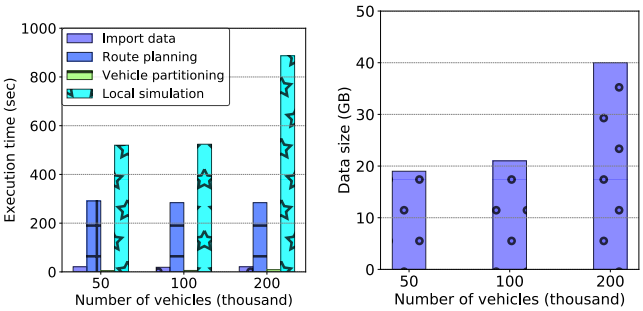


Fig. 3: GeoSparkSim GUI



(a) Execution time (b) Data size

Fig. 4: Performance on different numbers of vehicles

TABLE II: Parameters (default values are underlined)

Parameter	Range
Number of vehicles (thousand)	50, 100, 200
Time step (second)	0.2, 0.4, 0.6, 1
Simulation period (minute)	10, 20, 30

on the center shows the viewport of a road network with a map background. Users can zoom-in/out and pan on this panel to see different regions. The report panel on the bottom shows the completion time of each processing step to keep track of the simulating process.

Issue a simulation request. The user can begin to enter the number of moving objects, select VehicleRDD initialization approach and the simulation time period on the input panel. Then he or she will need to draw a rectangle on the map panel to specify the simulation region. Moreover, this interface allows the user to control vehicle behaviors by changing a vehicle configuration file such as higher steady speed, shorter safe distance or more lane-changing attempts.

Visualize the simulation result. Once the simulation is done, the user can opt to ask for visualized simulation results. The GUI will create an overlay painter over the map panel and render the simulated vehicle locations to points at every simulation time step. In each time step, the painter will update the points in the overlay object list to show the latest simulation locations. It is worth noting that GUI can only visualize a small number of vehicle trajectories because it runs only on the master machine. Therefore, after receiving the visualization request, GeoSparkSim will take a random sample from the vehicle trajectories to reduce the visualization overhead.

IX. Experiment

A. Experiment setting

Parameters. We change the following parameters throughout the experiments (values listed in Table II): (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. (3) simulation period: the overall period that GeoSparkSim wants to simulate. By default, GeoSparkSim sets the temporal partition size to 5 minutes. In other words, it will invoke the vehicle partitioning layer to repartition the VehicleRDD after simulating every 5-minute traffic. For instance, assume a simulation workload (time step = 1 second, temporal partition size = 5 minutes, 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 three times (8:00, 8:05, 8:10). In addition, GeoSparkSim uses the Quad-Tree partitioning method from GeoSpark [23] in its spatial partitioning step.

Evaluation metrics. We use the following metrics to measure the performance of each approach: (1) Execution time: it is the time of running a GeoSpark simulation workload. (2) Data size: it is the size of generated traffic data.

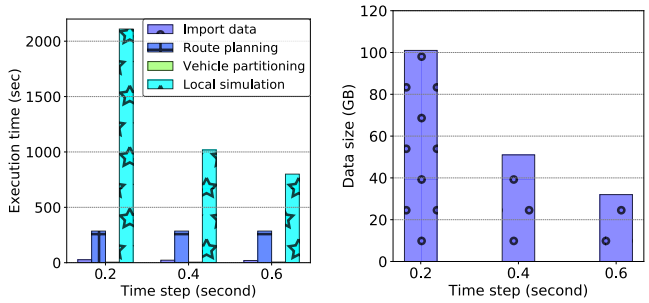
Tested data. We use the full road network of the Phoenix metropolitan area in the experiment. It consists of Maricopa and Pinal counties, comprising a total area of about 14600 square miles. The entire road network contains 250 thousand road junctions and 300 thousand road segments.

Cluster settings. 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.

B. The impact of the number of vehicles

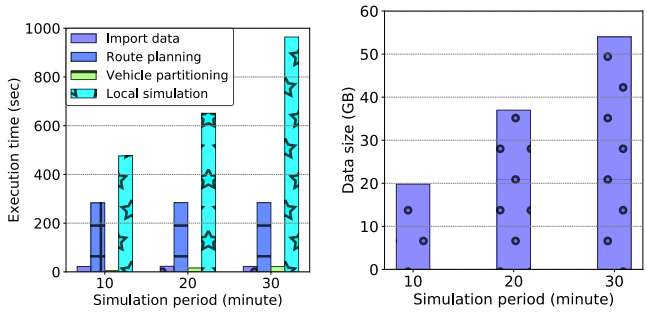
In this experiment, we study the impact of different numbers of vehicles. We vary the number from 50 thousand to 200 thousand and measure the execution time and data size. The results are reported in Figure 4. We also show the time taken by each layer of GeoSparkSim. During the simulation, GeoSparkSim partitions the VehicleRDD twice (temporal partition size is 5 minutes).

As shown in Figure 4a, both data importing and route planning part take almost constant time. This happens because we use the same big road network for all experiments. After



(a) Execution time (b) Data size

Fig. 5: Performance on different time steps



(a) Execution time (b) Data size

Fig. 6: Performance on different simulation period

loading the network, GeoSparkSim leverages GraphHopper to build an index on it to accelerate the route planning. The index construction is very time-consuming as opposed to the route lookup part. On the other hand, both vehicle partitioning layer and local microscopic simulation cost more time on the larger number of vehicles. This makes sense because GeoSparkSim needs to spend more time on shuffling data across the machines and simulating traffic on each partition if there are more vehicles in the VehicleRDD. In addition, the local microscopic simulation on each VehicleRDD partition takes most of the simulation time. This is because the local simulation costs lots of time to check the safe distance buffer among different vehicles.

As depicted in Figure 4b, as we increase vehicles in the VehicleRDD, the traffic data generated by GeoSparkSim also increases. This makes sense because GeoSparkSim will have to provide more GPS locations at each simulation time step if there are more vehicles.

C. The impact of time steps

In this experiment, we test the performance of GeoSparkSim on different simulation time steps, 0.2 second, 0.4 second and 0.6 second. The simulation period is 10 minutes and the temporal partition size is 5 minutes. 100 thousand vehicles are simulated in the experiment. We report the execution time and data size of GeoSparkSim simulation in Figure 5.

As described in Figure 5a, with larger time steps, GeoSparkSim will spend less time on local microscopic simulation on each RDD partition. This happens because, given a fixed simulation period, the system will generate fewer GPS locations if the time step is larger. And, in each partition, GeoSparkSim takes fewer iterations to simulate the vehicle movements. On the other hand, data importing, route planning and vehicle partitioning part have constant execution time because they are only affected by the number of simulated vehicles.

Figure 5b shows the size of generated data for different time steps. Simulation with 0.2 time step obtains the most data because it is more granular. Simulation with 0.6 time step has the least data because it is less granular.

D. The impact of simulation periods

In this experiment, we further examine the impact of different simulation periods. We vary the simulation period from 10 minutes to 30 minutes. 100 thousand vehicles are simulated, the temporal partition size is 5 minutes and the time step is 1 second. We report the results in Figure 6.

As shown in Figure 6a, as the simulation period increases, GeoSparkSim spends more time on simulating the traffic. This makes sense because the system has to calculate the vehicle movements for more time steps. The vehicle partitioning time is also larger for a larger simulation period. This happens because GeoSparkSim repartitions the VehicleRDD, 2, 4, and 6 times for different periods. On the other hand, data importing and route planning have constant execution time because they are only affected by the number of simulated vehicles.

As depicted in Figure 6b, with a larger simulation period, GeoSparkSim will generate more traffic data. This matches the expectation because the system will produce GPS locations for more time steps.

It is worth noting that, the simulation period can be very large because it will only increase the execution time linearly. GeoSparkSim will always partition the period to temporal partitions and run a simulation for them one by one.

X. Conclusion

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).

XI. Acknowledgment

This work is supported in part by the National Science Foundation (NSF) under Grant 1845789, the Salt River Project Agricultural Improvement and Power District (SRP), and the DOD-ARMY Training and Doctrine Command (TRADOC).

References

[1] Louai Alarabi, Mohamed F. Mokbel, and Mashaal Musleh. St-hadoop: A mapreduce framework for spatio-temporal data. In *Proceedings of the International Conference on Advances in Geographic Information Systems, SIGSPATIAL*, pages 84–104, 2017.

[2] Thomas Brinkhoff. A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180, 2002.

[3] Christian Düntgen, Thomas Behr, and Ralf Hartmut Güting. Berlinmod: a benchmark for moving object databases. *The VLDB Journal*, 18(6):1335–1368, 2009.

[4] Ahmed Eldawy, Louai Alarabi, and Mohamed F Mokbel. Spatial partitioning techniques in spatialhadoop. *Proceedings of the International Conference on Very Large Data Bases, PVLDB*, 8(12):1602–1605, 2015.

[5] Ahmed Eldawy and Mohamed F. Mokbel. Spatialhadoop: A mapreduce framework for spatial data. In *Proceedings of the International Conference on Data Engineering, ICDE*, pages 1352–1363, 2015.

[6] Edgar Gabriel, Graham E. Fagg, George Bosilca, Thara Angskun, Jack J. Dongarra, Jeffrey M. Squyres, Vishal Sahay, Prabhanjan Kambadur, Brian Barrett, Andrew Lumsdaine, Ralph H. Castain, David J. Daniel, Richard L. Graham, and Timothy 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*, pages 97–104, 2004.

[7] Apache Hadoop. http://hadoop.apache.org/.

[8] Peter Karich and S Schröder. Graphhopper. *http://www. graphhopper. com, last accessed*, 4(2):15, 2014.

[9] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[10] Arne Kesting, Martin Treiber, and Dirk 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, 2010.

[11] Raymond Klefstad, Yue Zhang, Mingjie Lai, R Jayakrishnan, and Riju Lavanya. A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, pages 813–818. IEEE, 2005.

[12] Daniel Krajzewicz, Georg Hertkorn, Christian Rössel, and Peter Wagner. Sumo (simulation of urban mobility)-an open-source traffic simulation. In *Proceedings of the 4th middle East Symposium on Simulation and Modelling (MESM20002)*, pages 183–187, 2002.

[13] Jiamin Lu and Ralf Hartmut Guting. Parallel Secondo: Boosting Database Engines with Hadoop. In *International Conference on Parallel and Distributed Systems*, pages 738 –743, 2012.

[14] Mohamed F Mokbel, Louai Alarabi, Jie Bao, Ahmed Eldawy, Amr Magdy, Mohamed Sarwat, Ethan Waytas, and Steven Yackel. Mntg: an extensible web-based traffic generator. In *International Symposium on Spatial and Temporal Databases*, pages 38–55. Springer, 2013.

[15] Kai Nagel and Marcus Rickert. Parallel implementation of the transims micro-simulation. *Parallel Computing*, 27(12):1611–1639, 2001.

[16] Kotagiri Ramamohanarao, Hairuo Xie, Lars Kulik, Shanika Karunasekera, Egemen Tanin, Rui Zhang, and Eman Bin Khunayn. Smarts: Scalable microscopic adaptive road traffic simulator. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):26, 2017.

[17] Zeyuan Shang, Guoliang Li, and Zhifeng Bao. DITA: distributed in-memory trajectory analytics. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, pages 725–740, 2018.

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

[19] Rashid A Waraich, David Charypar, Michael Balmer, and Kay W Axhausen. Performance improvements for large scale traffic simulation in matsim. In *9th STRC Swiss Transport Research Conference: Proceedings*, volume 565. Swiss Transport Research Conference, 2009.

[20] Randall T. Whitman, Michael B. Park, Bryan G. Marsh, and Erik G. Hoel. Spatio-temporal join on apache spark. In *Proceedings of the International Conference on Advances in Geographic Information Systems, SIGSPATIAL*, pages 20:1–20:10, 2017.

[21] Dong Xie, Feifei Li, Bin Yao, Gefei Li, Liang Zhou, and Minyi Guo. Simba: Efficient In-Memory Spatial Analytics. In *Proceedings of the ACM International Conference on Management of Data, SIGMOD*, 2016.

[22] Jia Yu, Jinxuan Wu, and Mohamed Sarwat. GeoSpark: A Cluster Computing Framework for Processing Large-Scale Spatial Data. In *Proceedings of the ACM Symposium on Advances in Geographic Information Systems, ACM GIS*, 2015.

[23] Jia Yu, Zongsi Zhang, and Mohamed Sarwat. Spatial data management in apache spark: The geospark perspective and beyond. *Geoinformatica*, 2018.

[24] Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauly, Michael J. Franklin, Scott Shenker, and Ion 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*, pages 15–28, 2012.