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

Zishan Fu
Arizona State University
Tempe, Arizona
ZishanFu@asu.edu

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

Mohamed Sarwat
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. The existing urban traffic simulators suffer from two critical issues (1) scalability: most of them only offer single-machine solutions which are 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, and 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. A full-fledged prototype of GeoSparkSim is implemented in Apache Spark. In this demonstration, we will show the attendees how to issue GeoSparkSim simulation tasks via the user interface, visualize simulated vehicle movements, and monitor the backend Spark cluster status.

CCS CONCEPTS

• **Information systems** → **MapReduce-based systems**; **Geographic information systems**; • **Computing methodologies** → **MapReduce algorithms**.

KEYWORDS

Traffic simulation, distributed computation, road network

ACM Reference Format:
Zishan Fu, Jia Yu, and Mohamed Sarwat. 2019. Demonstrating GeoSparkSim: A Scalable Microscopic Road Network Traffic Simulator Based on Apache Spark. In *16th International Symposium on Spatial and Temporal Databases (SSTD '19), August 19–21, 2019, Vienna, Austria.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340964.3340984

SSTD '19, August 19–21, 2019, Vienna, Austria

ACM ISBN 978-1-4503-6280-1/19/08…$15.00
https://doi.org/10.1145/3340964.3340984

1 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 [1] and BerlinMod [2]. The caveat of using these approaches is that they do not consider microscopic traffic models [8], and hence cannot simulate individual vehicle driving behaviors and tackle 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 [8] 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 to 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 [10] 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 [7] proposes to partition the geographical space to uniform grids. This does not work well if the vehicles and road network have skewed distribution. SMARTS [11] comes up with an approach that partitions the space

Table 1: Comparison among different traffic simulators

Feature	Brinkhoff [1]	SUMO [8]	BerlinMOD [2]	TRANSIMS [10]	MATSim [13]	ParamGrid [7]	SMARTS [11]	GeoSparkSim
Simulation model	Macroscopic	Microscopic	Macroscopic	Microscopic	Microscopic	Microscopic	Microscopic	Microscopic
Scalability	Single node	Single node	Distributed [9]	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 [4]	MPI	Thread sync.	CORBA [12]	TCP sockets	RDD [14]

into small chunks numbered in a Z-curve like order. It then assigns nearby chunks to the same machine. But 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 [9] uses Hadoop MapReduce [4] and SMARTS [11] leverages simple TCP sockets. Apache Spark, on the other hand, provides a novel data abstraction called Resilient Distributed Datasets (RDDs) [14] 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 demonstrates GeoSparkSim [3], 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 contributions are as follows:

• 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. In the demonstration section, we will show the attendees how to issue GeoSparkSim simulation tasks via the user interface, visualize simulated vehicle movements, and monitor the backend Spark cluster status.

2 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

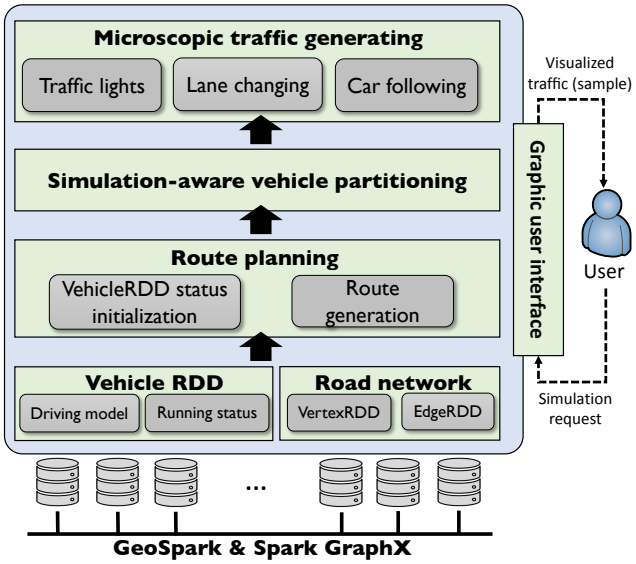


Figure 1: GeoSparkSim architecture

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 generate simulated traffic.

VehicleRDD and road network. VehicleRDD is a specialized Spark RDD which consists of millions of individual vehicle objects. Each vehicle has several attributes such as acceleration/deceleration, velocity, safe distance and so on. The values of these attributes are randomized so each vehicle has its personalized behavior. The user can also provide customized vehicle behavior model by extending *movement control* class (explained later). Besides that, each vehicle 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 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 on 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 GeoSparkSim can quickly compute routes for every source and destination pair on top of it.

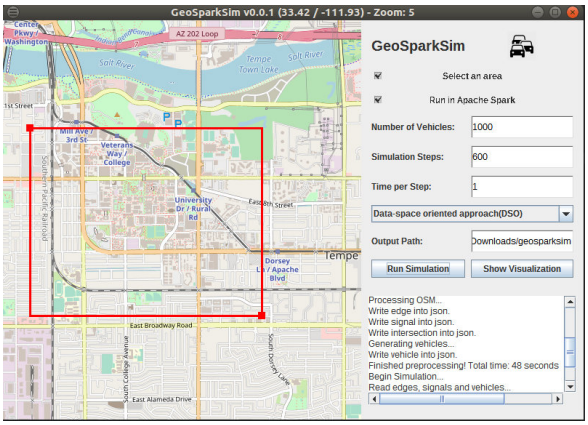


Figure 2: GeoSpark Sim user interface

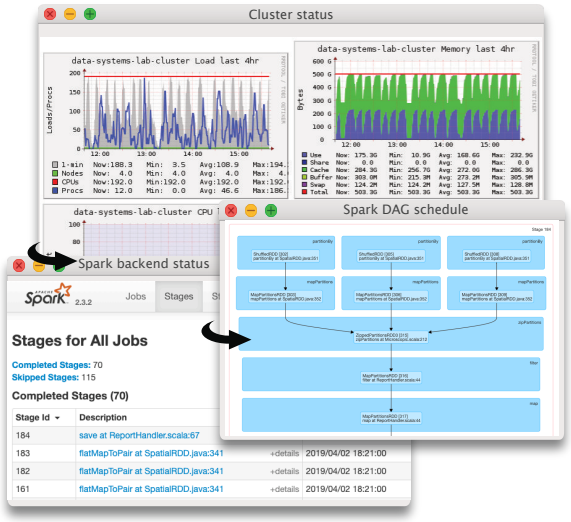


Figure 3: GeoSparkSim backend monitoring

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.

3 DEMONSTRATION

This section demonstrates the user interaction with GeoSparkSim.All GeoSparkSim components are packaged into a single JAR file. The user can easily load it into Spark cluster and start to enjoy the functionalities.

3.1 User-defined traffic model

GeoSparkSim by default uses intelligent driving model [6] and MOBIL [5] (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 own 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.

3.2 User interface

GeoSparkSim user interface consists of two panels (see Figure 2). The right panel includes the configuration options and a status bar. The user can enter the number of vehicles, the total simulation steps, time per step and vehicle initialization method. The total simulated period is equal to $simulation\ steps * time\ per\ step$. The example shown in the figure will simulate the 10-minute trajectories of 100K vehicles in parallel. When the simulation is triggered, the status bar at the bottom will print the real-time status of the on-going simulation. This way, the user can better learn the current progress of a simulation task. The left panel provides the map interface that allows the user to select an arbitrary simulation region. When the simulation begins, GeoSparkSim will download real OpenStreetMap road network data for the selected region and store it on HDFS.

It will then load the road network data into a Spark GraphX graph which contains a VertexRDD and EdgeRDD. After processing the road network data, GeoSparkSim will execute the simulation task in parallel and persist generated trajectories on HDFS periodically. Once it completes simulation, the status bar will print a notification and the user can opt to visualize the generated trajectories.

During the simulation, the user can also open GeoSparkSim backend monitoring tools (see Figure 3) to check the runtime status of the Spark cluster and explore the Spark Directed Acyclic Graph (DAG) schedule to learn the internal progress of GeoSparkSim.

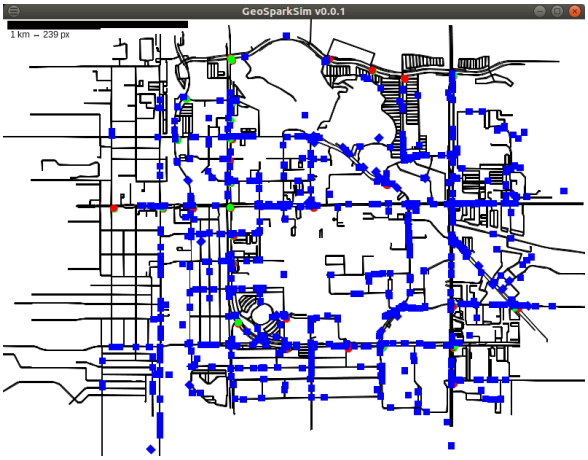


Figure 4: GeoSparkSim traffic visualization interface

3.3 Visualization interface

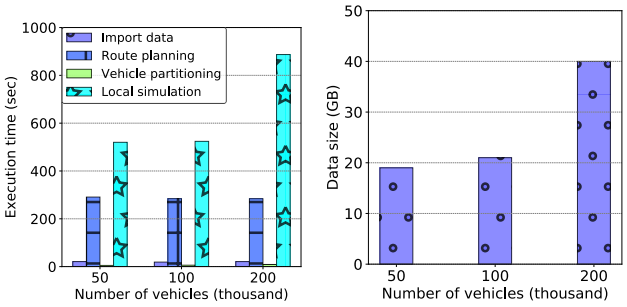
As illustrated in Figure 4, GeoSparkSim visualization interface will animate the movement of vehicles on a road network selected by the user (Arizona State University in this case). Once the simulation is completed, the user can click "Show visualization" bottom to call the visualization, this interface draws a graphical road network with traffic lights enabled and updates vehicle location and traffic signal at every time step. When a vehicle arrives at its destination, it will restart from its source. All vehicles will stop moving if the simulation data ends. Moreover, the user can freely zoom-in to a particular road intersection to check a traffic jam.

It is also worthing that the visualization interface runs in an offline fashion to guarantee a smooth user exploration experience. It starts playing the vehicle movements only after the entire simulation is completed. In addition, the visualization process works best for a small number of vehicles. When handling numerous vehicles, it will only visualize a sample set of simulation results.

3.4 Experiment

We conduct an experiment 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. This experiment studies the scalability of GeoSparkSim. We vary the number from 50 thousand to 200 thousand and measure the execution time and data size. The results are reported in Figure 5. We also show the time taken by each layer of GeoSparkSim. 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).

As shown in Figure 5a, 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 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



(a) Execution time **(b) Data size**

Figure 5: Performance on different numbers of vehicles

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 5b, as we increase vehicles in the VehicleRDD, the traffic data generated by GeoSparkSim also increases. This makes sense because GeoSparkSim has to provide more GPS locations at each simulation time step if there are more vehicles.

REFERENCES

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

[2] Düntgen, C., Behr, T., and Güting, R. H. Berlinmod: a benchmark for moving object databases. *VLDB J. 18*, 6 (2009), 1335–1368.

[3] Fu, Z., Yu, J., and Sarwat, M. Building a large-scale microscopic road network traffic simulator in apache spark. In *MDM* (2019).

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

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

[6] Kesting, A., Treiber, M., and Helbing, D. 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 (2010), 4585–4605.

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

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

[9] Lu, J., and Guting, R. H. Parallel Secondo: Boosting Database Engines with Hadoop. In *ICPADS* (2012), pp. 738 –743.

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

[11] Ramamohanarao, K., Xie, H., Kulik, L., Karunasekera, S., Tanin, E., Zhang, R., and Khunayn, E. B. Smarts: Scalable microscopic adaptive road traffic simulator. *TIST 8*, 2 (2017), 26.

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

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

[14] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *NSDI* (2012), pp. 15–28.