

*Dissecting GeoSparkSim: a scalable
microscopic road network traffic simulator
in Apache Spark*

Jia Yu, Zishan Fu & Mohamed Sarwat

Distributed and Parallel Databases
An International Journal of Data
Science, Engineering, and Management

ISSN 0926-8782

Distrib Parallel Databases
DOI 10.1007/s10619-020-07306-x



Your article is protected by copyright and all rights are held exclusively by Springer Science+Business Media, LLC, part of Springer Nature. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".



Dissecting GeoSparkSim: a scalable microscopic road network traffic simulator in Apache Spark

Jia Yu¹ · Zishan Fu¹ · Mohamed Sarwat¹

© Springer Science+Business Media, LLC, part of Springer Nature 2020

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 300 thousand vehicles over a very large road network (250 thousand road junctions and 300 thousand road segments) and outperform the existing competitors.

Keywords Spatio-temporal data · Apache Spark · Traffic model · Microscopic traffic simulation

✉ Jia Yu
jiayu2@asu.edu

Extended author information available on the last page of the article

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. Researchers from Microsoft Research spent more than five years on collecting 17621 trajectories over 182 volunteers [1]. Even if people manage to successfully collect a limited amount of historical data, it usually has low quality and only covers a small geographical area.

To remedy that, researchers turn to traffic data simulators which can easily generate massive synthetic road network traffic data. 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[4], 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[4] 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[5] 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[6] proposes to partition

the geographical space to uniform grids. This does not work well if the vehicles and road networks have a skewed distribution. SMARTS[7] 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[8] uses Hadoop MapReduce[9] and SMARTS[7] leverages simple TCP sockets. Apache Spark, on the other hand, provides a novel data abstraction called Resilient Distributed Datasets (RDDs)[10] 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 and simulated vehicles to RDDs. 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 data partitioning method to partition vehicles and the road network 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 attributes and spatial attributes of vehicles to handle the dynamic spatial distribution.
- A full-fledged prototype of GeoSparkSim is implemented in Apache Spark^{1,2}. Our experimental analysis shows that GeoSparkSim can simulate the movements of 300 thousand vehicles over a very large road network (250 thousand road junctions and 300 thousand road segments).

Giving this outlook, the rest of this paper is presented as follows: Section 2 studies the related work. An overview of GeoSparkSim is given in Sect. 3. Section 4 details the internals of VehicleRDD. Road network construction is depicted in Sect. 5. Section 6.2 illustrates how to determine the routes of numerous vehicles. The simulation-aware data partitioning method is explained in Sect. 7. Section 8 describes the microscopic simulation models in GeoSparkSim. Section 9 shows the graphic user

¹ Source code: <https://github.com/zishanfu/GeoSparkSim>.

² Demo video: <https://jiayuas.github.io/files/video/geosparksim-demo.mp4>.

interface. A comprehensive experimental analysis is given in Sect. 10. Section 11 concludes the paper.

2 Related work

2.1 State-of-the-art traffic simulators

Brinkhoff[2] is a framework for generating network-based moving objects. In Brinkhoff framework, the generated object distribution is correlated to the density of the network. It computes the fastest paths for moving objects which initially follows the uniform distribution and provides an interactive visualization interface for users. The input of Brinkhoff framework is the road network data and some relevant parameters. After triggering generation, source-destination pair will be generated by a motion computing algorithm. The shortest path between source and destination are computed by Dijkstra and A star path-finder algorithm. The trajectories will be collected and reported in text or database by Java Database Connectivity (JDBC). Brinkhoff generator preprocesses the road network to a compatible format, such as edge and node. Users can modify the configuration file and the parameters in UI to produce customized simulation.

BerlinMOD[3] is a benchmark for moving object databases based on SECONDO DBMS[11], an extensible DBMS architecture for large-scale data. BerlinMOD uses the approaches from Brinkhoff to create the start and destination node. The input of BerlinMOD is the start node, destination node and travel starting time. The system uses the Dijkstra algorithm, a well-known shortest path algorithm, to compute the route. It provides different sets of queries for benchmarking the moving objects. The user can also post SQL-like queries for moving objects.

Sumo[4] is an open-source continuous, microscopic and multi-modal traffic simulation package. Multi-modal means the movement model includes not only cars but also pedestrian, public transportation, etc. For example, a person may travel by walk or by car.

- *Components* The car-following model used in Sumo is Gipps-model[12] which moderates the safe velocity and helps to avoid collisions. Sumo also deploys traffic lights during the simulation. Sumo develops a network converter to convert various road network data into XML-description. Dijkstra algorithm is used to compute the shortest paths.
- *Usability* Sumo supports different types of moving objects, such as car, pedestrian, train, subway, ship, etc. Users can enter the types of moving objects, simulation scale and region. The user could customize data and explore the insights on a visualization interface. After entering the required parameters, Sumo prepares the road network and starts the simulation. When the simulation is finished, the interface will display the animated simulation.

TRANSIMS[5] is a distributed traffic microscopic simulator. TRANSIMS parallelizes the simulation process by utilizing MPI[13]. For 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 proposes methods to minimize message passing costs and achieve load balancing.

- *Graph partitioning* TRANSIMS uses graph partitioning to cut the geographical region into several similar size domains. It cuts the network streets in the middle of the edges rather than intersections. Each CPU computes the local simulation for a given time period then CPUs exchange the messages of boundaries, then continue to run local simulations.
- *Adaptive load balancing* To be efficient, the loads on different CPUs should be as similar as possible. TRANSIMS uses an adaptive algorithm to determine the load on each CPU. It first uses a naive approach to partition the graph for the first round local simulation then figures out better partitions based on the performance of the first local simulation.

MATSim[14] is an open-source toolbox to run large-scale traffic simulations. MATSim offers a set of the flexible toolkit that users can customize the modules and do personalized simulation job. MATSim makes the most of the CPUs by multithreading simulation jobs.

- *Simulation model* Generally, the approach to simulate moving objects in MATSim is to take advantage of the queue. MATSim applies a queue to execute the operations in road network edges, like a street. Each edge has a queue to report the entry time of moving objects. Adjacent edges collaborate to exchange moving objects to ensure the correctness of simulation. Based on this approach, MATSim designs QueueSim which is a fixed-increment time advance model. MATSim also provides another model in which the moving objects can change time steps according to events, such as entering a street, leaving a road street, etc.
- *Parallelization model* MATSim contains three main components, simulation units, messages and scheduler. Simulation units in MATSim are mainly about moving objects and road network edges. Messages are the information exchanged among different threads including vehicle simulation knowledge, such as a vehicle leave a street and enter another street. The scheduler is a message priority queue that sorts message time and type to control objects' actions. For example, the first moving object in the edge queue will be the first object to leave the edge because the object has the earliest entering time. By round robin to assign the same amount of workload to each thread, MATSim can handle simulation in parallel.

ParamGrid[6] is a distributed framework for large-scale microscopic traffic simulation.

- *Network division* In ParamGrid, a road network is divided into tiles of equal size zones with the number of rows and columns. The dividing process will generate the boundaries of zones. ParamGrid generates a set of the source-destination matrices which store the approximate simulation workload of the

vehicle. The matrix will be applied to the dividing process to balance the workload. When a vehicle travels across tiles, with knowledge of all boundaries, the global route thread will synchronize the status of these vehicles.

- *Components* ParamGrid uses [15], a suite of microscopic simulation modeling tools, to do traffic simulation jobs. It employs CORBA [16], a distributed model designed to facilitate the collaboration between systems, to distribute the workload to a computer cluster.
- *Architecture* ParamGrid follows the master-slave distributed computing architecture. The master manages three services, global routing service, CORBA naming service and event service. It first cuts the network, globally assigns the name and location to each tile (tiles are distributed in the cluster), handles cross tiles vehicles and provides a broadcast channel to synchronize the simulation time frame. The slaves take three plug-ins, object request broker, vehicle movement handler, simulation synchronization handler. A slave is responsible for managing vehicle movement, receiving transferred vehicles and synchronizing the simulation.

SMARTS [7] is a distributed large-scale microscopic simulator that is able to run jobs on multiple machines in parallel.

- *Architecture* SMARTS provides a comprehensive set of simulation features. It takes as input road network, routes, and a setup script. The road network data is extracted from OpenStreetMap [17] by loading the external OSM file. The vehicle routes are generated by the standard shortest path algorithm, Dijkstra algorithm. 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 configuration description of simulation parameters, such as maximum number of steps, number of generated random vehicles, maximum distance, number of updates per second, etc. After the data preparation, SMARTS uses several microscopic simulation models such as car-following, lane-changing, traffic light, route changing, traffic rules, and calibrations. The simulation results will be stored on disk and visualized in a graphical user interface.
- *Workload balancing* The start coordinates of vehicles are generated following road network distribution. Thus SMARTS assumes the simulation workload is approximately equal to the road network distribution. The road network in SMARTS is divided into a set of uniform grids ordered by z-curve. 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. In DS, the master doesn't play such a critical role and hold fewer pressures. The master only assigns the first round of simulation jobs, and workers will automatically run the simulation step by step.

Table 1 Comparison among different traffic simulators

Feature	Simulation Model	Scalability Scalability	Workload Partitioning	Partition Organization	Distribution Model
Brinkhoff[2]	Macroscopic	Single node	–	–	–
Sumo[4]	Microscopic	Single node	–	–	–
BerlinMOD[3]	Macroscopic	Distributed	Hash	Fixed	MapReduce[9]
TRANSIMS[5]	Microscopic	Distributed	Graph cut	Fixed	MPI[13]
MATSim[14]	Microscopic	Multi-thread	Uniform grids	Fixed	Thread sync.
ParamGrid[6]	Microscopic	Distributed	Uniform grids	Fixed	CORBA[16]
SMARTS[7]	Microscopic	Distributed	Z-curve	Fixed	TCP sockets
GeoSparkSim	Microscopic	Distributed	Quad-Tree	Dynamic	RDD

2.2 Comparison

Macroscopic traffic simulator Simulators in this category focus on general vehicular flow in a transportation road network. All vehicles drive in a similar manner and simply move from the sources to the destinations step by step. Brinkhoff simulator[2] 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[18] 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 vehicle and take into account many traffic events including lane changing, car following and traffic signals. Sumo[4] 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 1).

Non-spatial partitioning approach Some existing solutions partition the workload without taking into account the spatial proximity of the moving vehicles.

Parallel-BerlinMOD[8] integrates BerlinMOD with a distributed DBMS called Parallel-Secondo[8] 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[9]. 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[5] proposes to use graph cuts to partition the large road network then apply the same partitions to vehicles. It leverages MPI[13] 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[14] comes up with a method 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[6] uses a partitioning method similar to MATSim but utilizes CORBA[16] framework which internally uses RPC. SMARTS[7] 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 balance vehicles due to their skewed spatial distribution. GeoSpark[19] and SpatialHadoop[20] 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.

2.3 Distributed computation frameworks

Existing solutions Most of the existing traffic simulators are designed upon inefficient or inconvenient distributed programming models. Many of them still use message passing services and do not employ advanced computation models and job schedulers. SMARTS[7] leverages simple TCP sockets, TRANSIMS[5] uses MPI[13], and MATSim[14] only utilizes multi-thread synchronization. On the other hand, Parallel BerlinMod[8] uses Hadoop MapReduce[9]. Although Hadoop-based systems achieve high scalability, they still exhibit slow run time performance since it persists all the intermediate data on disk.

Apache Spark is a distributed general-purpose cluster computing framework which allows users to easily write distributed programs without being involved in the details of parallelism. It also can tolerate faults and scale out to many commodity machines. It is an implementation of Resilient Distributed Datasets (RDD)[21]. An RDD is an in-memory dataset that is partitioned across machines in a cluster. RDDs are immutable and fault-tolerant data structures that allow users to persist

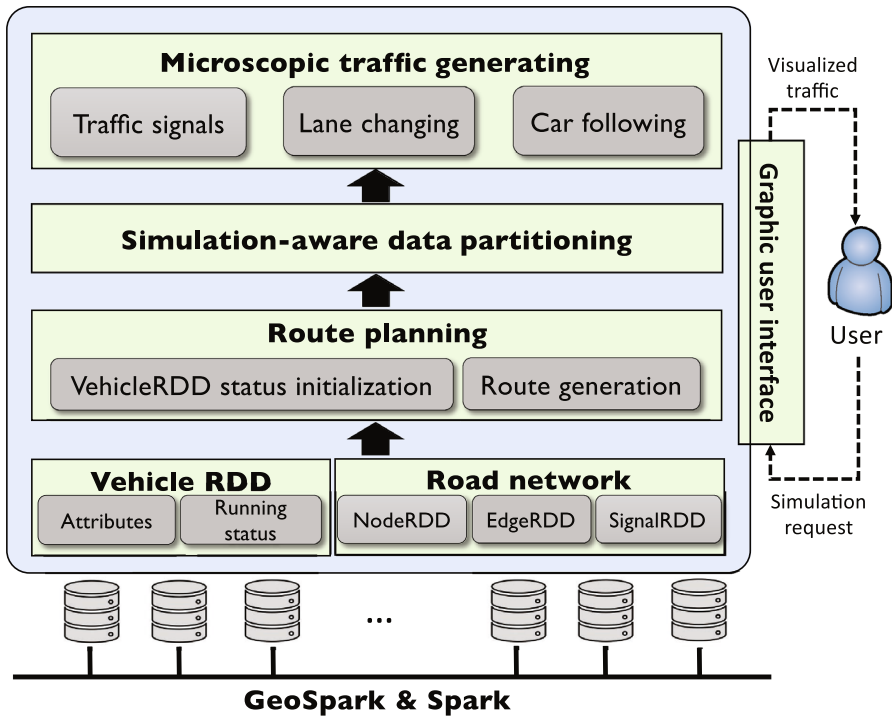


Fig. 1 GeoSparkSim architecture

intermediate results in memory to speed up distributed query processing. In-memory RDDs allow Spark to outperform existing models such as Hadoop MapReduce.

GeoSpark[19] is an in-memory cluster computing framework for processing large-scale spatial data. It employs a technique called Spatial RDD which extends Apache Spark RDD to support geospatial objects, indices and queries. A Spatial-RDD accommodates various types of spatial objects and provides spatial partitioning mechanism, such as R-tree, Quad-tree, and so on. It supports many spatial queries, such as spatial range, join, and k-nearest neighbors algorithm (KNN) queries and is able to run them in parallel.

3 GeoSparkSim architecture

GeoSparkSim consists of a Graphic User Interface (GUI) and five simulation components given below. 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 (Fig. 1).

VehicleRDD GeoSparkSim equips a specialized VehicleRDD which extends Spark RDD to accommodate millions of individual vehicle records. The driving status of each vehicle has several attributes such as velocity, safe distance, GPS location and so on. Each vehicle also possesses several personal characteristics such as politeness factor

and braking deceleration. The values of status change in a specific reasonable range according to the personal characteristics of vehicles. That way, each vehicle has its personalized behavior. The user can also define these behaviors via a vehicle configuration file.

Road network A road network describes the road situation of the specified simulation region. In GeoSparkSim, the road network is composed by three RDDs, NodeRDD, EdgeRDD, and SignalRDD. NodeRDD contains all road junction points, and EdgeRDD contains all road segments. Moreover, SignalRDD describes specific scenarios such as traffic lights in the road network. GeoSparkSim constructs this network using real geographical data from OpenStreetMap.

Route planning In this component, GeoSparkSim first creates the initial status for vehicles in VehicleRDD. Then it generates sources, and destinations for every vehicle following a particular spatial 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. GeoSparkSim computes routes for every source and destination pair on top of the index, gathers the results and attaches routes to corresponding vehicles.

Simulation-aware data 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 vehicles in VehicleRDD, GeoSparkSim co-partitions 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 data 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 patterns which consist 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.

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 configuration, 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.

Table 2 The attributes of a vehicle

Attribute	Explanation
Source	The starting coordinate of this vehicle
Destination	The final coordinate of this vehicle
Route	The planned route from the source to destination. It is a sequence of coordinates
Plate number	The unique identification of a vehicle. It is a random combination of 5 characters from English letters and number 0–9
Car length	The length of the vehicle (e.g., 1 m). This is used when compute the safe distance between two vehicles
Acceleration	The acceleration of this vehicle (e.g., 2 m/s^2)
Brake deceleration	The deceleration of this vehicle (e.g., -2 m/s^2)
Safe distance	If the distance between two cars is shorter than this safe distance (e.g., 1 meter), a car collision will happen.
Default speed	The default speed (e.g., 5 m/s). A vehicle will always accelerate to this speed. Then it will keep this speed if no event triggers a deceleration
Politeness factor	A value between (0, 1). A vehicle with a lower politeness factor will change its lane more frequently

4 VehicleRDD

GeoSparkSim VehicleRDDs are in-memory distributed datasets that extend traditional RDD to accommodate vehicle objects in Apache Spark. VehicleRDD consists of a set of vehicles with its randomized attributes and status such that yields arbitrary trajectories. Each VehicleRDD consists of many partitions and each partition contains thousands of vehicles. GeoSparkSim performs simulation in each partition and these partitions compute simulation in parallel by distributing the VehicleRDDs across the cluster.

4.1 Vehicle

A vehicle object in GeoSparkSim consists of two parts (1) vehicle attribute: defines driving behaviors. These attributes are parameters used in driving models and will affect the randomness factor in the running status. (2) running status: describes the current state of a vehicle. A status contains many attributes which can generate rich simulation information for the user. The value ranges of these attributes are determined by the driving model.

Attributes A vehicle has more than 10 attributes. Their names and explanation are given in Table 2. The values of these attributes (except source, destination and route) are randomly generated according to the value ranges specified in the vehicle configuration file. The user can also change the value range in the configuration file to produce customized simulation results. GeoSparkSim integrates two microscopic driving models, IDM[22] and MOBIL[23], in its simulation algorithm. The attributes listed in the table are the parameters used in the driving models. The simulation algorithm in Sect. 8 will utilize these attributes to generate vehicle statuses at different time steps.

Table 3 The running status of a vehicle

Status	Explanation
Current position	The current coordinate of this vehicle
Current lane	The lane that the vehicle is going through now. An edge may have many lanes
Current edge	The edge that the vehicle is going through now
Current path	The edges that the vehicle will go through in the current temporal partition (explained in Sect. 8)
Current path length	The length of each edge in the current path
Current speed	The current speed of the vehicle
Current signal in front	Indicates whether there is a red traffic light in front

Running status A running status is the status of a vehicle at a time step. The detailed items recorded in a running status is given in Table 3. During the simulation, every vehicle moves along its planned route over and over. But at every time step, it may stop at different traffic signals, run on separate lanes and generate various acceleration/deceleration events based on microscopic driving models.

4.2 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 parent VehicleRDD. The running status computation uses microscopic simulation models and will be detailed in Sect. 8. In other words, a VehicleRDD is a snapshot of current vehicle movements over the road network.

5 Road network

A road network is a graph that describes road paths and junctions. It is the fundamental infrastructure of any traffic simulator. An actual road network is extremely complex and requires tremendous efforts on the ETL phase (Extract, Transformation and Load). In order to achieve high performance, GeoSparkSim loads all road network elements to three RDDs, NodeRDD, EdgeRDD and SignalRDD.

5.1 Raw road network data

GeoSparkSim supports OpenStreetMap (OSM) XML road network data format, one of the most common road network formats. OpenStreetMap provides a thorough

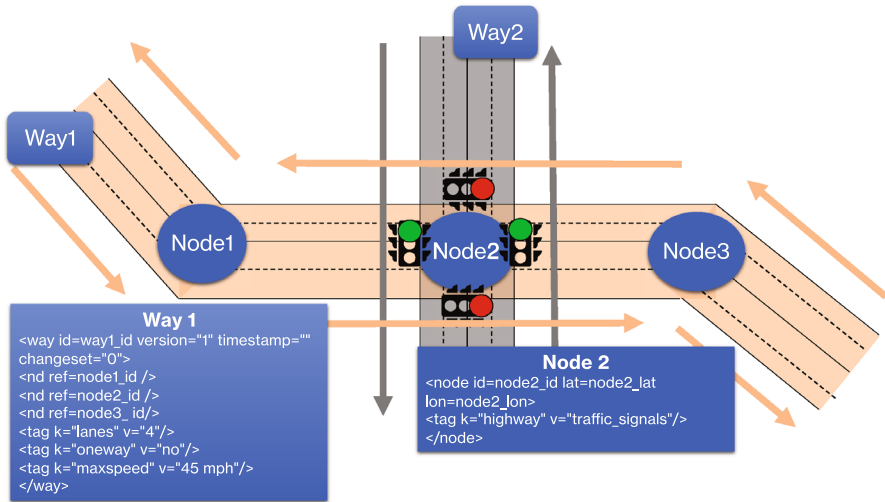


Fig. 2 The raw road network data

description of road networks on the earth. Figure 2 is an example of OSM road network information. After a user selects the region, GeoSparkSim will fetch XML data from OpenStreetMap and use a sinker software provided by OSM to transform the data to structured tables. The transformed data is saved as parquet format in HDFS or local disk (see Fig. 3).

An OSM road network consists of a node table and a way table. The former contains all nodes in the selected region and the latter includes all ways (e.g., Wall Street) in this region. A node usually is the junction, turning point or end point of the way. A node consists of id, latitude, longitude, and tags. A way consists of a sequence of edges marked out by nodes. A node on this way is a junction where two or more edges meet. These nodes are the vertex in the graph.

5.2 Road network in RDDs

The raw OSM data contains nodes and ways, but GeoSpark road network needs nodes, edges and signals. To achieve that, GeoSparkSim reads nodes and ways into Spark and runs a set of data cleaning and transformation to obtain specialized RDDs (see Fig. 3). Nodes represent intersections along the roads and edges represent the segments that connect two intersections.

Table 4 Node object

Attribute	Explanation
ID	Unique identification
Coordinate	Longitude and latitude
Intersect	A Boolean value. Some nodes just connect two edges on the same way but not the intersection of two ways.
Type	A node can be on a highway or residential street

Algorithm 1: Transform raw OSM data**Data:** raw nodes, raw ways**Result:** nodes, links, lights

- 1 Filter traffic signal nodes from raw nodes;
- 2 Break raw ways to edges;
- 3 Join nodes with edges by node ids;
- 4 Compute the distance of each edge;
- 5 **return** *Road Network*(*nodeRDD*, *linkRDD*, *lightRDD*)

Road network in GeoSparkSim RDDs A road network in GeoSpark consists of three specialized RDDs: (1) NodeRDD contains all needed nodes from an arbitrarily selected region, and each node is a road junction that connects two edges. Each node has four attributes as shown in Table 4. (2) NodeRDD accommodates all necessary edges, and each edge is a road segment which is a straight way between two nodes. Each edge has many attributes as depicted in Table 2. (3) SignalRDD includes all signal nodes in the simulation region. Each signal contains three main attributes (1) the node ID (2) the controlled way ID (3) coordinate.

Algorithm of transforming OSM data GeoSparkSim equips an algorithm (see Algorithm 1) to clean the original OSM data and transform to three specialized RDDs. Figure 2 is an example of a small road network. Way1 and way2 cross each other, and there are three nodes. Node2 is the shared node and intersection for way1 and way2. Way1 has three nodes in the diagram, node1, node2 and node3 with four lanes and 45 miles per hour maximum speed. The algorithm slices way1 into 4 edges, node1 to node2, node2 to node3, node3 to node2 and node2 to node1. Then it marks each edge with 2 lanes and 45 mph maximum allowed speed and computes the distance of this edge.

1. *Process nodes* Read the node table to Spark, then filter the nodes based on the tag column. Signal nodes and regular road junctions are put in SignalRDD and NodeRDD, respectively.
2. *Process ways to edges* After identifying all ways, we need to convert the ways to edges. A way should be chopped to directed edges. In the way transformation, we extract way id, way tags and node sequence. The tags may include speed limits, number of lanes, one-way and so on. The detailed information of the road is stored in each edge of this way. Eventually, GeoSparkSim obtains an edge table (see Table 5).

Table 5 Edge object

Attribute	Explanation
ID	Unique identification
Head	Node
Tail	Node
Distance	The length of this edge
Speed limit	Mile per hour
Lane	The number of lanes on this edge

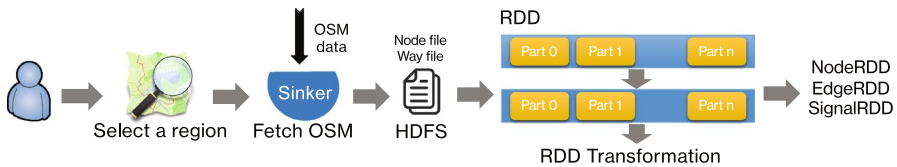


Fig. 3 Steps to process a road network in GeoSparkSim

3. *Join by node ids* The edges from the last steps have head and tail nodes but do not have the distance information. The distance is an important metric used in route planning. GeoSparkSim will do a join between the edge table and node table such that each edge can obtain the coordinates of its head and tail.
4. *Compute distances* GeoSparkSim then computes the distance of each edge according to the coordinates. The final edge table will keep the distance column and all other existing columns but drop the coordinates information.

6 Route planning

Vehicles in GeoSparkSim are moving objects. At each time step, each vehicle appears at a place (e.g., coordinate). A vehicle follows its route step by step and disappears when reaching the destination (e.g., coordinate). Therefore, before the simulation starts, GeoSparkSim first needs to decide the initial locations of vehicles and plan their routes during the simulation.

6.1 VehicleRDD Initialization

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

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)[2]. The DSO approach first randomly generates source points. Then it runs map matching to match points to their nearest nodes

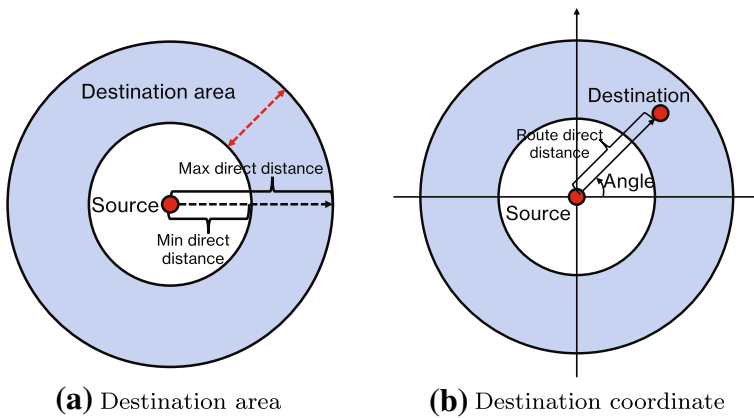


Fig. 4 Destination coordinate system. The blue ring is the destination area (Color figure online)

in the road network. GeoSparkSim can also apply a spatial distribution, such as the distribution of buildings, when generating random sources. Regions with more buildings will produce more source nodes. The Network-Based approach randomly selects nodes (road junctions) as sources. GeoSparkSim provides these two options for the user. By default, DSO is enabled in GeoSparkSim.

Destination area The destination of a vehicle should not be too far or too close from its source otherwise the simulation is not very meaningful. To yield reasonable routes, GeoSparkSim uses the direct distance between the source and destination to compute the destination area, which is the area that covers all possible destinations. As shown in Fig. 4a, we can draw a blue ring using the source as the center. The radius of the inner circle is the minimum direct distance which is always 1 kilometer. The radius of the outer circle is $\frac{1}{10}$ of the diagonal of the user-selected region. GeoSparkSim takes a random point in the destination area as the destination of the vehicle. To make sure the road network always covers the destinations, GeoSpark adds a boundary buffer, max direct distance - min direct distance, to the region selected by a user. That way, GeoSpark will download a little larger road network but cover all possible destinations.

Destination coordinate The destination coordinate of a vehicle is determined by the source coordinate, route direct distance, and the angle [0, 360] (see Fig. 4b). Route direct distance is a random value between the minimum direct distance and maximum direct distance. The angle is also a random value between 0 and 360. To be specific, the destination coordinate is computed by the equation below.

$$\begin{aligned} longitude &= source.longitude + direct_distance * \sin(angle) \\ latitude &= source.latitude + direct_distance * \cos(angle) \end{aligned}$$

A destination coordinate randomized by this method will fall in the destination area.

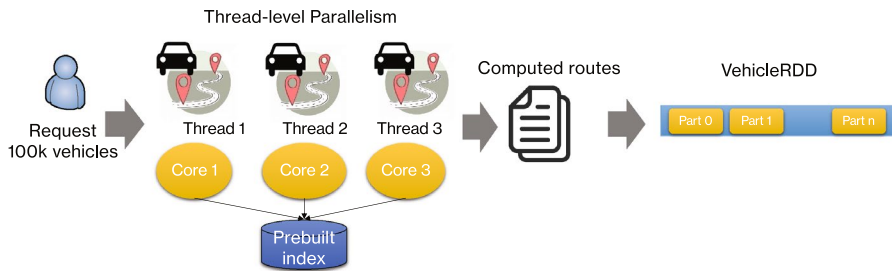


Fig. 5 Parallelized route generation

6.2 Route generation

After initializing VehicleRDD, GeoSparkSim will generate the shortest path for every pair of source and destination. For the sake of routing speed, GeoSparkSim leverages GraphHopper[24], an open-source route planning library to compute the shortest path for every source and destination pair.

GraphHopper routing engine Graphhopper supports various routing algorithms including Dijkstra and A star algorithms, for different purposes. GeoSparkSim uses Graphhopper to load processed road network and then builds an index over the imported road network. This index pre-computes short paths among common road junctions. Compared to other open-source routing engines, Graphhopper has much less preprocessing time.

Parallel route generation GraphHopper can generate a single shortest path every time based on the prebuilt index. In order to speed up the shortest path computing process and leverage idle CPU cores, GeoSparkSim parallelizes the process. By default, GeoSparkSim uses a thread pool with a number of threads (8 threads by default) to generate shortest paths for many vehicles in parallel. Each thread queries the same pre-built index and computes its own route. Figure 5 is an example of generating vehicles in parallel. A user requests the simulation with 100,000 vehicles. GeoSparkSim will divide the routing workload to 3 threads. Each thread sequentially computes the shortest path for vehicles assigned to this thread. When the threads finish their job, GeoSparkSim will collect results and convert to VehicleRDD. Then the thread pool will be shut down immediately.

7 Simulation-aware data partitioning

Workload distribution in GeoSparkSim has a significant impact on performance. Properly balancing the workload among machines can greatly shorten the execution time and reduce resource consumption in terms of memory footprint and network bandwidth.

7.1 Different data partitioning approaches

Non-spatial partitioning The default data partitioning method in Spark such as hash partitioner and round-robin partitioner exhibits good performance for regular ETL queries but does not well handle microscopic traffic simulation because it does not take into account the spatial proximity. A non-spatial partitioner cannot partition both VehicleRDD and road network in the same way. That means, during the simulation period, a vehicle may not find the corresponding road network on the same machine unless the entire road network is duplicated to this machine. Duplicating the entire large road network to every machine in this cluster is not scalable and dramatically slows down the processing speed.

Spatial partitioning Spatial partitioning in GeoSpark[19] partitions objects in a RDD by their spatial proximity. Nearby objects are put in the same partition. Its basic idea is to partition the datasets according to a grid file. The spatial partitioning approach can partition both VehicleRDD and road network (NodeRDD, EdgeRDD and SignalRDD) using the same partitioning approach. This way, vehicles will be on the same machine with the corresponding part of the road network. The grid file used in spatial partitioning has to adapt to a target static spatial distribution. There are several choices when selecting the target static distribution:

- *Partition by road network* The distribution of NodeRDD and EdgeRDD is static but the distribution of NodeRDD and EdgeRDD is very different from VehicleRDD because most of the vehicles stay on the major streets. The place which has many nodes and edges may not have many vehicles passing by.
- *Partition by source coordinates of vehicles* Although the source coordinates of vehicles are static, the distribution of vehicles changes over time. This will cause unbalanced workload with increased simulation steps.
- *Partition by planned routes of vehicles* Planned routes of vehicles are trajectories that show a static distribution. A trajectory is a collection of coordinates that vehicle will follow. Since the grid file is built upon the Minimum Bounding Rectangles (MBRs) of objects, the MBRs of these trajectories are mostly overlapped. It is not easy to make a clear cut among overlapped trajectories.

7.2 Simulation-aware partitioning

The data partitioning layer in GeoSparkSim partitions the workload according to both spatial and temporal attributes. It takes as input a VehicleRDD and road network (NodeRDD, EdgeRDD, and SignalRDD) and spatially partitions these RDDs according to planned vehicle routes in the upcoming temporal partition. GeoSparkSim periodically invokes this layer to repartition these RDDs to make sure that they always carry out balanced partitions. 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

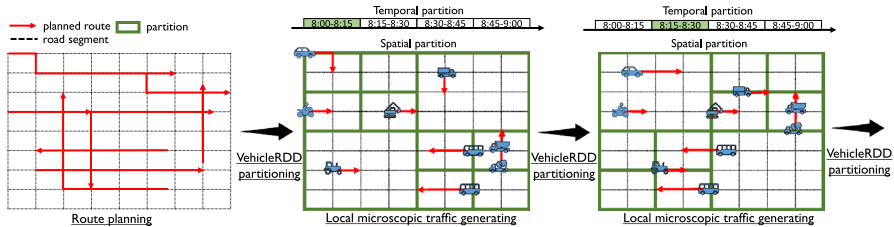


Fig. 6 GeoSparkSim temporal partitioning

(2) allow local microscopic traffic simulation inside each partition (3) support dynamic vehicle movement distribution

To be precise, GeoSparkSim sets a temporal partition size (say., 2 min), divides the simulation period to multiple temporal partitions, and runs the simulation for each temporal partition one by one. Data in each temporal partition is partitioned by spatial proximity. After completing the simulation in each temporal partition, GeoSparkSim re-runs spatial partitioning on current data to adapt to new spatial distribution. The detailed steps are as follows:

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 Fig. 6). It is very hard to do spatial partitioning 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 and road network. It includes the following sub-steps: (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 and road network: 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. Edges that intersect several partitions will be duplicated. 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 machine. Since VehicleRDD and road network are partitioned according to the spatial proximity of the estimated short-term routes in this spatial-temporal partition, 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 all RDDs for the upcoming temporal partition. This is to maintain the workload balance because these vehicles keep moving in the simulated region and their spatial distribution varies in different temporal partitions.

Figure 6 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-min temporal partitions. Then, for temporal partition 1 (8:00 am to 8:15 am), GeoSparkSim runs Step 2-4 to partition VehicleRDD and road network (see the middle part in Fig. 6) and then performs local simulation in each machine. GeoSparkSim will execute the same steps for the rest of the temporal partitions one by one.

7.3 Determine the best temporal partition period

GeoSparkSim divides the simulation period into a set of uniform temporal partitions. It re-runs spatial-partitioning on VehicleRDD and road network after each temporal partition. The repartitioning step should not happen too frequently because each re-partitioning requires a time-consuming data shuffle. GeoSparkSim proposes Algorithm 2 to determine the best temporal partition size (e.g., repartition period). GeoSparkSim chooses five temporal partition size based on the simulation period, then it takes 1% samples from vehicleRDD, and runs the simulation on this sample. Eventually, GeoSparkSim finds the best one in terms of minimum simulation time.

Algorithm 2: GeoSparkSim Repartition Algorithm

Data: Simulation period (SP), VehicleRDD, NodeRDD, LinkRDD and SignalRDD
Result: Best temporal partition size

```

1 Compute the five temporal partition size candidates:  $\frac{SP}{10}, \frac{SP}{8}, \frac{SP}{6}, \frac{SP}{4}, \frac{SP}{2}$ ;
2 foreach temporal partition size candidate do
3   | execution time = Run the simulation on VehicleRDD sample;
4   | if execution time < minexecutiontime then
5     |   | min execution time = execution time ;
6   | Best temporal partition size = min execution time;
7 return Best temporal partition size
```

8 Distributed microscopic traffic simulation

8.1 Simulation algorithm

GeoSparkSim periodically runs repartitioning for vehicleRDD and applies the same partition mechanism to VertexRDD, LinkRDD and SignalRDD. After partitioning the RDDs, GeoSparkSim is ready to run the local microscopic simulation on each RDD partition. All vehicles will follow their short-term trajectories 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 s). It also indicates the number of simulation iterations needed to be run by GeoSparkSim. The simulation Algorithm 3 then runs a set of iterations and in each iteration, it first updates all signals in this iteration. Some signals will turn to different colors. For each vehicle, the algorithm then goes through other objects (signals, edges, and vehicles) in the surrounding environment to check the following microscopic driving models (1) car-following: if vehicles in the front invade the safe distance buffer of this vehicle, it will decelerate. 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. (2) traffic signal: a vehicle will decelerate if the traffic lights in the front changes to red. (3) lane-changing: the algorithm considers multi-lanes and will check the possible lane change opportunities and move the vehicle to the new lane. After taking into account these models, GeoSparkSim then computes the current position of this vehicle in the lane by time step and speed. Finally, GeoSparkSim updates the vehicle with necessary status information such as speed, lane, and position. After the local simulation on each RDD partition, GeoSparkSim will update VehicleRDD status and persist the simulation results on HDFS. If some vehicles reach their destinations before the simulation completes, GeoSparkSim will restart them from their sources.

Algorithm 3: GeoSparkSim Simulation Algorithm

```

Data: VehicleRDD and road network
Result: a sequence of VehicleRDDs
1 foreach temporal partition do
2   Partition VehicleRDD by vehicle planned routes in this temporal partition;
3   Apply same partition to NodeRDD, EdgeRDD and SignalRDD;
4   Zip VehicleRDD, NodeRDD, LinkRDD and SignalRDD by spatial proximity;
5   foreach partition in zipped RDDs do
6     foreach iteration in temporal simulation do
7       foreach signal S do
8         Update the timing and light of S;
9         foreach vehicle V do
10          if V not arrive destination then
11            Check the neighbor vehicle ahead;
12            Check the traffic light ahead;
13            Check the lane changing opportunity;
14            Compute V position, acceleration and velocity;
15            Compute coordinate by P and new lane id.;
16          else
17            Update V's current position to its source;
18          Update vehicleRDD and SignalRDD by the latest status;
19          Write the current VehicleRDD to HDFS;
20 return Simulation results

```

8.2 Microscopic simulation models

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

Car-following The car-following model uses the Intelligent-Driver Model (IDM) [23] 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. If there are no nearby vehicles, the model may accelerate the vehicle.

Lane-changing GeoSparkSim uses a general lane-changing model called MOBIL[24]. A vehicle changes the lane based on the politeness factor. 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 based on the politeness factor. Otherwise, it will stay in the same lane. Some lanes information is from OpenStreetMap and default lane is bidirectional with one lane each direction. In addition, the direction information of lanes will also be considered because all lanes are directed in a real road network. Vehicles can only move to the lanes that have the same direction as their current lanes. Taking the car following decision from IDM, GeoSparkSim uses MOBIL to calculate if the targeted new lane is allowed.

Traffic signals During the simulation, GeoSpark will update the signals, and the vehicle will respond to corresponding lights. GeoSparkSim assigns initial signals in SignalRDD randomly, and it exactly follows the green-yellow-red sequence. The time duration for green light is 55 s, yellow light 5 s and red light 60 s. 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

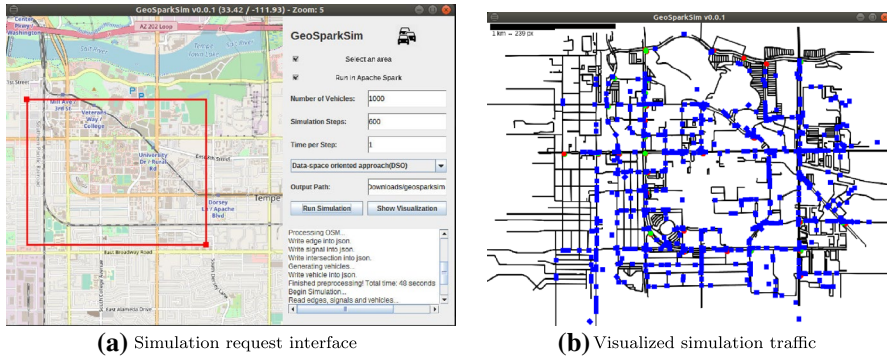


Fig. 7 GeoSparkSim graphic user interface

is red. Vehicles keep the same passing speed at a green light and check the safe distance to pass at a yellow light.

9 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 7a is a GUI example.

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 users can describe their personalized simulation request by checking several options and filling the parameters. The map panel on the left 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 right-bottom shows the status of current simulation job and reports the current ongoing task.

Issue a simulation request The user can enter the number of vehicles and simulation period. He or she can also select VehicleRDD initialization approach 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 graphical frame in order to render all road network elements and simulated vehicle locations every simulation time step. In each time step, the graphical painter will update and redraw the moving vehicles based on the current simulation step. Traffic signals will also be updated. The simulation panel is created by Java Swing which can run on any platform. 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

Table 6 Parameters

Parameter	Range
Number of vehicles (thousand)	<i>100</i> , 200, 300
Time step (s)	<i>1</i> , 0.8, 0.6, 0.4, 0.2
Number of partitions	<i>1000</i> , 1500, 3000
Temporal partition size (min)	<i>1</i> , 2, 4, 8, 10
Simulation period (min)	<i>10</i> , 30, 60, 120

projection changes. Figure 7b is a traffic visualization example using the road network in Arizona State University.

10 Evaluation

10.1 Environment setting

Compared approaches (1) GeoSparkSim: the system proposed in this paper. (2) Sumo[4]: a popular single-machine microscopic traffic simulator (3) SMARTS[7]: a distributed microscopic traffic simulator using static z-curve partitioning.

Cluster 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. Several monitoring software, such as Spark history server and Ganglia, are used to measure the cluster status. Ganglia is a distributed monitoring system that can check current CPU, memory, and network utilization of the cluster.

Parameters We change the following parameters throughout the experiments (listed in Table 6, default values are in italics): (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 min and 1 s per step, it requires 600 simulation iterations. If it is 0.8 s per step, 10 min need 750 steps. (3) The number of partitions: the number of partitions in all RDDs (4) Temporal partition size (min): the size of temporal partitions in GeoSparkSim. (5) Simulation period: the overall period that GeoSparkSim wants to simulate.

Default values The numbers underlined in Table 6 are the default values used in the experiment. By default, GeoSparkSim sets the temporary partition size to 2 min. In other words, it will invoke the spatial partitioning function to repartition the VehicleRDD and road network after simulating every 2-min traffic. For instance, assume a simulation workload (time step = 1 s, temporal partition size = 2 min, 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 s. GeoSparkSim will repartition the

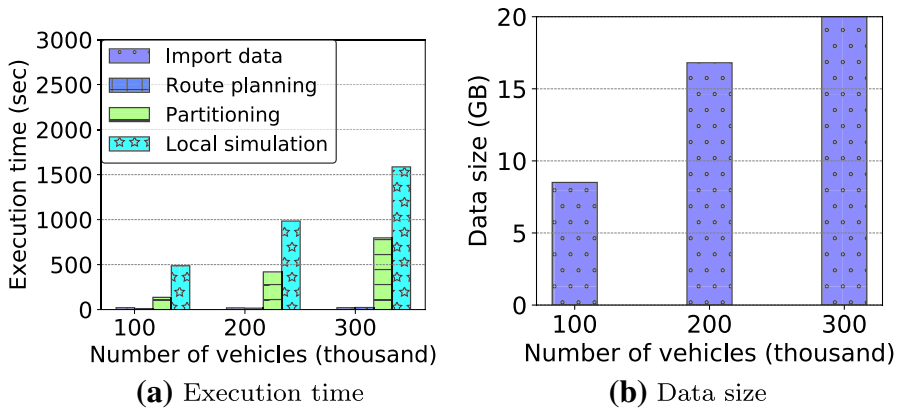


Fig. 8 The impact of the number of vehicles

vehicleRDD and road network 8 times (8:00, 8:02, ..., 8:14). Besides, GeoSparkSim uses the KDB-tree partitioning method from GeoSpark[19] in its spatial partitioning step. If the simulation period is less than the repartition period, GeoSparkSim will not invoke the repartition 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 GeoSparkSim simulation workload. (2) Data size: it is the size of generated traffic data.

Tested data We use the road network of the Phoenix metropolitan area in the experiment, comprising a total area of 250 thousand road junctions and 300 thousand road segments.

10.2 The impact of the number of vehicles

In this experiment, we study the impact of different numbers of vehicles. We vary the number from 100 thousand to 300 thousand and measure the execution time and data size. The results are reported in Fig. 8. We also show the time taken by each layer of GeoSparkSim. During the simulation, GeoSparkSim partitions the VehicleRDD twice (temporal partition size is 5 min).

As shown in Fig. 8a, both data importing part and route planning part take almost constant time. This happens because we use the same road network for all experiments. After loading the network, GeoSparkSim leverages GraphHopper to build a shortest path index on it and creates a thread pool to access the index in parallel. The thread pool significantly shortens the route planning time. 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.

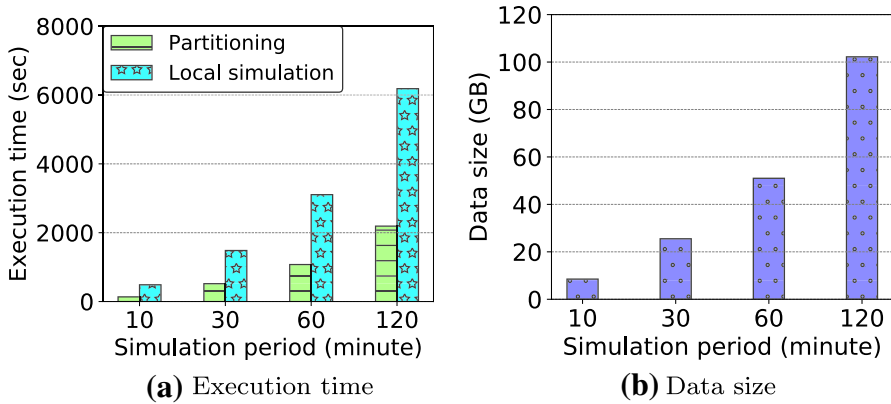


Fig. 9 The impact of simulation period

As depicted in Fig. 8a, 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.

10.3 The impact of simulation periods

In this experiment, we further examine the impact of different simulation periods. We vary the simulation period from 10 to 120 min. One hundred thousand vehicles are simulated, the temporal partition size is 2 min, the number of partition is 1500 and the time step is 1 s. We report the results in Fig. 9. Data importing time and route planning time are omitted in this figure because it is reported in Fig. 8a.

As shown in Fig. 9a, 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 partitioning time is also longer for the more extended simulation period. This happens because GeoSparkSim repartitions the VehicleRDD, 5, 15, 30, and 60 times for different periods.

As depicted in Fig. 9b, 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.

10.4 The impact of time steps

In this experiment, we explore the impact of the different simulation time step. We vary the time step from 1 to 0.2 s. One hundred thousand vehicles are simulated, the

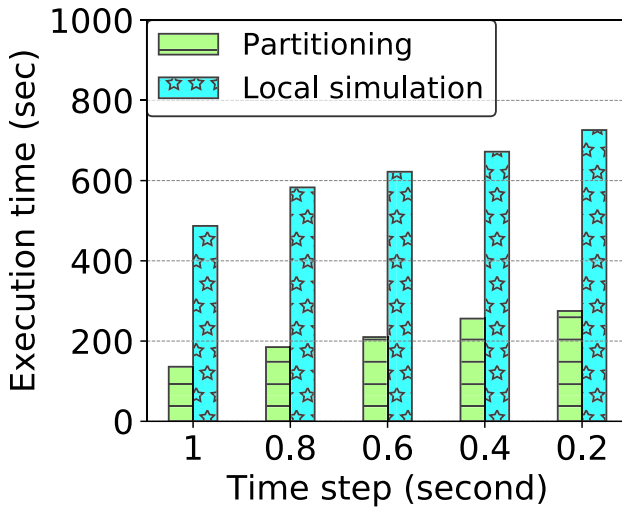


Fig. 10 The impact of time steps

temporal partition size is 2 min, the number of partition is 1500 and the simulation period is 10 min.

As described in Fig. 10, 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. Data importing time and route planning time are omitted in this figure because it is reported in Fig. 8a.

10.5 The impact of the number of spatial partitions

In this experiment, we study the impact of different numbers of spatial partitions. We use three different numbers of partitions, 1000, 1500, and 3000. 100 thousand vehicles are simulated, the time step is 1 s, the repartition period is 1 min and the simulation period is 10 min. We report the results in Fig. 11. In this figure, the time spent on partitioning is increasing when there are more partitions. This is reasonable because more partitions will incur more shuffling overhead. However, too few partition or too many partitions will both lead to high local simulation time. This makes sense because too many partitions will generate small spatial boundaries of partitions. This eventually leads to lots of duplicated vehicle routes and edges such that slows down the simulation performance. On the other hand, too few spatial partitions may lead to unbalanced partitions.

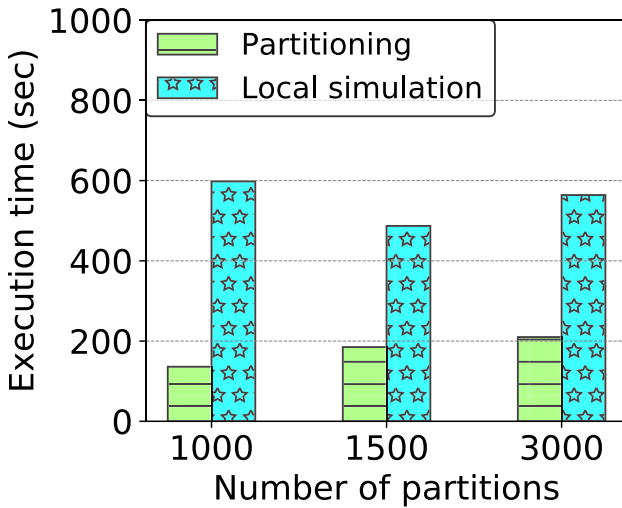


Fig. 11 The impact of spatial partitions

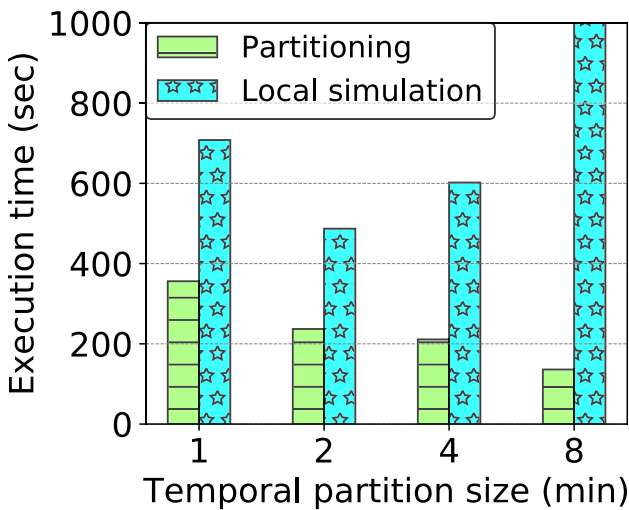


Fig. 12 The impact of temporal partition size

10.6 The impact of temporal partition sizes

In this experiment, we analyze the impact of different temporal partition sizes. We vary the temporal partition sizes from 1 to 10 min, 100 thousand vehicles are simulated, the time step is 1 s, the number of partition is 1500 and the simulation period is 10 min. Results are shown in Fig. 12.

As shown in Fig. 12, the partitioning time decreases when the temporal partition size increases because a smaller temporal partition size means less re-partitioning

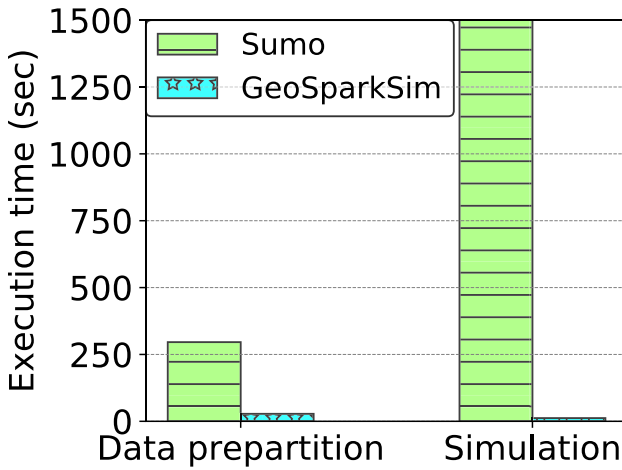


Fig. 13 Sumo and GeoSparkSim

steps. However, the local simulation shows the shortest execution time when the temporal partition size is 2. This makes sense since large temporal partition will lead to more vehicle route overlaps and small temporal partition will cause unnecessary data computation.

10.7 Sumo and GeoSparkSim

In this experiment, we compare the data preparation time and simulation time between GeoSparkSim and Sumo, a popular single-machine microscopic simulator, using one thousand vehicles. The simulation part includes simulation-aware partitioning and local simulation. The simulation period is 1 min and the time step is 1 s. Sumo and GeoSparkSim adopt same driving models, IDM and MOBIL. In Fig. 13, GeoSparkSim has 10 times faster data preparation speed and more than 100 times faster simulation speed. This is reasonable because GeoSparkSim parallelizes both data preparation and simulation.

10.8 SMARTS and GeoSparkSim

$$Speed\ up\ factor = \frac{simulation\ period}{time\ cost} \tag{1}$$

In this experiment, we compare the simulation time and speedup factor between GeoSparkSim and SMARTS, a distributed microscopic traffic simulator. We simulate 100 thousand vehicles, and vary the simulation period from 10 to 120 min. The data preparation time including vehicle generation and road network is omitted since the difference between GeoSparkSim and SMARTS is similar to that between GeoSparkSim and Sumo.

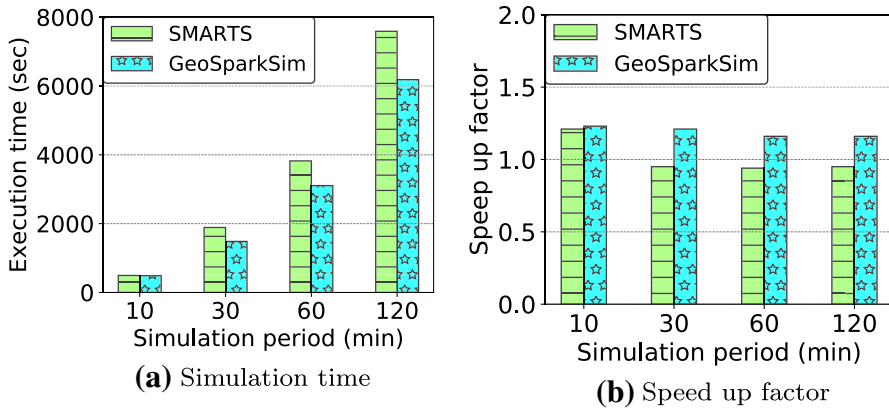


Fig. 14 SMARTS and GeoSparkSim

Speedup factor is defined in Eq. (1)[7]. It is the result of dividing the requested simulation period by the actual execution time. For example, if the user requests 20 min of simulation and the simulator takes 10 min to generate the traffic, the speedup factor is 2. SMARTS and GeoSparkSim use the same traffic models, IDM and MOBIL. The simulation output for SMARTS is a collection of vehicle simulation GPS coordinate by time steps, while GeoSparkSim contains not only the GPS trajectories but also vehicle events in each step, such as acceleration and velocity. As depicted in Fig. 14a, GeoSparkSim is 1.2 times faster than SMARTS when the simulation period is 120 min. GeoSparkSim has a better performance on longer simulation periods because GeoSparkSim takes into account the dynamic spatial distribution and tries to balance workload by periodically repartitioning the data while SMARTS only partitions the data once. Figure 14b also indicates that GeoSparkSim can speed up the simulation even when the request period is very long.

11 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 300 thousand vehicles over a very large road network (250 thousand road junctions and 300 thousand road segments) and outperform the existing competitors.

Acknowledgements This work is supported by the National Science Foundation (NSF) under Grant 1845789.


References

- Zheng, Y., Xie, X., Ma, W.Y.: Geolife: a collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* **33**(2), 32 (2010)
- Brinkhoff, T.: A framework for generating network-based moving objects. *GeoInformatica* **6**(2), 153 (2002)
- Düntgen, C., Behr, T., Güting, R.H.: BerlinMOD: a benchmark for moving object databases. *VLDB J.* **18**(6), 1335 (2009). <https://doi.org/10.1007/s00778-009-0142-5>
- Krajzewicz, D., Hertkorn, G., Rössel, C., 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)*, pp. 183–187 (2002)
- Nagel, K., Rickert, M.: Parallel implementation of the TRANSIMS micro-simulation. *Parallel Comput.* **27**(12), 1611 (2001)
- Klefstad, R., Zhang, Y., Lai, M., Jayakrishnan, R., Lavanya, R.: A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation. In: *Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE*, IEEE, pp. 813–818 (2005)
- Ramamohanarao, K., Xie, H., Kulik, L., Karunasekera, S., Tanin, E., Zhang, R., Khunayn, E.B.: Smarts: scalable microscopic adaptive road traffic simulator. *ACM Trans. Intell. Syst. Technol.* **8**(2), 26 (2017)
- Lu, J., Güting, R.H.: Parallel secondo: boosting database engines with Hadoop. In: *International Conference on Parallel and Distributed Systems*, pp. 738–743 (2012)
- Hadoop (n.d.). <https://hadoop.apache.org/>
- Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauly, M., Franklin, M.J., Shenker, S., Stoica, I.: 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)
- Güting, R.H., Almeida, V., Ansoerge, D., Behr, T., Ding, Z., Hose, T., Hoffmann, F., Spiekermann, M., Telle, U.: Secondo: an extensible dbms platform for research prototyping and teaching. In: *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, IEEE, pp. 1115–1116 (2005)
- Gipps' model (2019). https://en.wikipedia.org/wiki/Gipps%27_model
- Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kamradur, P., Barrett, B., Lumsdaine, A., Castain, R.H., Daniel, D.J., Graham, R.L., Woodall, T.S.: 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)
- Waraich, R.A., Charypar, D., Balmer, M., Axhausen, K.W.: Performance improvements for large scale traffic simulation in MATSim. In: *9th STRC Swiss Transport Research Conference: Proceedings*, vol. 565 (Swiss Transport Research Conference, 2009), vol. 565
- Paramics Microsimulation (2019). <https://www.paramics.co.uk/en/>
- Vinoski, S.: CORBA: integrating diverse applications within distributed heterogeneous environments. *IEEE Commun. Mag.* **35**(2), 46 (1997)
- OpenStreetMap (2019). <http://www.openstreetmap.org/>
- Mokbel, M.F., Alarabi, L., Bao, J., Eldawy, A., Magdy, A., Sarwat, M., Waytas, E., Yackel, S.: MNTG: an extensible web-based traffic generator. In: *International Symposium on Spatial and Temporal Databases*, Springer, pp. 38–55 (2013)
- Yu, J., Zhang, Z., Sarwat, M.: Spatial Data Management in Apache Spark: The GeoSpark Perspective and Beyond. *GeoInformatica* (2018)
- Eldawy, A., Alarabi, L., Mokbel, M.F.: Spatial partitioning techniques in SpatialHadoop. *Proc. Int. Conf. Very Large Data Bases* **8**(12), 1602 (2015)

21. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: Cluster Computing with Working Sets. In: USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010 (2010)
22. Kesting, A., Treiber, M., Helbing, D.: Enhanced intelligent driver model to access the impact of driving strategies on traffic capacity. *Philos. Trans. R. Soc. Lond A* **368**(1928), 4585 (2010)
23. Kesting, A., Treiber, M., Helbing, D.: General lane-changing model MOBIL for car-following models. *Transp. Res. Rec.* **1999**(1), 86 (2007)
24. Karich, P., Schröder, S.: Graphhopper. <http://www.graphhopper.com>, Last accessed 4(2), 15 (2014)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Jia Yu¹  · Zishan Fu¹ · Mohamed Sarwat¹

Zishan Fu
zishanfu@asu.edu

Mohamed Sarwat
msarwat@asu.edu

¹ Arizona State University, Tempe, AZ, USA