Distributed Agent-Based Traffic Simulations

Matthieu Mastio, Mahdi Zargayouna, Gérard Scemama, Omer Rana

Distributed agent-based traffic simulations

Matthieu Mastio[a] Mahdi Zargayouna[a] Gérard Scemama[a] Omer Rana[b]
[a] Université Paris Est, IFSTTAR, GRETTIA
14-20, boulevard Newton,
77447 Champs sur Marne, France.
[b] School of Computer Science and Informatics, Cardiff University,
Cardiff, United Kingdom
{matthieu.mastio, hamza-mahdi.zargayouna, gerard.scemama@ifsttar.fr}@ifsttar.fr
ranaof@cardiff.ac.uk

***Abstract*—Modeling and simulation play an important role in transportation networks analysis. With the widespread of personalized real-time information sources, it is relevant for the simulation model to be individual-centered. The agent-based simulation is the most promising paradigm in this context. However, representing the movements of realistic numbers of travelers within reasonable execution times requires significant computational resources. It also requires relevant methods, architectures and algorithms that respect the characteristics of transportation networks. In this paper, we tackle the problem of using high-performance computing for agent-based traffic simulations. To do so, we define two generic agent-based simulation models, representing the existing sequential agent-based traffic simulations. The first model is macroscopic, in which travelers do not interact directly and use a fundamental diagram of traffic flow to continuously compute their speeds. The second model is microscopic, in which travelers interact with their neighbors to adapt their speeds to their surrounding environment. We define patterns to distribute these simulations in a high-performance environment. The first distributes agents equally between available computation units. The second pattern splits the environment over the different units. We finally propose a diffusive method to dynamically balance the load between units during execution. The results show that agent-based distribution is more efficient with macroscopic simulations, with a speedup of 6 compared to the sequential version, while environment-based distribution is more efficient with microscopic simulations, with a speedup of 14. Our diffusive load-balancing algorithm improves further the performance of the environment based approach by 150%.**

I. Introduction

Mobility policies makers need decision support systems to decide which transportation policies they should implement. In this context, simulation is one of the important tools to test strategies and multiple scenarios without impacting the real traffic [1], [18]. However, transport systems are becoming progressively more complex since they are increasingly composed of connected entities (mobile devices, connected vehicles, etc). It becomes critical that simulation tools take into account this fact. Indeed, with the generalization of real-time traveler information, the behavior of modern transport networks becomes harder to analyze and to predict.

For these reasons, agent-based simulation, which adopts an individual-centered approach, is one of the most relevant paradigms to design and implement such applications. The development of agent-based traffic simulations is relevant in several contexts and in pursuit of various objectives. The simulation can be used to validate the impact of the use of cooperative systems [16], [19], to test changes in behavior after the introduction of new mobility services, such as carpooling, etc. An agent-based traffic simulation platform simulates the behavior of travelers interacting in a complex, dynamic and open environment, on which they have a partial perception [3]. Each agent tries to find the most efficient route to reach his destination in a network evolving dynamically. In some applications (e.g. [33]), an agent can potentially be informed of the status of the network and use this information to modify his route. In this kind of simulations, it is important to model and simulate a realistic number of travelers to correctly observe the effects of individual decisions. In the European project *Instant Mobility*[1] for instance, the objective was to supply a multimodal platform with individual and multimodal travel queries and dynamic positions of travelers and vehicles. To allow the platform to demonstrate its efficiency in an operational context, we implemented a simulator (called SM4T [34]), which had to be executed with an actual volume of travelers. Other examples where simulations must be scalable concern testing of new mobility services such as carpooling, car sharing, dial a ride, evacuation modeling, the exchange of information between connected vehicles, etc.

However, the simulation of an actual number of passengers in a big city (several millions of travelers) requires both considerable computing power and an architecture allowing the distribution of computations on many hosts. The majority of current agent-based traffic simulators do not allow such distribution. This induces limitations on

[1] http://www.instant-mobility.com/

the number of simulated travelers, means of transport and the size of the considered networks. provide reproducible generic distribution patterns that could be used by existing and future implementations of agent-based traffic simulations.

In this paper, we propose to study distribution methods for agent-based traffic simulations. We define two generic agent-based simulation models, representing the main types of agent-based simulations of the literature. The first model is called macroscopic, in the sense that travelers do not interact directly but use a fundamental diagram of traffic flow to continuously compute their speeds. This is the choice performed for instance in these works [34], [22], [8]. The second model is called microscopic, in which travelers interact with their neighbors to adapt their speeds to their surrounding environment. This is the most common choice performed in the literature for agent-based simulations (e.g. [5], [24]). This paper studies two distribution patterns (agent-based and environment-based) applied to these two simulation models. The results show that agent-based distribution is more efficient with macroscopic simulations while environment-based distribution is more efficient with microscopic simulations. We propose a load-balancing mechanism for the environment-based distribution and show that, with the right parameters definition, it has a positive impact on the distribution platform.

The remainder of this paper is structured as follows. In section II, we present the previous studies for agent-based traffic simulation and the existing distributed agent-based platforms. Section III presents a generic simulator for the execution of both macroscopic and microscopic agent-based traffic simulations. Section IV presents the two distribution patterns (agent-based and environment-based) and their application to the two simulation models. Section V presents the load-balancing mechanism. Section VI explains our experimental setup and the results of our simulations. Section VII concludes this paper.

II. Related work

In this section, we position our work with the previous works in the literature. In the next paragraph, we present the existent agent-based traffic simulators. Then we will focus on the existing proposals for distributing these platforms. We will finally describe the generic parallel multiagent platforms.

There exists several agent-based traffic simulations in the literature. Most of them are microscopic, in the sense that they rely on local interactions between traveler agents to define agents speeds. For instance, Transims [26], MATSim [24], Sumo [5] and Vissim [14] are widely used microscopic simulators of this type. Archisim [13] are also agent-based traffic simulation platforms describing precisely the behaviors of each traveler at a microscopic scale. Some existing agent-based simulations are "macroscopic", in the sense that they compute the agents speeds based on a function mapping the number of agents traveling on an edge with their speeds. This model is generally used when an individual representation of travelers is needed but there are no reliable data about their local behavior. The authors in [34], [22], [8] have made this choice.

The problem of distributing agent-based traffic simulations have attracted a lot of researches recently. Some previous works have addressed the specific problem of distributing agent-based traffic simulations. In [6], the authors propose dSumo, a distribution platform applied to the Sumo microscopic simulation platform. In [21], the authors propose a parallel version of Paramics [9]. However, they do not implement a load-balancing mechanism and present small clusters and networks (grid-like). The authors in [27] present a parallelization of the Transims platform, with a load-balancing mechanism. They use a master-slave model for synchronizing the different hosts. In the present work, our objective is to propose completely distributed mechanisms, independent from specific traffic platforms. In addition, to the best of our knowledge, our work is the only one dealing with the distribution of simulations using the two different interaction models (viz. microscopic and macroscopic). More theoretical works studied general methods to address the traffic simulation distribution problem. In [23] and [30], the authors propose to relax some synchronization constraints to achieve a better scalability by reducing the time the hosts wait for each other. This relaxation of constraint implies a loss a precision which is not viable in the case of a traffic simulator. Indeed, we could reach a state where the vehicles overlap and occupy the same position in the network. Our objective is to have a perfect accuracy of distributed traffic simulation, where the result in terms of traffic is exactly the same between a sequential and a distributed simulation.

Some general-purpose multiagent platforms have been specifically developed for large scale simulation in the last years. RepastHPC [12], a distributed version of Repast Symphony, uses the Repast's concepts of projections and contexts and adapts them for distributed environments. Pandora [2] is close to RepastHPC and automatically generates the code required for inter-server communications. GridABM [17] is based on Repast Symphony but takes another approach and proposes to the programmer general templates to be adapted to the communication topology of his simulation. Flame [11] allows the programmer to generate HPC simulations from finite state machines. It has also been sugested to use graphical units (GPGPU) to scale up the multiagents simulations. As we have seen in other articles, the TurtleKit 3 platform has been used in GPGPU [25]. However, these distributed platforms do not offer fine controls on how the communications between hosts are performed. Indeed, the communication layer is transparent for the programmer, which makes it easier for him to implement distributed simulations, but prevents him from optimizing the distribution. The best way to manage the communications depends on the application and using such general platforms for a traffic simulator would not produce optimal results. In [28] the authors

discuss the issues related to multiagent simulation in a distributed virtual environment. The authors describe methods to split the virtual environment in several zones to parallelize the simulation execution. This work proposes an efficient splitting of a continuous space in two dimensions. In the present paper, we use an adaptation of this work for a graph structure to distribute traffic-based simulations.

III. Generic agent-based traffic simulations

In the following, we present two generic agent-based traffic simulations. They are designed with the objective of representing the existing agent-based traffic simulations. They contain the main features of these types of simulation, namely the network representation and the agents movements on this network. The next section presents the common components of these simulations. The next sections present the differences between the two simulations, mainly concerning the travelers speed computation.

A. The multiagent system

In our proposals, we consider agents that are virtual autonomous entity, evolving in an environment, and taking actions to realize their objectives [7]. Agents are able to act on their environment, and to interact with the others agents.

A common base is shared by both simulations, which is composed of a dynamic set of agents representing travelers, interacting with a transportation network environment. We model the transportation network in which the travelers evolve with a graph $G(V, E)$, where $E = \{e_1, ..., e_n\}$ is a set of edges representing the roads and $V = \{v_1, ..., v_n\}$ is a set of vertices representing the intersections. The agents, representing the travelers, move on this network from their origins to their destinations, trying to minimize their travel costs. Fig. 1 describes the steps followed by a traffic simulation. First, the simulation platform loads the parameters (simulation duration, number of generated agents, etc.) and the description of the network. Then, it creates the logical graph from the network representation, to enable shortest paths calculation and agents movements. The scheduler, which is responsible of agents execution, ranges over the agents and asks them to execute one step of simulation (either to compute a shortest path or to move from one position to another). When an agent reaches his destination, he comes back to his origin point (to keep a constant number of agent in the simulation). When all agents have executed their step instructions for one tick of simulation time, the scheduler increments the simulation tick counter (*step++*), and the process starts again. When the simulation duration is reached, the simulation stops and the results are collected.

The agents execute a step method each time they are activated by the scheduler. When created, an agent has an origin node o and a destination node d. The first action that he executes when created and activated by the scheduler is to compute an $A^\star$ shortest path algorithm between o and d. The shortest path is performed on the

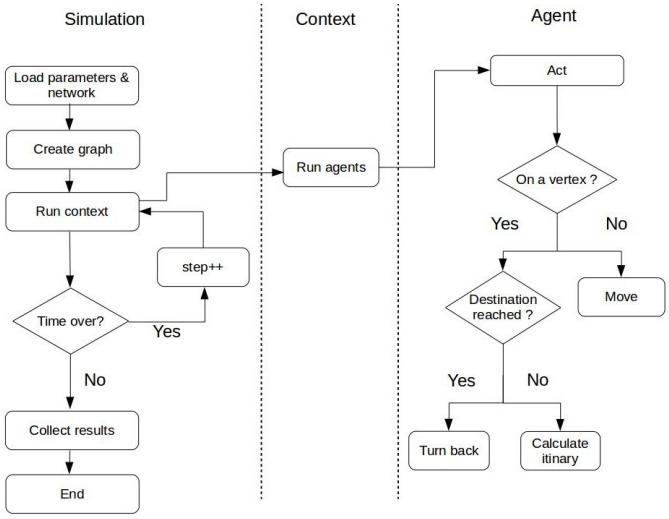

Fig. 1: Steps of a simulation

graph G, which edges costs are dynamic, depending on the current traffic. When he has a current path, the agent moves according to it. At each tick, he moves the allowed distance following his current speed. The speed of the agent is computed following the simulation model (microscopic or macroscopic), described in the following sections. Each time he reaches a node, the agent recomputes a shortest path, to check if the current traffic conditions have evolved and if a new shortest path has become available[2].

These are the main components of the model that are common to both types of simulations. In the following sections, we present the specific methods for the two simulations, namely macroscopic and microscopic.

B. Macroscopic simulation model

In the macroscopic simulation model, the speed of an agent on an edge is computed following the number of other agents traveling on the same edge. To this end, a fundamental diagram of traffic flow is used. The diagram defines a relation between the flow (vehicles/hour) and the density (vehicles/km) (cf. Fig. 2, left) on an edge or a part of an edge to calculate the speed of the agents at each time. The fundamental diagram suggests that if we exceed a critical density of vehicles k_c, the more vehicles are on a road, the slower they will move.

With the distribution objective that we have, the locations of the agents and their interaction patterns are the most important. In the macroscopic model, the agents do not interact directly. The speed of the agent is computed with an interaction between the agent and the edge. The latter knows the number of agents currently using it, and

[2]The graph being directed, turnarounds are only possible at nodes and there is no need for the agent to execute a shortest path while traveling on an edge.

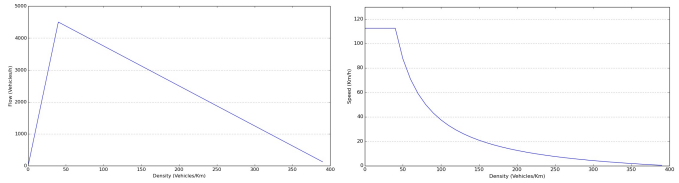

Fig. 2: Fundamental diagram (left) Speed in function of density (right)

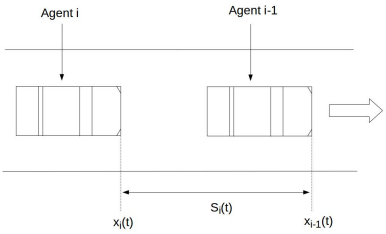

Fig. 3: The car-following model

based on the speed function providing the right speed to be used by the agent, based on the fundamental diagram (cf. Fig. 2, right).

C. Microscopic simulation

In the microscopic simulation model, the speed of an agent on an edge is computed following the position and speed of the vehicles surrounding him. In this model, the information available to each agent is only local. The agents perceive a part of their environment, delimited by their aoi[3] and then calculate their next move given the perceived information. This implies many local communications between the agents, because their actions will be conditioned by the actions of the other agents present in their aoi. This model is generally based on:

- a car-following model
- a lane-changing model
- and/or a gap acceptance model

All three models focus on local interactions. In the following, we describe an example of car-following model.

At each time tick of the simulation, each agent computes his speed based on the speed and position of the agent before him. The variables needed to describe our model are the following:

1) $x_n(t)$ — the position of n at time t
2) $x'_n(t)$ — the speed of n at time t
3) $x''_n(t)$ — the speedup of n at time t
4) $s_n(t) = x_{n-1}(t) - x_n(t)$ — the inter-agent distance
5) $s'_n(t) = x'_{n-1}(t) - x'_n(t)$ — the relative speed
6) T — the reaction time

Thus, at any time step, the speed of an agent is given by the relation:

$x''_n(t+T) = \alpha s'_n(t) + \beta s_n(t) + \gamma x'_n(t)$

If there is no vehicle preceding the agent, he will accelerate until he reaches the speed limit of his edge.

In this model, each agent registers his experienced travel time when he reaches the end of the edge (as proposed by the authors in [31]). The shortest path calculation is based on the graph where the travel times costs are fed by the agents following this procedure.

In contrast with the macroscopic model, the agents in the microscopic model do interact directly. The speed of the agent is computed with a direct interaction between the considered agent and the agents before him. This difference between the two models conditions the choice of the relevant distribution pattern for the considered simulation type. The distribution patterns are described in the following section.

[3]Area of interest

IV. Distribution

We define two patterns to distribute traffic simulations. The patterns are the same than those identified by the authors in [29] for general-purpose situated multiagent simulations, and we believe that they present two representative distribution patterns for this kind of simulations. The first pattern (called agent-based distribution) is the distribution model used by [4]. It consists in the duplication of the transport environment on all processing units, and the equal dispatching of the agents on each one. As a consequence, agents stay on the same unit during all the simulation. The second pattern (called environment-based distribution) is the mostly used pattern in the literature. It consists in partitioning the transportation environment and the dispatching of each subpart of the environment - and all the agents in it - on each processing unit. In this pattern, agents might have to move from one unit to another if their itinerary crosses several subparts of the transport environment.

A. Agent-based distribution

The first distribution pattern is agent-based, since it clusters the set of agents in k equal parts (with k the number of available processing units), and distributes each subset on a unit and executes the simulation (cf. Fig. 4). The transportation network is duplicated on each unit. This method ensures that each unit has the same amount of work at any time of the simulation. In the following, we describe the use of this pattern for both simulation models that we have defined.

1) Macroscopic simulation with agent-based distribution: In a macroscopic simulation, when it is distributed following the agent-based distribution pattern, every units continuously (at each simulation tick) informs the other units of its network state. This is due to the fact that they do not have a complete view of the network state, since

only a part of the agents evolves in the unit. Thus, they send the list of edges together with the number of agents currently on them. Each unit is then able to compute the shortest paths and the relevant speed of the agents (using the fundamental diagram of traffic).

2) Microscopic simulation with agent-based distribution: When distributed following the agent-based distribution pattern, the agents in a microscopic simulation do not use a fundamental diagram of traffic to compute their speeds. Instead, they need to know the state of the agents preceding them. To do so, they interrogate the edges in the other units to know if there are agents preceding them, and if it is the case, to know their states. Moreover, the units exchange the current travel times (provided by the agents as explained in the microscopic model), in order to compute the shortest paths of the agents.

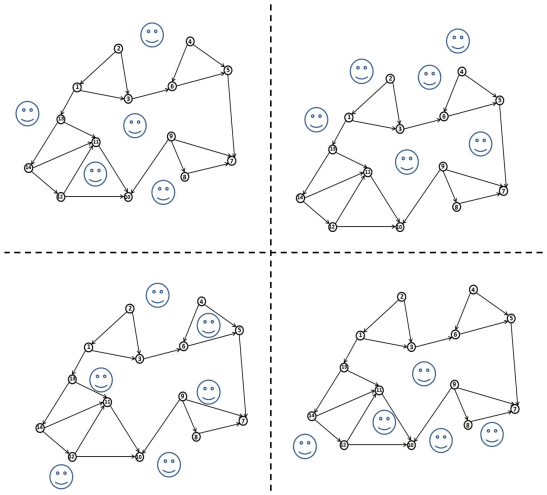

Fig. 4: Agents distribution

B. Environment-based distribution

The second approach to distribute traffic simulations is environment-based. It tries to keep on the same unit the agents who are geographically close in the transport network (cf. Fig. 5). To this end, the network is splitted in k parts (with k the number of available processing units), and distributed on the different units. Each unit is only aware of what is happening on the part of the graph that it is managing, and the agents that are in the same area are now likely to be on the same unit. If an agent reaches a part of the network that is not managed by his current unit, he moves to the proper unit. In order for the environment distribution method to be effective, each unit has to manage approximately the same number of agents and the number of edges connecting the partitions has also to be minimized (to reduce the number of agents being transferred between units).

The problem of graph partitioning has been widely studied in the scientific literature. We propose a method

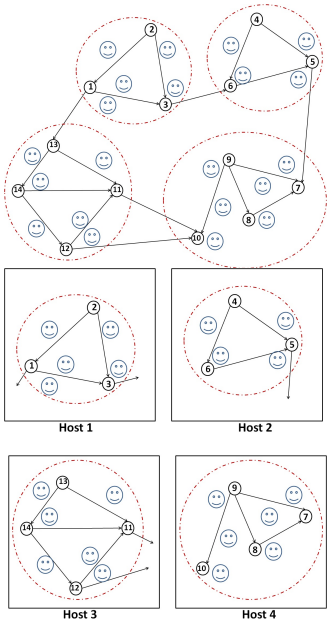

Fig. 5: Environment distribution

derived from the Differential Greedy algorithm [15] that allows us to use the algorithm with weighted vertices while producing more connected partitions (Algorithm 1). For edges partitioning, we make the same choice as [10] by not cutting edges in the middle. We associate each edge with the partition of its origin node.

Algorithm 1 Differential Greedy algorithm

Require: Graph $G = (V, E)$, number k of partition
Ensure: Partition P
(1) $P \leftarrow P_0, ..., P_{k-1}$
(2) $V' \leftarrow V$
for $p \in [0, k-1]$ **do**
  (3) $v \leftarrow$ random vertex of V'
  (4) $P_p \leftarrow \{v\}$
  (5) $V' \leftarrow V' \setminus \{v\}$
end for
while $|V'| > 0$ **do**
  (6) $p \leftarrow$ index of the lightest partition
  (7) $m = min_{v \in V'}(1+\epsilon)$(number of v's neighbors $\in P_p$) − (number of v's neighbors $\notin P_p$)
  (8) $mv =$ random vertex of $v \in V' | (1+\epsilon)$(number v's neighbors $\in P_p$) − (number of v's neighbors$\notin P_p$) $= m$
  (9) $P_p \leftarrow P_p \cup \{mv\}$
  (10) $V' \leftarrow V' \setminus \{mv\}$
end while
(11) **Return** P

The algorithm starts by creating a minimal partition with only one node each (instructions (1) to (5)). Then, while there are nodes to be associated to partitions, the

algorithm:

- chooses the lightest partition P_p, in terms of agents present in it (instruction (6))
- finds the nodes that are the most connected with the nodes already in P_p and that are the least connected with the nodes that are not in P_p. The parameter ϵ gives more or less importance to the nodes that are close to the partition (instruction 7).
- chooses one of these nodes, adds it to the partition and removes it from the nodes to process (instructions 8 to 10).

This algorithm is fast and intuitive. Our modification of the original differential greedy algorithm concerns the choice of the current partition to treat. The "lightest" partition in the original algorithm concerns the number of nodes in the partition, while in our algorithm, it concerns the number of agents in the partition.

1) Macroscopic simulation with environment-based distribution: When used with an environment-based distribution, the computation units in the macroscopic simulation exchange the current travel times on the transport edges, to be able to compute the shortest paths for the agents. However, since all the agents on an edge are present on the same unit, they do not need to exchange the number of agents per edge. The fundamental diagram of traffic and the speeds of the agents can indeed be defined locally.

2) Microscopic simulation with environment-based distribution: When distributed following the environment-based distribution pattern, the agents in a microscopic simulation need to know the state of the agents preceding them. In contrast with the agent-based distribution model, the agents preceding them are by definition present on the same computation unit. The interrogation of the edges is then local to the concerned computation unit. The units keep on exchanging the current travel times (provided by the agents) to compute the shortest paths for the agents.

V. Diffusive load balancing

With the environment-based distribution, the graph partitioning is executed once, based on the initial positions of the agents and the network structure. However, if the network structure is stable, agents positions are of course changing over the simulation, which could lead to load imbalance during the simulation. Typically, travelers drive from their residential areas to work areas in the morning and drive back home in the evening. Certain parts of the network, and consequently their corresponding computation units, would have many more agents to handle than the others, and the whole simulation would slow down. Indeed, at the end of each time step of the simulation, all the units have to wait for each others to synchronize. The overall execution time of a simulation step is then equal to the execution time of the slowest unit. As the execution time of a given unit is directly linked to the number of agents executing on this unit, it is important in these conditions to keep the load balanced.

A. The algorithm

A straightforward way to balance the load dynamically would be to part the graph from scratch when one unit is overloaded. But in the traffic simulations we are targeting, we have to deal with big graphs and many agents: the time needed to part the graph and to move all the agents from one unit to another would be counterproductive and would slow down the simulation.

Algorithm 2 Diffusive Load Balancing algorithm

Require: P partition of a graph $G = (V, E)$
Require: P_i current partition
Require: n total number of agents
Require: k number of processing units
$threshold \leftarrow \alpha(n/k)$
if number of agents in $P_i > threshold$ **then**
  $P_{min} \leftarrow$ partition connected to P_i with the minimum load
  $v_{max} \leftarrow$ the heaviest vertex $\in P_i$ connected to P_{min} **with** $|v| < 0.5(n/k)$
  move v_{max} to P_{min}
end if

That is why we have developed a dynamic load balancing algorithm, able to diffuse incrementally the excessive workload of a unit on the units around. At the beginning of the simulation, we use the modified differential greedy algorithm to part the graph. Then, during the simulation, each unit maintains a list of boundary vertices of the traffic network. These vertices are the ones who have a common edge with a vertex managed by another unit. When the load of a processing unit (in terms of number of agents) exceeds a threshold, we trigger the load balancing mechanism. The unit will request the load of the processing units around, and will transfer its most heavy vertex (in terms of number of agents on it) to his least loaded neighbor (cf. algorithm 2). However, when there is a huge number of agents on a vertex, the latter will continually be sent between the units. To avoid this perpetual oscillation, we define a limit on the number of agents from which the vertex will not be moved. This algorithm avoids to part the graph from scratch, and allows a good load-balancing with a linear complexity: $O(n) + O(k)$.

For instance, in Fig. 6, the W values represent the number of agents in each partition. The partition 1 is initially overloaded (a), compared to the others. The partition 2, which is the neighboring partition with the smallest load is selected for the transfer. At this point, both vertices 7 and 9 are candidates to be transferred, as they are in the boundary between the partition 1 and 2. The heaviest vertex (9) is selected, and sent to partition 2. This gives us a new graph partition, by load diffusion.

The choice of the coefficient α is crucial here, because it will determine how often the mechanism will be triggered. Indeed, the closer α is of 1, the most often the procedure of load balancing will be triggered. Triggering it too often would leads to unstable partitions.

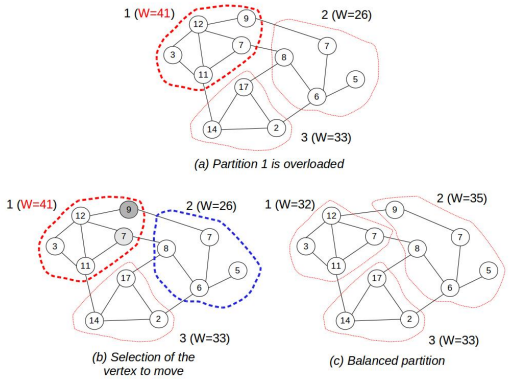

Fig. 6: Diffusive load balancing

VI. Experiments and Results

A. Implementation

A way to execute a distributed simulation is to define a distributed program where each computation unit, while executing the same program, owns only a part of the program data in its private memory, and all the processors are connected by a network. The advantage of this approach is its high scalability. Indeed, it can be implemented on most parallel architectures and we can deploy the same simulation on larger systems if we need more computing power and memory. We use Python to develop our simulator, for its efficiency in quick prototyping. Python is a mature portable language with a lot of well tested scientific libraries and is along with C and Fortran one of the most used languages for high performance computing [20]. Here, we do not seek absolute performance, but we aim to study the relative efficiency of different distribution methods. Thus we believe that Python is a relevant choice. The interprocess communications are managed by MPI, which is the standard language for parallel computing with a huge community of users. MPI offers a simple communication model between the different processes in a program and has many efficient implementations that run on a variety of machines[4].

We have executed the distributed simulations on an experimental cluster that we have set up. For our tests, we used two hosts under Linux Mint 17.2 Rafaela (kernel version 3.16.0-38-generic) each with an Intel Xeon processor CPU E7-4820 (32 cores at 2Ghz) with 250GB of memory. We ran the simulations on two configurations: the first is a sequential version of the program on a single core, the second is a distributed version on the whole 64 cores.

The considered network is a real network concerning the Paris-Saclay region, France, with 1895 nodes and 3831 edges. The number of travelers using this network is around 110,000. We consider from 10,000 travelers to 500,000 travelers in our simulations. That means that we represent from around 10% to around 500% of the real number of travelers in our simulations.

[4]MPI4PY is an efficient interface that allows to use MPI with Python.

B. Results

1) Macroscopic Vs. Microscopic model: We compare the two methods of distribution (agent-based and environment-based distributions) with the different paradigms (micro and macro) increasing the number of agents (from 10,000 to 500,000).

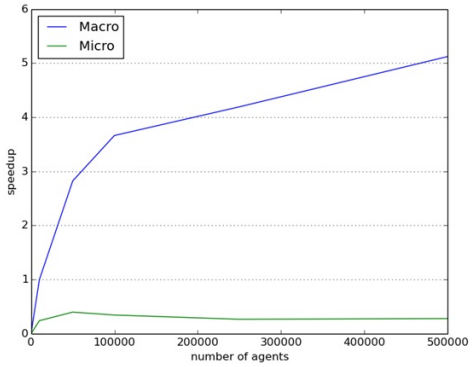

Fig. 7: Speedup for the agent-based distribution

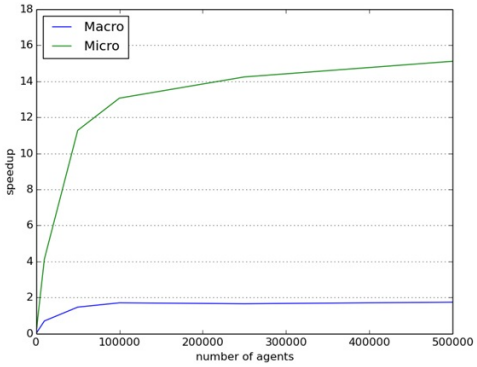

Fig. 8: Speedup for the environment-based distribution

The speedups for the two distributions methods applied on the different paradigms are plotted in Fig. 8 and Fig. 7. The speedup measures how many times the distributed simulation is faster compared to the corresponding sequential execution.

As we can see, the agent distribution is efficient for a macroscopic model (more than 5 times faster with 500,000 agents). There is no local interactions in this type of simulations. This method allows to get a perfectly balanced

load all along the simulation, while keeping the amount of inter-servers communications at the minimum.

However, this method is particularly ineffective in the case of a microscopic simulation. Indeed, the agents will now interact a lot with other agents that are not situated in the same unit. This will generate a lot of communications between the servers, and the gain of the parallelization is annihilated by the time required by these communications. This method is even less efficient than the sequential execution for the microscopic model (speedup < 1).

For a macroscopic simulation, the environment distribution is less efficient than agent distribution. It is well adapted for a microscopic simulation though. This method is up to 15 times faster than a sequential execution applied in a microscopic simulation.

2) Impact of load-balancing: For the assessment of the load-balancing mechanism, we have to define the optimal value of α for the experiments. To do so, we execute three different types of simulations, each applied to the microscopic simulation type: the first, called "static" is the environment-based distribution approach presented earlier. The second is the load-balancing approach with $\alpha = 1.2$ (called "dynamic_1_2") and the third is the load-balancing approach with $\alpha = 1.3$ (called "dynamic_1_3")[5]. Fig. 9 shows the results. Each point in the curves represents the difference between the optimal load (equal agents distribution between units) and the load on the most loaded process, for each time step. As we can see, with the "static" approach, the difference (the imbalance) is big. With the dynamic approach and $\alpha = 1.2$, the balance is better than with the static approach, but the load is unstable. Finally, with $\alpha = 1.3$ the oscillations ceases and the load of the simulation is successfully balanced between the processes. Based on a series of experiments that we have executed, choosing a bigger α would lead to more imbalanced partitions, so we choose to keep $\alpha = 1.3$ for the rest of our experiments.

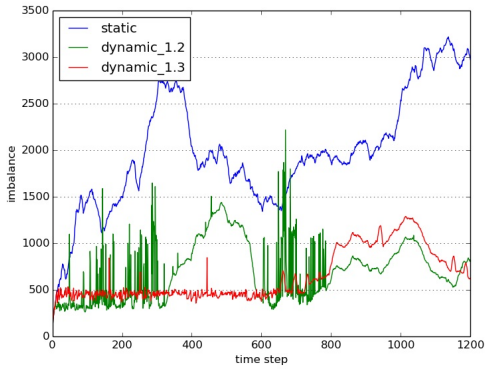

Fig. 9: Load imbalance

[5]We do not display the curve for $\alpha = 1.1$ because it was extremely unstable

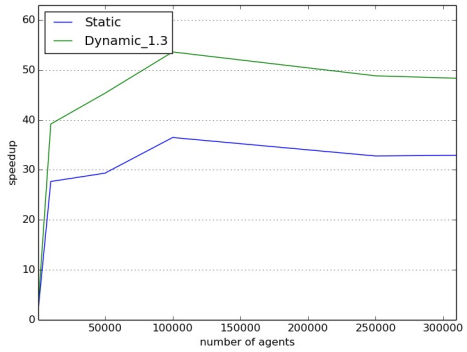

Fig. 10: Speedup for the different methods

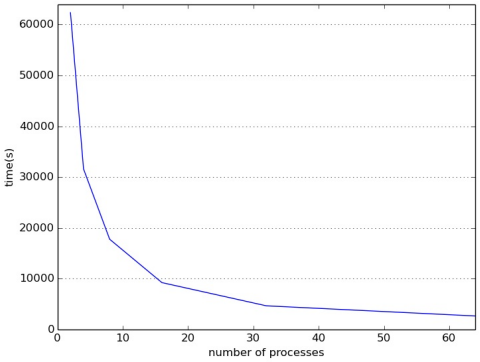

Fig. 11: Execution time in function of the number of processes

Table I indicates the execution times for a simulation of 1,000 time steps with the sequential method and the two distribution methods (static and dynamic with $\alpha = 1.3$). Fig. 10 shows the speedups of the two methods in comparison to the sequential execution.

Finally, Fig. 11 exhibits the efficiency of the dynamic load balancing in function of the number of used processes. The simulation we ran here was for 100,000 agents and 1000 time steps, with 64 processes. We can see that with our load balancing, the simulation scale very well with the number of process we have at our disposal. For 100000 (which is approximately the load we can expect on the Paris-Saclay network) we reach a speedup of 54 with 64 processes.

VII. Conclusions and perspectives

In this paper, we applied two distribution methods on two types of agent-based traffic simulators. We have seen that agent-based distribution is well suited for macroscopic simulators while environment-based distribution is well suited for microscopic simulations. These findings

number of agents	10,000	50,000	100,000	250,000	500,000
Sequential (1 proc)	12814	62672	142350	315876	631243
Static (64 cores)	463	2136	3902	9636	18929
Dynamic_1.3 (64 cores)	327	1382	2665	6468	13480

TABLE I: Load balancing: computational times (in seconds)

are useful for the distribution of the existing agent-based traffic simulations. Microscopic simulations can be more optimally distributed using dynamic load-balancing mechanisms, such as the diffusive load-balancing method presented in this paper.

The proposed diffusive load balancing algorithm that we have proposed is able to dynamically balance the loads of a traffic simulation, and is efficient with our experimental setup. In a future work, we will test this method on a cloud-like environment (single core units, linked by a network). We will also investigate a hybrid approach: when a process has to manage a very loaded vertex, being able to distribute the agents on this vertex between two or more units would allow us to improve furthermore the performance of the simulation.

We also plan to consider multimodal agent-based traffic simulators. The presence of different transport modes and networks could encourage to mix the patterns presented in this paper with a distribution per transport mode. We are also working on the integration of information networks (such as social networks or intervehicular interaction [32]) and their impact on the distribution performance. Indeed, if travelers interact often, they should be preferably executed on the same units, or else they will generate too many communication and deteriorate the performance of the system.

References

[1] A. Abadi, T. Rajabioun, and P. A. Ioannou. Traffic flow prediction for road transportation networks with limited traffic data. *IEEE Transactions on Intelligent Transportation Systems*, 16(2):653–662, 2015.

[2] E. Angelotti, E. Scalabrin, and B. Avila. PANDORA: a multi-agent system using paraconsistent logic. In *Fourth International Conference on Computational Intelligence and Multimedia Applications, 2001. ICCIMA 2001. Proceedings*, pages 352–356, 2001.

[3] F. Badeig, F. Balbo, G. Scemama, and M. Zargayouna. Agent-based coordination model for designing transportation applications. In *Intelligent Transportation Systems, 2008. ITSC 2008. 11th International IEEE Conference on*, pages 402–407. IEEE, 2008.

[4] J. Barceló, J. Ferrer, D. García, R. Grau, M. Forian, I. Chabini, and E. Le Saux. Microscopic traffic simulation for att systems analysis. a parallel computing version. *Contribution to the 25th Aniversary of CRT*, 1998.

[5] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. SUMO - simulation of urban MObility - an overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, pages 55–60, 2011.

[6] Q. Bragard, A. Ventresque, and L. Murphy. Self-balancing decentralized distributed platform for urban traffic simulation. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–8, 2016.

[7] M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational intelligence*, 4(3):349–355, 1988.

[8] R. Cajias, A. Gonzalez-Pardo, and D. Camacho. A multi-agent traffic simulation framework for evaluating the impact of traffic lights. In *ICAART (2)*, pages 443–446, 2011.

[9] G. D. Cameron and G. I. Duncan. Paramics parallel microscopic simulation of road traffic. *The Journal of Supercomputing*, 10(1):25–53, 1996.

[10] N. Cetin, A. Burri, and K. Nagel. A large-scale agent-based traffic microsimulation based on queue model. In *Proceedings of Swiss Transport Research Conference (STRC), Monte Verita, CH*, pages 3–4272, 2003.

[11] S. Coakley, M. Gheorghe, M. Holcombe, S. Chin, D. Worth, and C. Greenough. Exploitation of high performance computing in the FLAME agent-based simulation framework. In *2012 IEEE HPCC-ICESS*, pages 538–545, 2012.

[12] N. Collier and M. North. Repast HPC: A platform for large-scale agent-based modeling. In W. Dubitzky, K. Kurowski, and B. Schott, editors, *Large-Scale Computing*, pages 81–109. John Wiley & Sons, Inc., 2011.

[13] A. Doniec, R. Mandiau, S. Piechowiak, and S. Espié. A behavioral multi-agent model for road traffic simulation. *Eng. Appl. of AI*, 21(8):1443–1454, 2008.

[14] M. Fellendorf and P. Vortisch. Microscopic traffic flow simulator vissim. In *Fundamentals of Traffic Simulation*, pages 63–93. Springer, 2010.

[15] C. Fiduccia and R. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Conference on Design Automation, 1982*, pages 175–181, June 1982.

[16] M. Gueriau, R. Billot, N.-E. El Faouzi, S. Hassas, and F. Armetta. Multi-Agent Dynamic Coupling for Cooperative Vehicles Modeling. In *The 29th Conference on Artificial Intelligence AAAI'2015*, Jan. 2015.

[17] L. Gulyas, G. Szemes, G. Kampis, and W. de Back. A modeler-friendly API for ABM partitioning. In *ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 219–226, 2009.

[18] T.-Y. Hu, C.-C. Tong, T.-Y. Liao, and W.-M. Ho. Simulation-assignment-based travel time prediction model for traffic corridors. *IEEE Transactions on Intelligent Transportation Systems*, 13(3):1277–1286, 2012.

[19] M. A. S. Kamal, J.-i. Imura, T. Hayakawa, A. Ohata, and K. Aihara. A vehicle-intersection coordination scheme for smooth flows of traffic without using traffic lights. *IEEE Transactions on Intelligent Transportation Systems*, 16(3):1136–1147, 2015.

[20] H. P. Langtangen and X. Cai. On the efficiency of python for high-performance computing. In *Modeling, Simulation and*

Optimization of Complex Processes, pages 337–357. Springer Berlin Heidelberg, Jan. 2008.

[21] D.-H. Lee and P. Chandrasekar. A framework for parallel traffic simulation using multiple instancing of a simulation program. *ITS Journal*, 7(3-4):279–294, 2002.

[22] D. Meignan, O. Simonin, and A. Koukam. Simulation and evaluation of urban bus-networks using a multiagent approach. *Simulation Modelling Practice and Theory*, 15(6):659 – 671, 2007.

[23] D. Mengistu and M. v. Lowis. An algorithm for optimistic distributed simulations. In *Modelling, Simulation, and Identification / 658: Power and Energy Systems / 660, 661, 662*. ACTA Press, 2011.

[24] M. Michal and N. Kai. Towards multi-agent simulation of the dynamic vehicle routing problem in matsim. In *Proceedings of the 9th International Conference on Parallel Processing and Applied Mathematics - Volume Part II*, PPAM'11, pages 551–560, Berlin, Heidelberg, 2012. Springer-Verlag.

[25] F. Michel. Délégation GPU des perceptions agents : intégration itérative et modulaire du GPGPU dans les simulations multi-agents. application sur la plate-forme turtlekit 3. *Revue d'Intelligence Artificielle*, 28(4):485–510, 2014.

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

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

[28] O. Rihawi, Y. Secq, and P. Mathieu. Effective distribution of large scale situated agent-based simulations. In *ICAART 2014 6th International Conference on Agents and Artificial Intelligence*, volume 1, pages 312–319. SCITEPRESS Digital Library, 2014.

[29] O. Rihawi, Y. Secq, and P. Mathieu. Effective distribution of large scale situated agent-based simulations. In *ICAART 2014 - Proceedings of the 6th International Conference on Agents and Artificial Intelligence, Volume 1, ESEO, Angers, Loire Valley, France, 6-8 March, 2014*, pages 312–319, 2014.

[30] M. Scheutz and P. Schermerhorn. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *Journal of Parallel and Distributed Computing*, 66(8):1037–1051, 2006.

[31] J. Wahle, A. L. C. Bazzan, F. Klügl, and M. Schreckenberg. The impact of real-time information in a two-route scenario using agent-based simulation. *Transportation Research Part C: Emerging Technologies*, 10(5):399–417, 2002.

[32] M. Zargayouna, F. Balbo, and K. Ndiaye. Generic model for resource allocation in transportation. application to urban parking management. *Transportation Research Part C Emerging Technologies*, 71:538–554, 2016.

[33] M. Zargayouna, B. Zeddini, G. Scemama, and A. Othman. Agent-based simulator for travelers multimodal mobility. *Frontiers in Artificial Intelligence and Applications*, 252:pp 81–90, Jan. 2013.

[34] M. Zargayouna, B. Zeddini, G. Scemama, and A. Othman. Simulating the impact of future internet on multimodal mobility. In *The 11th ACS/IEEE International Conference on Computer Systems and Applications AICCSA'2014*. IEEE Computer Society, 2014.

Matthieu Mastio received his M.Sc degree in computer science from the University of Lille (France) in 2011 and his Ph.D degree in computer science from the University of Paris-Est (France) in 2017. He is mainly interested in multiagent simulation, traffic modeling and high-performance computing.

Mahdi Zargayouna received his M.Sc degree in computer science and the Ph.D. degree in computer science and artificial intelligence from the University of Paris Dauphine (France), in 2003 and 2007 respectively. Since 2008, he is researcher in computer science and transportation science at IFSTTAR, the French Institute of Sciences and Technologies for Transport, Development and Networks, in the Transport Engineering and Computer Science Lab GRETTIA. In 2009, he was visiting researcher at TU-Delft (Netherlands). He is since 2014 head of the "Modeling & Multimodality" research team.

Mahdi Zargayouna is mainly interested in multiagent systems (languages, coordination models, simulation, planning, etc.), and complex transportation applications (traveler information, crisis management, dial a ride, urban parking, etc.). He teaches multiagent systems and intelligent transportation systems at University of Paris Dauphine, University Paris Est, and EPITA Engineering School. He has published more than 50 papers in peer-reviewed journals and conference proceedings and is member of the reviewer board of several international journals and conferences.

Gérard Scemama received his M.Sc degree and the Ph.D degree in mathematics and operational research from the University of Pierre et Marie Curie (France), in 1976 and 1977 respectively. He joined IFSTTAR, the French Institute of Sciences and Technologies for Transport, Development and Networks, in 1981. He is research director since 1992 and has been director of the Transport Engineering and Computer Science Lab GRETTIA for 14 years (from 1997 to 2011).

Gérard Scemama is mainly interested in supervision and regulation systems in multimodal networks. He is the inventor of Claire and ClaireSiti supervision systems, for road and multimodal networks respectively. During his 35 years of research, Gérard Scemama has managed several National and European projects and participated in scientific commissions of several institutes in France and Europe.

Omer Rana is Professor of Performance Engineering at Cardiff School of Computer Science & Informatics. He was formerly the deputy director of the Welsh eScience Centre at Cardiff University. He holds a PhD in "Neural Computing and Parallel Architectures" from Imperial College London. His research interests are in the areas of high performance distributed computing, data mining and analysis and multi-agent systems.

Prior to joining Cardiff University he worked as a software developer with Marshall BioTechnology Limited in London. He is an associate editor of the ACM Transactions on Autonomous and Adaptive Systems, IEEE Transactions on Cloud Computing, series co-editor of the book series on "Autonomic Systems" by Birkhauser publishers, and on the editorial boards of "Concurrency and Computation: Practice and Experience" (John Wiley) & Journal of Computational Science (Elsevier).