Towards a Distributed Multiagent Travel Simulation

Matthieu Mastio[1], Mahdi Zargayouna[1], Omer Rana[2]

[1] Université Paris-Est, IFSTTAR, GRETTIA
Boulevard Newton, Champs sur Marne
F-77447 Marne la Vallée Cedex 2, France
[2] School of Computer Science & Informatics,
Cardiff University, United Kingdom
{matthieu.mastio, hamza-mahdi.zargayouna}@ifsttar.fr, ranaof@cardiff.ac.uk

Abstract. With the generalization of real-time traveler information, the behavior of modern transport networks becomes harder to analyze and to predict. It is now critical to develop simulation tools for mobility policies makers, taking into account this new information environment. Information is now individualized, and the interaction of a huge population of individually guided travelers have to be taken into account in the simulations. However, existing mobility multiagent and micro-simulations can only consider a sample of the real volumes of travelers, especially for big regions. With distributed simulations, it would be easier to analyze and predict the status of nowadays and future networks, with informed and connected travelers. In this paper, we propose a comparison between two methods for distributing multiagent travelers mobility simulations, allowing for the consideration of realistic travelers flows and wide geographical regions.

1 Introduction

Transport systems are more and more complex and they have to evolve to integrate more connected entities (mobile devices, connected vehicles, etc). Indeed, we can now provide optimal routes for the travelers but we are also able to update these routes in real time based on new network status (congestions, accidents, bus down, canceled carpooling, etc). Giving information to the traffic network users is generally good and allows the improvement of the global network traffic flows. However, without control, the massive spread of information via billboards, radio announcements and individual guidance may have perverse effects and create new traffic jams. Indeed, with this generalization of real-time traveler information, the behavior of modern transport networks becomes harder to analyze and to predict. It is then important to model and simulate a realistic number of travelers to correctly observe these effects.

The ability to run a traffic simulator with real volumes of travelers at a city, a region or a country scale, would allow to observe the consequences of different information strategies on the status of multimodal traffic before implementing them in the real world. The management of millions of travelers in real time requires considerable computational power and current mobility simulations do not scale up in a way that would make it possible to predict the effects of regulation and information actions on the network. Our main objective in this paper is to test the scalability of this kind of simulators and develop scenarios at actual city scale. For this purpose, we aim to split the simulation between several servers on a grid and balance the load optimally between the servers while minimizing inter-server communications.

To have a generic distribution pattern, we propose a reference mobility simulator based on the multiagent paradigm. The multiagent paradigm is relevant for the modeling and simulation of transport systems [1]. This is why the multiagent approach is often chosen to model, solve and simulate transport problems. This approach is particularly relevant for the simulation of individual travels since the objective is to take into account human behaviors that interact in a complex, dynamic and open environment.

The remainder of this paper is structured as follows. In section 2, we present the previous proposals for travelers mobility simulation and the existing distributed multiagent platforms. Section 3 presents a formal definition of the multiagent environment. In section 4, we describe two methods for distributing simulations over several hosts. Section 5 explains our experimental setup and provides a comparison of the two proposed methods, before concluding and describing some further work we are pursuing.

2 Related work

There exists several multiagent simulators for travelers mobility. For instance, MATSim [2] is a widely known platform for mobility micro-simulation. However, the mobile entities in MATSim are passive and their state is modified by central modules, which limits its flexibility and its ability to integrate new types of (proactive) agents. Transims [3] simulates multimodal movements and evaluates impacts of policy changes in traffic or demographic characteristics while Miro [4] reproduces the urban dynamics of a French city and proposes a prototype of multiagent simulation that is able to test planning scenarios and to specify individuals' behaviors. AgentPolis [5] and SM4T [6] are also multiagent platforms for multimodal transportation. Finally, SUMO [7] is a widely used microscopic simulator mainly focused on traffic. However, none of these proposals considers the distribution problem.

Some general-purpose multiagent platforms have been specifically developed for large scale simulation in the last years. RepastHPC [8], a distributed version of Repast Simphony, uses the same concepts of projections and contexts and adapts them for distributed environments. Pandora [9] is close to RepastHPC and automatically generates the code required for inter-server communications. GridABM [10] is based on Repast Simphony but takes another approach and proposes to the programmer general templates to be adapted to the communication topology of his simulation.

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 develop distributed simulations, but prevents him from optimizing the distribution. The best way to manage the communications depends of the application and using such general platforms for a travel simulator would not produce optimal results. More theoretical works study general methods to address this problem. In [11] and [12] the authors propose to relax some synchronization constraints to achieve a better scalability by reducing the time the hosts wait for each other. In [13] and [14] the authors discuss the issues related to multiagent simulation in a distributed virtual environment. In the present paper, we propose specific approaches to distribute traffic-based simulations, which could be of great benefit to the work on travelers mobility simulations.

3 The Multiagent Environment

3.1 The model

The multiagent environment of a travel simulation is made of the transportation network in which the traveler agents evolve. We model the transport network 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. A set of agents A is traveling in this network from origins to destinations trying to minimize their travel time. The travel time of an edge at time t depends on the number of agents using it. To calculate this time, we use a triangular fundamental diagram of traffic flow that gives a relation between the flow q (vehicles/hour) and the density k (vehicles/km). The fundamental diagram suggests that if we exceed a critical density of vehicles k_c, the more vehicles are on a road, the slower their velocity will be. Here is the equation we use to model this phenomena:

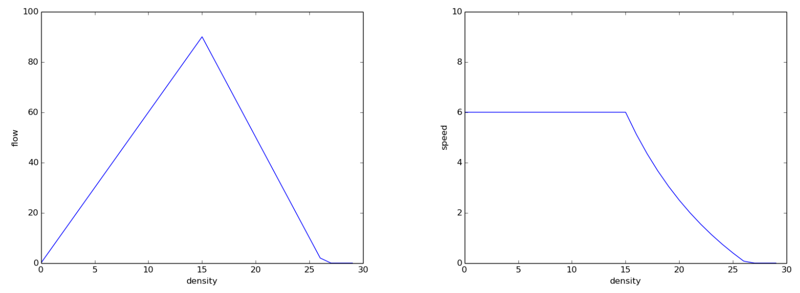

Fig. 1. Fundamental diagram with $\alpha = 6$, $\beta = 8$ and $k_c = 15$ (left). Speed in function of density (right).

$$q = \begin{cases} \alpha k & \text{if } k \leq k_c \\ -\beta(k - k_c) + \alpha k_c & \text{if } k > k_c \end{cases} \tag{1}$$

This equation is parametrized with α the free flow speed on this road, β the congestion wave speed and k_c the critical density. As $v = \frac{q}{k}$:

$$v = \begin{cases} \alpha & \text{if } k \leq k_c \\ \frac{-\beta(k-k_c)+\alpha k_c}{k} & \text{if } k > k_c \end{cases} \tag{2}$$

Thus we can define a cost function that returns a travel time per distance units $(1/v)$ in function of the number of agents $|A_e|$ on this edge:

$$cost(|A_e|) = \begin{cases} \frac{1}{\alpha} & \text{if } |A_e| \leq k_c \\ \frac{|A_e|}{-\beta(|A_e|-k_c)+\alpha k_c} & \text{if } |A_e| > k_c \end{cases} \tag{3}$$

Both edges and vertices are weighted with positive values evolving dynamically with the number of agents present on them. Given A_v, the set of agents on a vertex v, and A_e, the set of agents on a edge e, we have $|v| = |A_v|$ representing the weight of a vertex and $|e| = |A_e|$ the weight of an edge. $|V| = \sum_{v \in V} |v|$ is the weight of a subset of vertices. In the same way, if $|e|$ is the weight of a vertex, $|E| = \sum_{e \in E} |e|$ is the weight of a subset of edges.

3.2 The simulator

We have developed a reference simulator where each agent represents a traveler evolving in a multiagent environment as described in the previous paragraph. Agents appear nondeterministically with an origin and a destination vertex. They compute the shortest path based on the current status of the network before to start traveling. They ask for a new shortest path each time they reach a vertex in their path, to check wether a new shortest path becomes possible, following the dynamics of the network. At each time step of the simulation, if the agents are currently on an edge, they go forward as far as the road state (current mean speed) allows them to go. The simulation ends when all the travelers have reached their destinations or when a time step threshold is reached.

4 Proposed Approach

To launch our simulation at a city scale, we need a large memory and computing power. This is why we aim to deploy it on several machines. A simulation running on a cluster is typically SPMD (Single Program Multiple Data), i.e. each processor runs the same program but owns only a part of the program data in its private memory, and all the processors are connected by a network. Communications are explicitly declared by the programmer. The advantage of this approach is its high scalability; it can be implemented on most parallel architectures and we can deploy the same simulation on larger systems if we need more power.

In mobility simulations, traveler agents are moving on a transport network. To distribute such simulations, we have to split the workload between the available servers efficiently. In our model, the main workload is generated by the calculation of the shortest path. An agent's path has to be recalculated each time this agent reaches a new intersection (because the travel times evolve dynamically), thus an agent can be considered as a unit of workload. Therefore, in order to distribute this model we need to split the agents between the servers. To do so, we can either distribute the environment, or the agents.

4.1 Agent distribution

One approach would be to cut the set of agents in k equal parts (with k the number of available servers), distribute each subset on a server and run the simulation. As the travel times depend on the number of agents on each arc, all the agents need to know at every step how many agents there are on each arc in order to compute their shortest path. In our implementation, at each time step, if an agent managed by a server quits or arrives on an edge, this server communicates the information to all other servers. Thus at any time step, all the servers know the state of the entire network. At the moment, it is the only communication needed for the simulation, so the total communication cost is $k|E|I$, with I representing the size of an integer[1].

4.2 Environment distribution

The second distribution pattern tries to keep agents that are close in the graph on the same server. Instead of distributing the agents, we distribute the vertices and the outgoing edges (and thus the agents located on these vertices and edges) so the agents that are in the same place of the graph are on the same server.

Each server is only aware of what is happening on the part of the graph that it is managing. So at each time step, before the agents act, all the servers need to synchronize with the servers running in other processes. There are now two types of communications: the servers have to communicate the weight of their edges (the number of agents on it) and when an agent moves to a vertex that is not on his current server he has to move to that vertex's server. Let C be the cost of the edge communications and M the agents migration cost. At each step, the total communication cost is given by $T = C + M$. The cost of the edges weight communications is: $C = |E| \times I$. An agent could be coded with three integers (ID, current location and destination). Let n be the number of migration for one step. Thus the agents migration cost is: $M = 3In$. There are on average $|A|/|E|$ agents per vertex. So with E_c the set of edges between different servers we have on average $n = |A|/|E| \times |E_c|$. Thus $T = I(|E| + 3|A|/|E| \times |E_c|)$. As we could expect, the less edges there are between two servers and the less the communication cost there is.

[1] Coding the number of agents arriving or leaving an edge

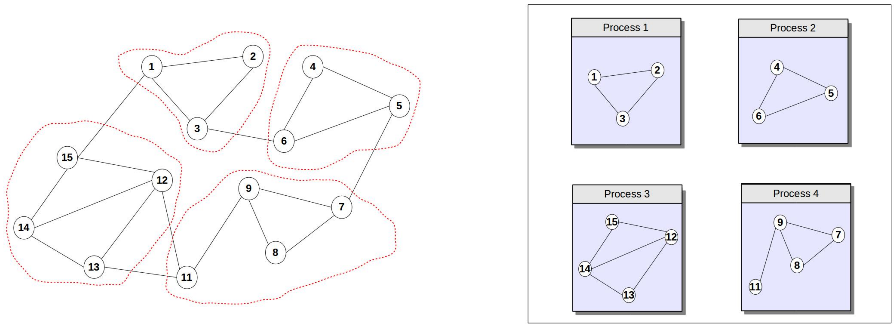

Fig. 2. The graph is parted and each part are distributed between the available processes.

So for the environment distribution method to be effective, we need to split the vertices into k disjoint sets such that each set has approximately the same vertex weight and such that the cut-weight, the total weight of edges cut by the partition, is minimized. This problem is known as the $(k, 1+\epsilon)$-balanced partitioning problem, that is the problem of finding a collection of disjoint subsets $V_1, \ldots, V_k$ that cover V , i.e., $V = V_1 \cup ... \cup V_k$ such as each part contains at most $(1+\epsilon)\frac{|A|}{k}$ and $|E_c|$ is minimized.

The problem of partitioning a network has been widely studied in the scientific literature. As demonstrated in [15] this is a NP-hard problem so trying to find an optimal solution with, for example, integer programming is not an option for large graphs. This is why some heuristics have been proposed to solve this problem in reasonable time. The multilevel partitioning method has been recognized as a very powerful method that offers a more global vision on graphs than traditional techniques. As the complexity of the partitioning problem is dependent on the size of the partitioned graph, the simple idea of multilevel partitioning is to regroup the vertices and to work with the groups instead of the independent vertices. The multilevel partitioning has been formalized in a generic framework by Walshaw in [16]. To distribute the environment, we use a slightly modified version of the Differential Greedy algorithm [17]. We modified this algorithm to use it with weighted vertices and to produce more connected partitions.

5 Experiments and Results

5.1 Implementation

To test the effectiveness of our approach, we have implemented our model and deployed it on an actual cluster. We choose Python to develop the model since this language is efficient for 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 [18].

For the inter-process communications we use MPI, that is the *de facto* 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. Moreover MPI4PY is an efficient interface that allows to use MPI with Python.

5.2 Results

We have launched the distributed simulations on the Cardiff University cluster. For our tests, we used eight hosts under CentOS Linux (kernel version 2.6.32-220) on a processor Intel Xeon CPU E5-2620 (12 cores at 2Ghz) with 32GB of memory. We ran the simulation on three configurations: the first is a sequential version of the program on a single host (conf1), the second is a distributed version on the eight hosts (conf2), and the last is run on the eight hosts using the 12 cores of each one (conf3). The simulation is performed for 100 time steps on a 200 nodes power-law graph generated with the Barabasi-Albert model [19]. We compared the two methods of distribution (agent-based and environment-based distributions) on the different configurations with an increasing number of agents (from 1,000 to 40,000).

number of agents	1000	5000	10000	20000	30000	40000
conf1 (1 core)	10	27	43	67	104	140
conf2 agent distribution (12 cores)	3	6	8	12	17	23
conf3 agent distribution (96 cores)	2	3	3	4	4	5
conf2 environment distribution (12 cores)	6	13	17	22	31	41
conf3 environment distribution (96 cores)	5	10	11	11	15	17

Table 1. Computational times (in seconds) for a 100 time steps simulation on a 200 nodes scale free graph.

As we could expect, the agent-based distribution is more effective than the environment-based distribution with the proposed simulation model[2]. Indeed, at the moment we have not defined any local interactions in our model. As a consequence, we are in an perfect case for the agent base distribution since the amount of inter-server communications will be limited. But with the further implementation of local interactions (for example pursuit model or vehicle to vehicle communications) the environment-based approach will be able to take

[2] The computational times are not strictly growing with the number of agents for the environment-based method. This is more likely due to the random origins and destinations of the agents. Therefore the simulation could sometimes be more complex with fewer agents.

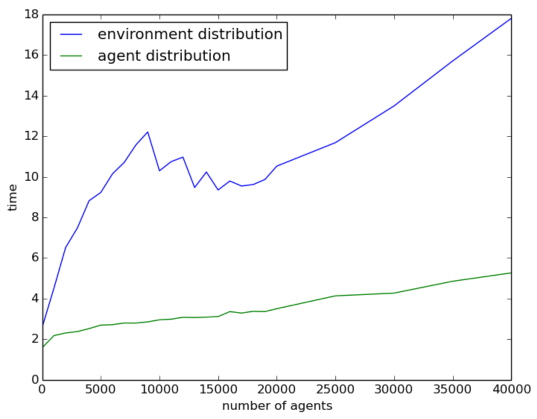

Fig. 3. Comparison of computational times between the different distribution methods with conf3.

advantage of the collocation in the same server of physically close agents. Furthermore, there is at the moment no dynamic load balancing mechanism for the environment distribution. Indeed, if an important number of agents are concentrated in the same part of the network, they will be nevertheless in the same server. It will hence take more time for this server to calculate all the shortest paths and, all the other servers will have to wait for it. Thus, if a server is overloaded it can slow down all the simulation.

The speedups between the sequential run and the two distribution methods performed on conf3 are shown on figure 4 and 5. The speedup is a measures of how much faster the simulation is on conf3 (96 cores) than on conf1. As we can see on these figures, with 40000 agents for example, the simulation is 8 times faster with the environment based distribution and 28 times faster with the agent based distribution. Both of these methods improve largely the execution time of our multiagent traffic simulator. As explained above, the agent-based distribution method shows better results due to the relative simplicity of the current model but after adding inter-agents communications the difference in performance between the two methods should shrink.

6 Conclusions and perspectives

In this paper, we have presented two distribution methods of a multiagent travel simulation over several hosts. The two methods are efficient to scale the simulation up with the number of agents. With our current simulation model, the agent-based method is more efficient than the environment-based distribution method. Our simulation model is very general, our proposals and findings are applicable to all the state-of-the-art travel simulators.

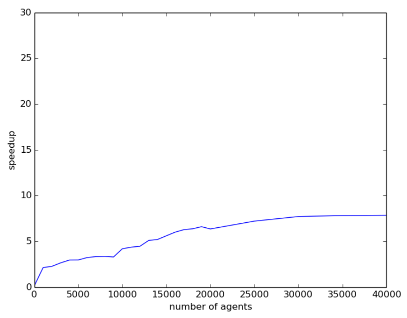

Fig. 4. Speedup between conf1 and conf3 with environment distribution.

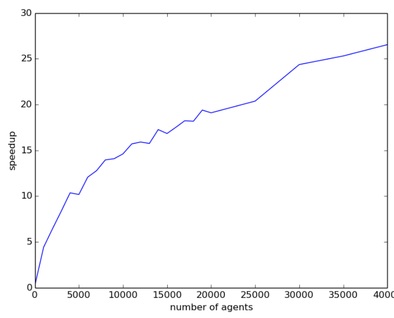

Fig. 5. Speedup between conf1 and conf3 with agent distribution.

Our future works will deal with two aspects. The first concerns the simulation model. We will tackle more specific travel simulation models, where all types of communications are present (local, global and community-based communications). We expect the environment-based distribution method to show better performance than with the current model. The second aspect concerns the environment-based distribution method. The distribution is currently done statically at the beginning of the simulation and we believe that the speedup could be largely improved by adding dynamic load balancing mechanisms.

References

1. Fabien Badeig, Flavien Balbo, Gérard Scemama, and Mahdi 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.
2. MichałMaciejewski and Kai Nagel. 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.
3. Kai Nagel and Marcus Rickert. Parallel implementation of the transims micro-simulation. *Parallel Computing*, 27(12):1611–1639, 2001.
4. Sébastien Chipeaux, Fabrice Bouquet, Christophe Lang, and Nicolas Marilleau. Modelling of complex systems with AML as realized in MIRO project. In *LAFLang 2011, workshop of the Int. Conf. WI/IAT (Web Intelligence and Intelligent Agent Technology)*, pages 159–162, Lyon, France, 2011. IEEE Computer Society.
5. Michal Jakob, Zbynek Moler, Antonín Komenda, Zhengyu Yin, Albert Xin Jiang, Matthew Paul Johnson, Michal Pechoucek, and Milind Tambe. Agentpolis: towards a platform for fully agent-based modeling of multi-modal transportation (demonstration). In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pages 1501–1502, 2012.

6. Mahdi Zargayouna, Besma Zeddini, Gérard Scemama, and Amine 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.
7. Michael Behrisch, Laura Bieker, Jakob Erdmann, and Daniel Krajzewicz. SUMO - simulation of urban MObility - an overview. In *SIMUL 2011, The Third International Conference on Advances in System Simulation*, pages 55–60, 2011.
8. Nicholson Collier and Michael North. Repast HPC: A platform for large-scale agent-based modeling. In Werner Dubitzky, Krzysztof Kurowski, and Bernhard Schott, editors, *Large-Scale Computing*, pages 81–109. John Wiley & Sons, Inc., 2011.
9. E.S. Angelotti, E.E. Scalabrin, and B.C. 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.
10. Laszlo Gulyas, Gabor Szemes, George Kampis, and Walter de Back. A modeler-friendly API for ABM partitioning. In *Proceedings of ASME 2009 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 219–226, 2009.
11. 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.
12. Matthias Scheutz and Paul Schermerhorn. Adaptive algorithms for the dynamic distribution and parallel execution of agent-based models. *Journal of Parallel and Distributed Computing*, 66(8):1037–1051, 2006-08.
13. Beatrice Ng, Antonio Si, Rynson W.H. Lau, and Frederick W.B. Li. A multi-server architecture for distributed virtual walkthrough. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, VRST '02, pages 163–170. ACM, 2002.
14. Omar Rihawi, Yann Secq, and Philippe 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.
15. Thang Nguyen Bui and Curt Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42(3):153–159, May 1992.
16. Chris Walshaw. Multilevel refinement for combinatorial optimisation: Boosting metaheuristic performance. In *Hybrid Metaheuristics*, number 114 in Studies in Computational Intelligence, pages 261–289. Springer Berlin Heidelberg, January 2008.
17. C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *19th Conference on Design Automation, 1982*, pages 175–181, June 1982.
18. Hans Petter Langtangen and Xing Cai. On the efficiency of python for high-performance computing. In *Modeling, Simulation and Optimization of Complex Processes*, pages 337–357. Springer Berlin Heidelberg, January 2008.
19. Albert-László Barabási and Réka Albert. Emergence of scaling in random networks. *Science*, 286(5439):509–512, October 1999.