Self-Balancing Decentralized Distributed Platform for Urban Traffic Simulation

Quentin Bragard, Anthony Ventresque, and Liam Murphy

***Abstract*—Microscopic traffic simulation is the most accurate tool for predictive analytics in urban environments. However, the amount of workload (i.e., cars simulated simultaneously) can be challenging for classical systems, especially for scenarios requiring faster than real-time processing (e.g., for emergency units having to make quick decisions on traffic management). This challenge can be tackled with distributed simulations by sharing the load between simulation engines running on different computing nodes, hence balancing the processing power required. This paper studies the performance of dSUMO, a distributed microscopic traffic simulator. dSUMO is fully decentralised and can dynamically balance the workload between its computing nodes, hence showing important improvements against classical, centralised and not dynamic, solutions.**

I. INTRODUCTION

Simulation of urban dynamics is a popular topic in the literature and in industry. Traffic prediction [1–3], crowd management [4], [5], evaluation of V2X solutions [6], [7], all benefit from using fine-grained simulation which is why the area has been studied widely and has led to numerous tools, such as SUMO [8], CityFlow [5] or DynaTAI-WAN [9].

In particular, microsimulation, i.e., simulation of individual agents, such as, drivers and pedestrians, gives the most accurate results and is hence widely used [10]. The challenge with this type of simulation is that it requires important computing resources to process typical city-scale scenarios which easily involve hundreds of thousands of vehicles simultaneously. On top of that, other elements, such as, pedestrians, communications between agents and the infrastructures, can increase significantly the running time of the computation.

Distributed systems are a common way to tackle processing power issues: load is divided over computing nodes called Logical Processes(LPs) with each LP processing its part and sharing information about its state. Distributed simulations rely on partitioning algorithms to distribute the load, synchronization and communication algorithms to share information, and the simulator itself. While the necessity of partitioning, communication and synchronization mechanisms is straightforward, it is also globally accepted that, for efficient distributed simulations, dynamic load-balancing algorithms are required to balance fairly the amount of loads between the LPs [11–13]. If one LP has more load to compute than its peers, it will take it longer to update its share of the simulated environment. The other LPs will have to wait for the slower nodes to pursue the simulation, therefore the performances of the entire distributed simulation will be hampered by the slowest LP. In order to achieve a large-scale, fine granularity simulation in a decent amount of time, a *distributed simulation would need to be scalable but also self-balancing and accurate, i.e., give the same results a single simulator would give.*

In a previous paper [14] we described dSUMO, a platform for distributed microscopic traffic simulation. We interfaced it with SUMO, the Simulator for Urban MObility, a microscopic vehicle simulator developed by the DLR [8]. We also showed that the accuracy of dSUMO when comparing the distributed version and the single-simulator version, is almost 100%.

In the current paper, we add synchronisation and self-balancing mechanisms to dSUMO,

Manuscript received ... . This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855. Anthony Ventresque was supported, in part, by Science Foundation Ireland Industry Fellowship grant 13/IF/12789. Corresponding author: Q. Bragard (email: quentin.bragard@ucdconnect.ie).

Q. Bragard, A. Ventresque and L. Murphy are with Lero@UCD, School of Computer Science, University College Dublin, Ireland

A. Ventresque was also with IBM Research, Damastown Industrial Estate, Dublin, Ireland

and we do a thorough performance evaluation of dSUMO on a realistic scenario. The results showed that dSUMO is scalable and significantly reduced the simulation time (by up to 55% simulation time saved).

In the remainder of the paper, we first do a literature study (Section II); we then detail dSUMO (Section III); Section IV explains the dynamic synchronization and dynamic load-balancing algorithms implemented; then (Sections V and VI) we present the experiments and show the performance of our platform on a realistic city-scale scenario; Section VII discusses the difference between load-balancing in traffic microscopic simulation contexts and in other behavioural simulation. Finally, Section VIII concludes this paper.

II. MOTIVATION AND RELATED WORK

[15], [16] show that one of the main issues for distributed microscopic urban traffic simulation is the synchronization. The central unit used in Suzumura and Kanezashi [15] becomes a bottleneck when the number of partitions increases. Cetin et al. [16] try to limit the synchronization cost by using faster network connexions, which is expensive, both for researchers and most urban authorities, while their scenario was of limited scale and probably not realistic. dSUMO [14] is a distributed and decentralised microscopic simulation and overcomes both problems: there is no central entity, the bottleneck element for the synchronisation, and it does not need to run on dedicated powerful servers – any commodity machine could run dSUMO. Our scenario is also derived from a real observation of the traffic condition in the city of Cologne. Note that the synchronisation problem is not limited to traffic simulations: air traffic management, another computationally expensive process, is also moving towards using decentralized and distributed engines [17], [18].

In [19] Lee and Chandrasekar present a parallel version of Paramics [20] using the shared memory on a multi-core system. They note the importance of load-balancing while not implementing such a mechanism (not even a static one) and they work on small systems (2 or 3 cores) and on simple urban scenarios (grid like generated road network). On the contrary, dSUMO has a dynamic load-balancing mechanism and in this paper we propose large scale simulation, with up to 16 LPs and a real scenario.

Barceló et al. [21] use a multi-core architecture as well for their parallel microscopic simulator. The limitations of their approach are that the speed-up is logarithmic and the accuracy (i.e., how close it is to the results obtained with a single engine) of the parallel simulator is not 100%. On the contrary, dSUMO shows a linear speed-up and a perfect accuracy [14]. Another point worth mentioning is that the workload partitioning in [21] is not the usual space partitioning but a driver partitioning: the agents (drivers) are distributed over the different computing nodes. This has been proven inefficient for microscopic traffic simulations given the complexity of the car following models (how each driver makes a decision based on the car before them) which requires a lot of communication between partitions [22].

Nagel and Rickert [23] propose a parallel version of Transims implementing a master/slave model of communication and an interesting pseudo-load-balancing – they use a heuristic to adapt the load at each turn. Their results, although the authors admit their solution does not scale, are very good and interesting. dSUMO is different on three aspects: it is fully decentralised, it has an explicit load balancing mechanism and it can accommodate different microscopic simulator systems running the same global simulation.

dSUMO is a multi-engine distributed traffic simulator, i.e., it can ,in theory, run various microscopic simulators together in a single distributed simulation. What dSUMO requires is only that the simulators provide APIs that allow to dynamically instrument the simulation engine (e.g., SUMO's Traci, VISSIM's COM interface). So far we have focused on SUMO, a very popular [10] open-source microscopic traffic simulator. SUMO has been used extensively in research projects with high visibility, such as, iTetris [24] or CityMobil [25]. We have shown [14] that the accuracy of dSUMO is almost 100%, which means that there is no difference between running a distributed simulation and the same simulation on a single node in terms of the results obtained. In the current paper, we address the other main challenge of distributed traffic simulations: the *performance* of the distributed system. We show that the decentralised

nature of dSUMO (i.e., there is no central entity for the synchronisation) and the load-balancing technique implemented make dSUMO scalable and improve the running time of the simulation.

III. DSUMO

In this section, we describe the architecture of our solution and the different choices we have made to ensure the scalability of dSUMO: a dynamic load-balancing and a conservative local synchronization mechanisms.

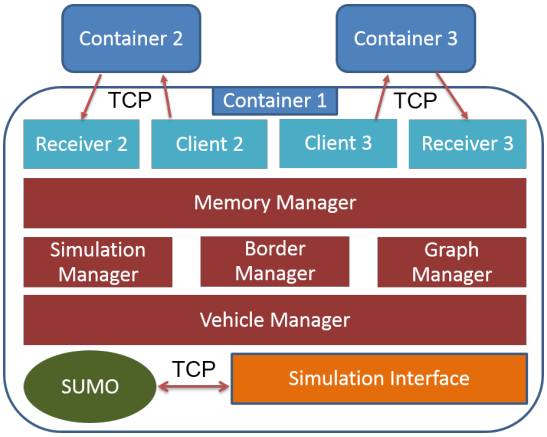


Fig. 1: Software architecture of our decentralized distributed platform.

A. Architecture

Figure 1 presents the software stack of dSUMO. At the top of the stack are the objects used for the communication between nodes: *Client* and *Receiver*. When a connection is made with a neighbour, a Client and a Receiver are created to respectively send and receive information from this neighbour. Messages exchanged contain synchronization information, vehicles data and load exchange. Incoming data is parsed and then inserted into the shared memory. Shared objects are managed by the singleton *Memory Manager* which ensures their consistency and manages concurrent access. *Simulation Manager* is the core of dSUMO and handles the simulation at a high level. The Simulation Manager initiates the simulation and executes sequentially the following phases: processing of new steps, transferring vehicles data to Clients (i.e., sending them to the neighbours when they cross the borders between partitions), applying vehicles data from Receivers (i.e., new vehicles arriving from neighbouring nodes), and, if required, triggering dynamic synchronization and dynamic load-balancing. The *Border Manager* is a singleton handling the links between Client/Receiver and incoming/outgoing borders. Other objects request the Border Manager when they receive or send vehicles data. When the load is balanced during the simulation, the Border Manager ensures that the borders remain consistent. *Graph Manager* stores the sub-graph of the road-network representing the partition as well as road data (e.g., speed limits). When the load-balancing mechanism is triggered, the Graph Manager provides the different exchangeable roads and their connections. Graph Manager also ensures the consistency of the sub-graph throughout the simulation. Finally, *Vehicle Manager* provides a broad range of high level functions: retrieving vehicles' and borders' data and adding, removing and modifying simulation elements such as vehicles. At a lower level, *Simulation Interface* manages the connection with the simulator. We developed our own interface based on TraCI4J, a Java adaptation of SUMO communication interface TraCI. Finally, the simulator SUMO processes the simulation. However, dSUMO is extensible and can easily be interfaced with others traffic simulators.

B. Conservative Local Synchronization

As explained by Fujimoto [26], two types of synchronization mechanisms can be used in distributed simulations: conservative and optimistic. A synchronization is conservative when all LPs know when they can process safely the next step or event, while a synchronization is optimistic when LPs process steps/events that may lead to inconsistencies (they then need to roll-back to a coherent state). There is no consensus on which mechanism is the best and it is suggested to be scenario dependent. However, while the optimistic mechanism prevents some waiting time, it requires a roll-back mechanism which can be expensive in terms of processing time as well as memory consumption; especially in the context of traffic simulation where reversing a step is not easy or even possible.

C. Borders Management

In order to compute the next position of a vehicle on a road, SUMO needs information about the preceding vehicle such as speed and acceleration. To make sure that the simulator has this information, the borders between two LPs are duplicated and processed by both. When a vehicle arrives at an outgoing border, it is sent to the next LP but the original LP keeps the vehicle in its simulation. At the following step, the next LP processes the new position, speed and acceleration of the vehicle and send them to the original LP which corrects the position of the vehicle on its part of the shared border. Therefore, if a traffic jam appears on one side, it will grow properly through the border if needed. The duplication of the borders creates some difficulties and constraints but this is the price to pay to have an accurate distributed simulation. For instance as if dSUMO needs to ensure that vehicles do not ”jump” over a border when the simulator runs step-by-step (e.g., SUMO). In dSUMO this is solved by an algorithm that makes sure that the length of the road at the border between partitions is long enough to see every vehicle crossing the border (avoiding the jumping risk): checking both the length of and speed limit on roads when partitioning.

IV. DYNAMIC LOAD-BALANCING

As mentioned in the introduction, a load-balanced simulation is desirable as it makes better use of the resources provided by the different LPs. However, as a simulation progresses, the workload on each LP may evolve and the system is likely to become unbalanced. In this section we present a Dynamic Synchronization (DS) and Dynamic Load-Balancing (dLB) mechanisms for decentralized simulations. The algorithm is triggered by an LP that we call here *tLP* when the difference between its load and its neighbours' loads is greater than a defined threshold (in our scenario, the threshold is set at 40%).

A. Dynamic Synchronization

Two algorithms [27], TaSyn and GenSyn, perform dynamic synchronization in a decentralized distributed simulation. Most distributed simulations rely on central entities for synchronization, creating a risk of bottlenecks when the number of LPs increases. dSUMO synchronizes only locally to tackle this risk, which makes a global synchronization more complicated: for instance when all LPs need to decide which simulation step is the deadline before a load-balancing. TaSyn and GenSyn have different mechanisms to address this: TaSyn uses the eccentricity of the graph formed by the interconnected LPs to suggest to each LP a step to stop at (this use the fact that an LP can only be a step ahead of its neighbours).GenSyn does not require any topological information to perform the same task but is a bit slower than TaSyn. GenSyn starts by sending a request to its neighbours to know which LP is the most advanced in their surrounding. The request is propagated in every LP which stops its simulation and gets back with the step of the most advanced LP. GenSyn then sends a second message including the step where everyone should synchronize. It is a bit slower as two ‘waves’ of messages need to be sent and LPs cannot continue the simulation during this time.

B. Dynamic Load Balancing

In a prior work [28], we presented a global dynamic load-balancing (DLB) algorithm, working on a simulation of flocks of birds, to show its efficiency against common local load-balancing. Figure 2 shows the theoretical differences between dynamic global load-balancing (a) and dynamic local load-balancing (b). In Fig. 2 (a), the simulation stops longer than in Fig. 2 (b) but produces a better result and has an immediate impact, while in Fig. 2 (b) the load needs to be propagated through several LPs before an approximated balance is reached. The general idea is to gather topological information during the global synchronization and spread the load evenly among the LPs in one shot. The algorithm introduced in [28] was presented as a proof-of-concept and, after further analyses, showed some limitations in the context of traffic simulations. First of all, the algorithm can only send load currently on the borders between the sender and the receiver (excluding, for instance, crossroads connected to the sender, the receiver and a common neighbour). This limitation reduces the amount of exchangeable parts and, in the long term, creates complex and communication-inefficient regions. Second, when the number of

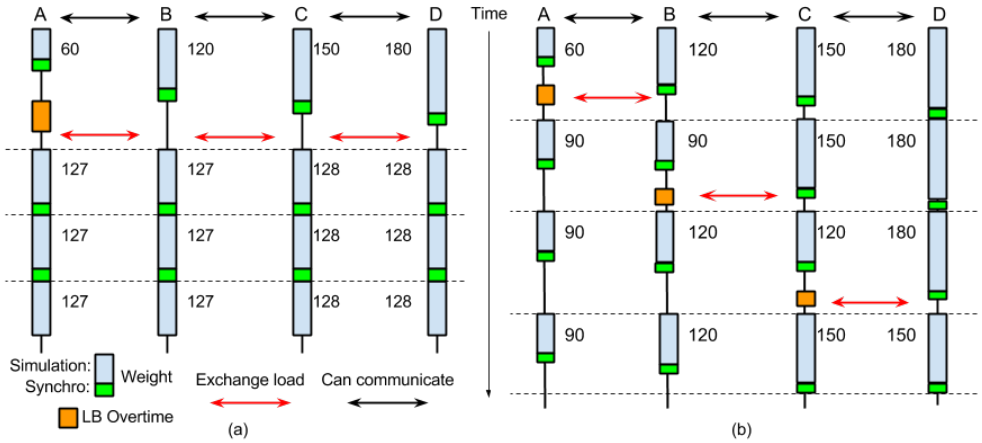


Fig. 2: Theoretical evolution of global load-balancing (a) and local load-balancing (b) techniques.

LPs increases, circular exchanges (LP *A* gives load to LP *B*, which gives to LP *C*, which gives load to LP *A*, etc.) generate deadlocks. Finally, the algorithm was designed to balance space-partitioned environments, which has been demonstrated inefficient for traffic simulation [23], [29]. However, the new DLB algorithm we present in this paper comes from the same general ideas, but removes the risk of deadlocks, works on graph-like environment and allow mores freedom in the exchanges.

When LPs answer to the dynamic synchronization, they send information about their current load as well as which LPs are their neighbours to *tLP*, the LP triggering the synchronization. *tLP* then generates a matrix of loads to be exchanged between any two LPs in the system. The matrix is simplified until at maximum only one exchange remains between any two LPs and circular exchanges are removed. Next, *tLP* establishes an order in which loads are going to be exchanged to maximize the number of exchangeable roads. This slows down the load-balancing process a little but improves the quality of the results. Finally, *tLP* sends exchange orders which are propagated over to every LP. Locally, the load balancing is done by transferring crossroads, roads and vehicles, from one LP *A* to one of its neighbours *B*, with the goal to minimize the number of borders and maximize the balance. The algorithm is based on a region-growing mechanism. It starts by ranking all exchangeable crossroads at the border between *A* and *B*. Next, the algorithm spreads selecting crossroads and adding their neighbours into the list of exchangeable crossroads. The process is repeated until the required amount of vehicles to be sent is reached. The heuristic used to rank the crossroads is an average of a crossroads' values and their surrounding's values, following those equations:

$$h(c) = \frac{h_s(c) + \sum_{i=1}^{N} h_s(n_i)}{N+1} \tag{1}$$

$$h_s(c) = \frac{B_{borders}}{R_{normal}} \tag{2}$$

with c the current crossroad, n the list of neighbours of c, N the number of neighbours, $B_{borders}$ and R_{normal}, respectively the number of roads connected to c which are borders and the ones which are normal. Therefore, the stronger a crossroad and its surrounding are connected to the neighbouring LP, the higher its value will be.

To balance between the constraint of minimizing the communication and maximizing the load-balance, the algorithm also uses two thresholds, a lower threshold *lt* and an upper threshold *lu*, calculated as a percentage of the load to exchange. When *lt* is reached, the algorithm starts recording the crossroads selected and stores the crossroad at which the number of borders is minimized. When *lu* is reached, the algorithm stops and roll-backs to the last stored crossroads between *lt* and *lu*. This method allows a trade-off between the number of

borders generated and the load balance. See Figure 3 for a graphical description of the algorithm used for dynamic load-balancing in dSUMO. Empirical results show that, in our scenarios, 60% and 140% of the original load to exchange are efficient lower and upper thresholds.

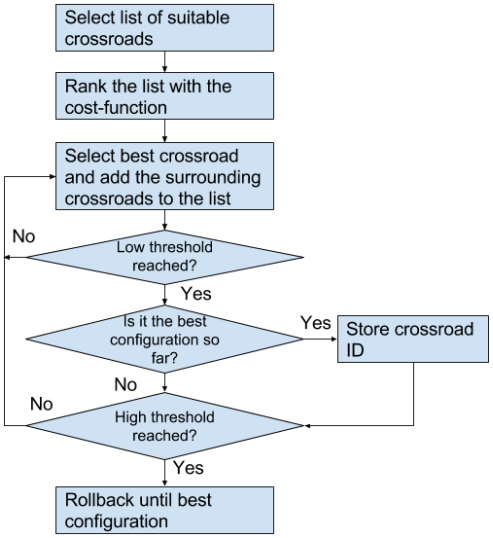


Fig. 3: Flowchart of the dynamic load-balancing algorithm.

V. EVALUATION

Our experiments have been conducted on Virtual Machines (VMs) with dual cores at 2.2GHz, 3GB of RAM running Windows 7. VMs are inter-connected with a 100Mb/s and communicate through Java sockets using TCP.

The environment we simulate is the city of Cologne, Germany, using the TAPASCologne scenario[30]. It is composed of 30,356 nodes and 68,642 edges representing intersections and road segments. In total 252,007 vehicles run on the simulation with a maximum of 130,000+ simultaneous vehicles. The simulation models the evolution of the traffic from 6am to 8am. The traffic starts appearing slowly all over the city and converges to the city centre leading to important traffic jams.

We partitioned the environment into 4, 8, 12 and 16 partitions using three different partitioning techniques: Quad-Tree, Smart Quad-Tree and SParTSim [29], [31]. Quad-Tree and Smart Quad-Tree are space partitioning algorithms, while SParTSim uses a region growing technique. Quad-Tree divides the space into four partitions of equal size. Then it keeps dividing the largest partition by four until a specific number of partitions is reached. Smart Quad-Tree, on the other hand, starts by dividing the space into a large number of small tiles. Then, it chooses n tiles to be the starting points of the n partitions requested. One after the other, partitions choose surrounding tiles with the goal to minimize the number of roads at the border. As long as tiles are available, partitions keep growing. SParTSim is an algorithm designed to partition environments for traffic simulation and works in three phases. First, the algorithm chooses nodes on the road network, called seeds, according to their importance (based on the number and the hierarchy level of incoming and outgoing roads). The seeds grow by merging with surrounding nodes until no node is free (i.e., does not belong to any partition). The second phase consists in load balancing the partitions. Overloaded partitions send nodes to under-loaded partitions until the partitioning is balanced. During the last phase, the algorithm sacrifices a part of the balance to re-establish the connectivity within the partitions in case it is lost during the balancing phase.

The experiments focus on evaluating (i) the time gained thanks to the dynamic load-balancing mechanism, (ii) the cost of the synchronisation between the LPs, (iii) the speed-up gain with the distributed simulation.
The time gain with the dynamic load-balancing mechanism is measured by subtracting the time required to run the simulation with the mechanism to the time required without. The synchronization time is measured by subtracting the simulation time to the time it would have required to run the heaviest region on SUMO. The difference includes the time required to scan the borders, extract the vehicle data and send them to the neighbours, receive neighbours data and treat them. Finally the speed-up is measured by dividing the time required for a single instance of SUMO to run the scenario by the time required by dSUMO on the same scenario.

VI. RESULTS

We refer to every experiment in this Section using three elements: dLB if the dynamic load-

	4SPS	4SQT	4QT
No dLB (s)	2,447.22	2,895.13	2,638.69
dLB (s)	2,165.97	2,022.96	2,043.29
Gain (%)	11.53	30.13	22.52
	8SPS	8SQT	8QT
No dLB (s)	1,879.96	1,553.61	2,722.82
dLB (s)	1,514.87	1,186.19	1,515.31
Gain (%)	19.42	23.65	44.34
	12SPS	12SQT	12QT
No dLB (s)	1,487.41	1,483.08	2,687.86
dLB (s)	1,076.78	1,089.89	1,202.73
Gain (%)	27.60	26.51	55.26
	16SPS	16SQT	16QT
No dLB (s)	1,030.12	1,987.76	1,346.56
dLB (s)	894.99	1,048.21	919.24
Gain (%)	13.12	47.27	31.73

Table 1: Simulation time gain with Dynamic Load-Balancing

balancing has been used, the number of partitions (i.e., 4, 8, 12 or 16) and the partitioning scheme used (i.e., QT, SQT or SPS, for QuadTree, Smart-QuadTree and SParTSim, respectively).

Figure 4 represents the time required by the slowest LP to simulate one step when the dynamic load-balancing mechanism is disabled (curve named *Time per Step - no DLB*), and when it is enabled (curve named *Time per Step - DLB*). On the other hand Figure 5 represents the time required by the time the fastest LP has to wait at each step when the load-balancing mechanism is disabled (curve named *Waiting time per Step - no DLB*), and when it's enabled (curve named *Waiting time per Step - DLB*). The graphs show the curves as well as their trends to improve readability. Figure 4 presents a clear reduction of the time required to simulate one step; however the load-balancing mechanism has a bigger impact on the maximum waiting time per step. Figure 5 shows that the mechanism maintains an almost constant waiting time through the simulation. The steps when the load-balancing mechanism is triggered are easily identifiable around the steps 1000, 2050 and 5250, as the waiting time increases dramatically for one step but then gets back lower than it was previously.

Table1 presents the time gained when using the dynamic load-balancing mechanism throughout the simulation. The gains in percentage vary from 11.53% to 55.26% with an average of 29.42%. The dynamic load-balancing mechanism is highly dependent on the original partitioning scheme: when the original partitioning is good, the effect is small (e.g. 11.53% for 4SPS) and when the original partitioning is bad, the effect is important (e.g. 55.26% for 12QT). The time required to perform the dynamic synchronization is directly dependent on how strongly connected the distributed simulation is. The more connected it is, the faster the messages will propagate to the furthest LPs and come back. The time required for the dynamic load-balancing, on the other hand, is dependent on how much unbalanced the distributed simulation is. If important amounts of workload need to be exchanged, the dynamic load-balancing process will take more time.

Figures 6 show the evolution of the total synchronization time required by the different scenarios. The time required to perform the synchronization is dependent on the number of borders to scan and the number of vehicles which need to be transferred between partitions or updated. When the number of partitions increases, to keep the load balanced, the size of the borders increases as well. While a distributed simulation with a central unit would have to deal with the increasing amount of data exchanged, in our decentralized platform, the synchronization process is shared among all the LPs. Therefore, as shown in our results, the synchronization time remains stable, or even decreases, when we increase the number of neighbours.

Figures 7 show the speed-up gain in the different scenarios. As the synchronization and the simulation processing are distributed over all the LPs, the speed-up grows linearly when we increase the number of partitions for QT (Figure 7(a)) and SPS (Figure 7(b)). The linearity of SQT, Figure 7(c), is broken due to better-than-expected results for 8 partitions and lower-than-expected results for 16 partitions. The values in table 1 show that the original partitioning for 8SQT and the dynamic load-balancing mechanism achieve results much better than 8QT and 8SPS. On the other hand, the values show that 16SQT performs worse than 16QT and 16SPS.

VII. DISCUSSION

In distributed simulations, the overall simulation time is impacted by three elements: the processing

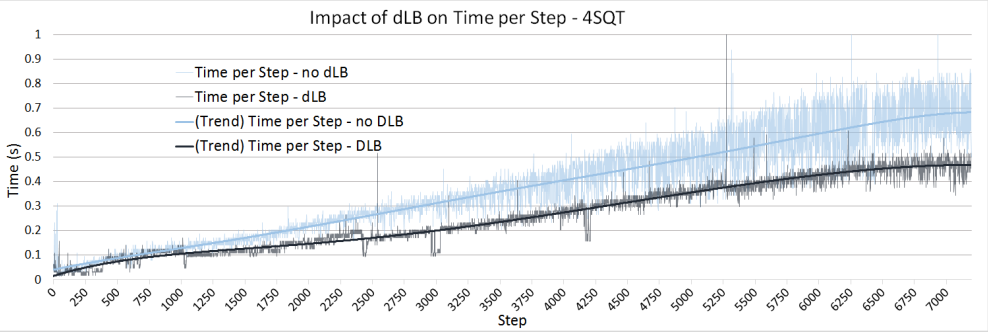


Fig. 4: Impact of dynamic load-balancing mechanism on the maximum simulation time per step: 4SQT vs. dLB4SQT.

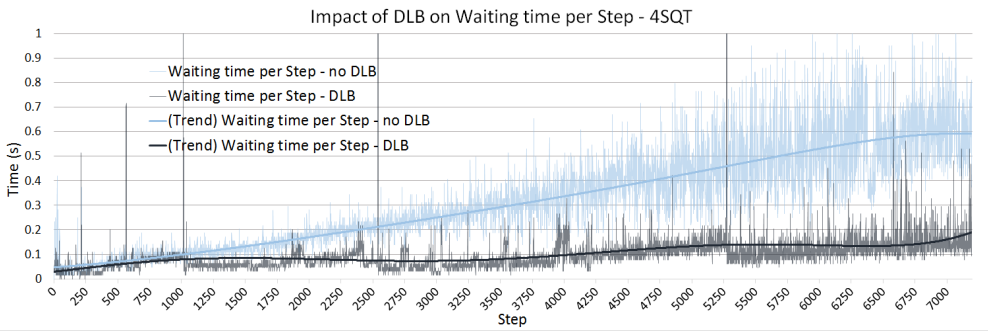
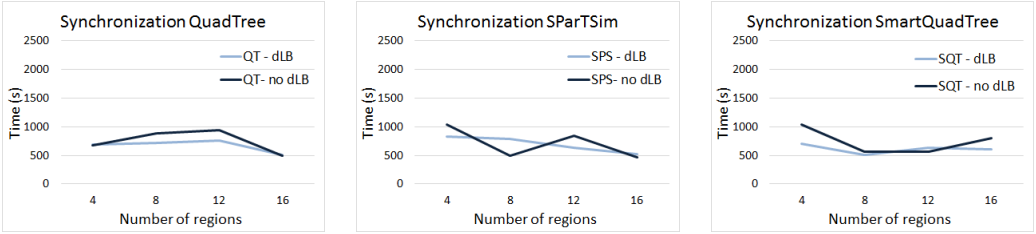
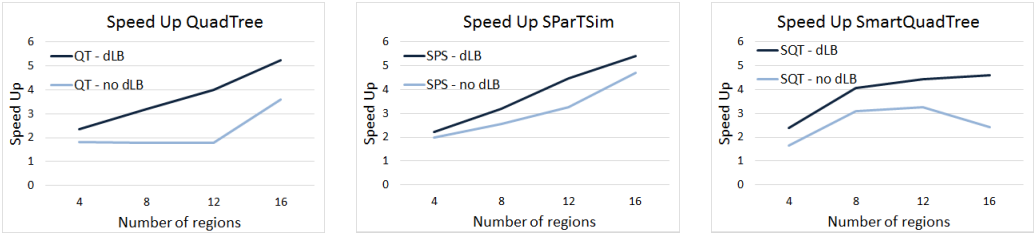


Fig. 5: Impact of dynamic load-balancing mechanism on the maximum waiting time per step: 4SQT vs. dLB4SQT.



(a) (b) (c)

Fig. 6: Synchronization time required by running dSUMO with up to 16LPs compared to SUMO using as partitioning algorithms (a) QuadTree (b) SParTSim and (c) Smart QuadTree.



(a) (b) (c)

Fig. 7: Speed-up gained by running dSUMO with up to 16LPs compared to SUMO using as partitioning algorithms (a) QuadTree (b) SParTSim and (c) Smart QuadTree.

time of each individual simulators, the communication time and the synchronization time. The first element is defined by the simulator used and issues such as unbalancing are solved with a load-balancing algorithm. The second element is limited by the network connection used. The last element is the synchronization cost (in other words the overhead due to the distribution) and can be unfairly spread among the LPs in a decentralized context. The complexity required to balance the synchronization cost over the LPs can dramatically change depending on the type of environment simulated: space or graph. In our previous work[28], we simulated a flock of birds in a 2D environment, partitioned with space-partitioning algorithms producing square tiles. The regularity of the tiles (each tile has exactly 8 neighbours) strongly links the size of the borders with the circumferences of the partitions. Border-balancing algorithms can use this regularity to improve their efficiency, but the regularity is absent in graph-like environments such as road networks. Road networks are represented by an oriented graph with each nodes having x incoming edges and y outgoing edges. Graph partitioning algorithms such as METIS [32] or SParTSim [29] can balance the number of edges at the borders, however they require knowledge of the entire graph to process the partitioning. In the context of decentralized distributed simulation, each partition has by definition only a local knowledge, limiting the efficiency of traditional graph partitioning algorithms. As far as we know, the problem of borders-balancing has not thoroughly been addressed in the context of decentralized distributed simulation and we believe that it would be an interesting and useful research topic.

VIII. CONCLUSION

We have presented in this paper our Self-Balancing Decentralized Distributed Platform for Urban Traffic Simulation to address the following problems: how can we distribute a microscopic traffic simulator, such as, SUMO, over several LPs? How can we improve the synchronization mechanism to increase the scalability of a distributed traffic simulator? And how can we balance a distributed simulation in a decentralized context?

We showed that our solution has a speed-up of 5.5 with 16 LPs. More importantly, the speed-up keeps improving at a linear pace while we increase the number of LPs in the distributed simulation up to 16 LPs. The dynamic load-balancing algorithm we propose is able to deal with both the evolution of the load and the defect of the original partitioning. Finally, we showed that the decentralized synchronization mechanism is scalable as the time required to synchronize the LPs is constant when we increase the number of LPs. It is important to notice that while the experiments in this paper have been conducted on SUMO only, dSUMO is microscopic simulator agnostic: as long as the simulator engine comes with APIs allowing to dynamically instrument the simulation (e.g., add/remove vehicles at run time), it can be added to dSUMO easily. In our future work, we would like to (i) test our platform on a larger and more complex scenario to verify that equivalent results can be obtained independently of the simulated environment, (ii) run our platform on a multi-core machine to test the improvement of the simulation time when the communication are reduced to a minimum and (iii) include the possibility to add and remove new LPs at runtime to adjust the number of LPs according to the global load, therefore making our platform more dynamic and fault-tolerant.

REFERENCES

[1] M. Ben-Akiva, M. Bierlaire, H. Koutsopoulos, and R. Mishalani, "Dynamit: a simulation-based system for traffic prediction," in *DACCORS short term forecasting workshop*, 1998.

[2] A. Abadi, T. Rajabioun, and P. Ioannou, "Traffic flow prediction for road transportation networks with limited traffic data," *IEEE Transactions on ITS*, vol. 16, no. 2, pp. 653–662, April 2015.

[3] 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 ITS*, vol. 13, no. 3, pp. 1277–1286, Sept 2012.

[4] X. Pan, C. S. Han, K. Dauber, and K. H. Law, "A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations," *Ai & Society*, vol. 22, no. 2, pp. 113–132, 2007.

[5] S. Liu, S. Lo, J. Ma, and W. Wang, "An agent-based microscopic pedestrian flow simulation model for pedestrian traffic problems," *IEEE Transactions on ITS*, vol. 15, no. 3, pp. 992–1001, June 2014.

[6] C. Sommer, R. German, and F. Dressler, "Bidirectionally Coupled Network and Road Traffic Simulation for Improved IVC Analysis," *IEEE Transactions on Mobile Computing*, vol. 10, no. 1, pp. 3–15, January 2011.

[7] M. 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 ITS*, vol. 16, no. 3, pp. 1136–1147, June 2015.

[8] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz, "Sumo - simulation of urban mobility: An overview," in *SIMUL*, Barcelona, Spain, 2011.

[9] T.-Y. Hu, L.-W. Chen, P.-H. Hung, Y.-K. Huang, and M.-L. Chiang, "An object-oriented simulation-assignment model with consideration of mixed traffic flows for its applications," in *ITSC*, Sept 2006, pp. 1483–1488.

[10] D. Krajzewicz, "Summary on publications citing sumo, 2002–2012," in *Proceedings of the 1st SUMO Conference–SUMO*, 2013, pp. 11–24.

[11] B. Hendrickson and K. Devine, "Dynamic load balancing in computational mechanics," *Computer Methods in Applied Mechanics and Engineering*, vol. 184, no. 24, pp. 485 – 500, 2000.

[12] K. D. Devine, E. G. Boman, R. T. Heaphy, B. A. Hendrickson, J. D. Teresco, J. Faik, J. E. Flaherty, and L. G. Gervasio, "New challenges in dynamic load balancing," *Applied Numerical Mathematics*, vol. 52, no. 23, pp. 133 – 152, 2005.

[13] T. Zou, G. Wang, M. V. Salles, D. Bindel, A. Demers, J. Gehrke, and W. White, "Making time-stepped applications tick in the cloud," in *SoCC*, 2011, pp. 1–14.

[14] Q. Bragard, A. Ventresque, and L. Murphy, "dsumo: Towards a distributed sumo," in *First SUMO conference*, Berlin, Germany, 2013.

[15] T. Suzumura and H. Kanezashi, "Highly scalable x10-based agent simulation platform and its application to large-scale traffic simulation," in *DS-RT*, 2012, pp. 243–250.

[16] 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*. Citeseer, 2003.

[17] D. Šišlák, P. Volf, M. Jakob, and M. Pěchouček, "Distributed platform for large-scale agent-based simulations," in *Agents for Games and Simulations*, 2009, pp. 16–32.

[18] M. Prandini, L. Piroddi, S. Puechmorel, and S. Brazdilova, "Toward air traffic complexity assessment in new generation air traffic management systems," *IEEE Transactions on ITS*, vol. 12, no. 3, pp. 809–818, Sept 2011.

[19] D.-H. Lee and P. Chandrasekar, "A framework for parallel traffic simulation using multiple instancing of a simulation program," *ITS Journal*, vol. 7, no. 3-4, pp. 279–294, 2002.

[20] G. D. Cameron and G. I. Duncan, "Paramics - parallel microscopic simulation of road traffic," *The Journal of Supercomputing*, vol. 10, no. 1, pp. 25–53, 1996.

[21] 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," *25th Aniversary of CRT*, 1998.

[22] K.-H. Kastner, R. Keber, P. Pau, and M. Samal, "Real-time traffic conditions with sumo for its austria west," in *Simulation of Urban Mobility*, 2013, pp. 146–159.

[23] K. Nagel and M. Rickert, "Parallel implementation of the transims micro-simulation," *Parallel Computing*, vol. 27, no. 12, pp. 1611 – 1639, 2001.

[24] J. Härri, P. Cataldi, D. Krajzewicz, R. J. Blokpoel, Y. Lopez, J. Leguay, C. Bonnet, and L. Bieker, "Modeling and simulating its applications with itetris," in *PM2HW2N*.

[25] "Citymobil: Toward advanced road transport for the urban environment," http://www.citymobil-project.eu/.

[26] R. M. Fujimoto, "Parallel and distributed simulation," in *WSC*. ACM, 1999, pp. 122–131.

[27] Q. Bragard, A. Ventresque, and L. Murphy, "Synchronisation for Dynamic Load Balancing of Decentralised Conservative Distributed Simulation," in *PADS*, 2014.

[28] ——, "Global dynamic load-balancing for decentralised distributed simulation," in *WSC*, 2014, pp. 3797–3808.

[29] A. Ventresque, Q. Bragard, E. S. Liu, D. Nowak, L. Murphy, G. Theodoropoulos, and J. Q. Liu, "Spartsim: A space partitioning guided by road network for distributed traffic simulations," in *DS-RT*, Dublin, Ireland, 2012, pp. 202–209.

[30] S. Uppoor and M. Fiore, "Large-scale urban vehicular mobility for networking research," in *VNC*. IEEE, 2011, pp. 62–69.

[31] A. Steed and R. Abou-Haidar, "Partitioning Crowded Virtual Environments," in *VRST*, 2003, pp. 7–14.

[32] G. Karypis and V. Kumar, "Metis - unstructured graph partitioning and sparse matrix ordering system, version 2.0," Tech. Rep., 1995.

AUTHOR BIOGRAPHIES

QUENTIN BRAGARD is a PhD student in the Performance Engineering Laboratory at University College Dublin, Ireland, under the supervision of Liam Murphy. His main research topics are Road Traffic Partitioning and Distributed Simulation. His email address for these proceedings is quentin.bragard@ucdconnect.ie.

ANTHONY VENTRESQUE, PhD, is a Lecturer in the School of Computer Science at University College Dublin, Ireland. He received BSc (2002), MSc (2004) and PhD (2008) in Computer Science from the University of Nantes, France. His email address for these proceedings is anthony.ventresque@ucd.ie.

LIAM MURPHY received a B.E. in Electrical Engineering from University College Dublin in 1985, and an M.Sc. and Ph.D. in Electrical Engineering and Computer Sciences from the University of California, Berkeley in 1988 and 1992 respectively. He is currently a Professor in Computer Science at University College Dublin, where he is Director of the Performance Engineering Laboratory. His email address for these proceedings is liam.murphy@ucd.ie.