Proceedings of the 2014 Winter Simulation Conference
A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds.

EFFICIENT GRAPH-BASED DYNAMIC LOAD-BALANCING FOR PARALLEL LARGE-SCALE AGENT-BASED TRAFFIC SIMULATION

Yadong Xu
Wentong Cai

School of Computer Engineering
Nanyang Technological University
Singapore, 639798, SINGAPORE

Heiko Aydt

TUM CREATE Ltd.
1 CREATE Way
Singapore, 138602, SINGAPORE

Michael Lees

Informatics Institute
University of Amsterdam
Science Park 904
Amsterdam, 1098 XH, THE NETHERLANDS

ABSTRACT

One of the issues of parallelizing large-scale agent-based traffic simulations is partitioning and load-balancing. Traffic simulations are dynamic applications where the distribution of workload in the spatial domain constantly changes. Dynamic load-balancing at run-time has shown better efficiency than static partitioning in many studies. However, existing work has only focused on geographic partitioning methods which do not consider the minimization of communication overhead. In this paper, a graph-based dynamic load-balancing mechanism which minimizes the communication overhead during load-balancing operations is developed. Its efficiency is investigated in the agent-based traffic simulator SEMSim Traffic using real world traffic data. Experiment results show that it has significantly better performance than static graph partitioning methods in improving the overall speed of the simulation.

1 INTRODUCTION

The modeling and simulation of road traffic has been utilized as a useful tool to study road traffic for more than half a century. The modeling and simulation of road traffic can be approached in various ways, depending on the level of abstraction necessary. They are commonly known as macroscopic (Lighthill and Whitham 1955, Richards 1956), mesoscopic (Paveri-Fontana 1975), microscopic (Gipps 1981), and nanoscopic (a.k.a., sub-microscopic) (Ni 2003). Road traffic is a complex system whose behavior is difficult to predict. The behaviors of constituent components have an impact on the whole system. The behaviors of individual components are modeled at the microscopic and nanoscopic levels of detail. Both microscopic and nanoscopic simulations can be conducted in an agent-based approach. Large-scale agent-based traffic simulation is a promising method for studying road traffic and solving problems, for instance, the influence of risky driving and the influence of adopting electric vehicles or automatic vehicles in the transportation system. SEMSim Traffic is a nanoscopic traffic simulator that is able to capture this level of detail (Xu, Aydt, and Lees 2012). It is designed to study how different vehicle designs and different infrastructures will influence the transportation system when introducing electric vehicles at a large-scale in mega-cities like Singapore. However, conducting such a simulation at large-scale, e.g., the whole city, requires high

computational resource. Certain computing techniques should be deployed to make large-scale agent-based traffic simulations computationally feasible.

To harness more computational resource, parallel computing can be used. There are many critical aspects of considerations when building an efficient parallel simulation, for example, clock synchronization (Fujimoto 1990), partitioning and load balancing (Xu and Lau 1997), and interest management (Lees, Logan, and Minson 2005). The problem of partitioning and load-balancing is addressed in this paper.

Considering the traffic pattern in the real world, people usually drive to work in the morning and drive home in the evening, which creates different traffic flows during the day, as well as peak traffic hours. Corresponding to this real world scenario, in traffic simulations agents always move in the road network and they are dynamically created and removed during the simulation. This leads to constant workload changes at different areas of the simulation space and necessitates a properly designed partitioning and load-balancing mechanism.

The ultimate optimization objective of a partitioning method is to minimize the overall execution time. To achieve this objective, two sub-objectives should be considered during the partitioning. The inter-process communication overhead should be minimized, and the workload imbalance should be minimized (Xu, Aydt, and Lees 2012). Partitioning of a parallel simulation can be performed at the beginning of the simulation or dynamically at run-time. When a dynamic load-balancing approach is adopted, the overhead of repartitioning should also be minimized. Some work has shown that dynamic load-balancing is able to balance the workload of processes according to run-time information and has shown better efficiency in parallel microscopic traffic simulations (Igbe 2010, Sun, Chen, Li, and Wang 2012). Dynamic load-balancing can be achieved by refining existing partitions incrementally, or by invoking a global partitioning algorithm to generate totally new partitions. Partitioning algorithms used in traffic simulations generally fall into the following four categories: geographic partitioning, graph partitioning, hypergraph partitioning, and scatter partitioning. Among them, only graph partitioning methods consider the minimization of communication overhead (hypergraph is a generalization of graph). Graph partitioning has demonstrated much better static partitioning results compared to geographic partitioning algorithms (Nagel and Rickert 2001). However, graph partitioning is more computationally expensive which may hinder its application on dynamic load-balancing. Existing work on dynamic load-balancing of traffic simulations has only been the geographic partitioning algorithms, which does not consider the minimization of communication overhead.

In this paper, we have developed a new dynamic load-balancing mechanism for parallel agent-based traffic simulations that balances workload efficiently and minimizes the communication overhead at the same time. It uses run-time traffic volume and traffic flow information to estimate the workload and communication overhead respectively. A threshold of workload imbalance is used for the invocation of load-balancing operations. Initial partitioning is performed using a graph partitioning algorithm (Karypis and Kumar 1998). Repartitioning is achieved by invoking the graph partitioning algorithm globally to generate new partitions and mapping the new partitions to the original ones. The dynamic load-balancing mechanism is experimented in the traffic simulator SEMSim Traffic. SEMSim Traffic composes a number of components, one of which is a parallel discrete-event simulation engine developed for distributed memory systems. The road network of the whole Singapore, and real world data on the trip distribution are used in the scenario. The results showed that our dynamic load-balancing mechanism has improved the speed-up of the parallel simulation significantly compared to static graph partitioning methods.

The remainder of the paper is organized as follows: Sections 2 presents the exiting work on the partitioning and dynamic load-balancing of microscopic traffic simulations. Section 3 introduces briefly the simulation space and agent models, how the simulation space is partitioned, and how inter-process communication is conducted. Section 4 describes our dynamic load-balancing mechanism based on graph partitioning. After that, experiments and results are presented in Section 5. Finally, conclusions and future work are presented in Section 6.

2 RELATED WORK

2.1 Partitioning Methods

The partitioning methods in agent-based traffic simulations have largely used domain decomposition as apposed to functional decomposition. The target of partitioning is the road network. There are generally four categories of partitioning methods used: geographic partitioning, graph partitioning, hypergraph partitioning, and scatter partitioning.

Geographic Partitioning uses the geographic information of the road network. The most straight forward way is to cut the simulation space into strips or grids of equal sizes (Klefstad, Zhang, and Lai 2005, Lee 2002). This method is easy to implement and manage, however, the workload distribution is very unlikely to be even. Another geographic partitioning is known as Recursive Coordinate Bisection (RCB) (Berger and Bokhari 1987, Wei, Chen, and Sun 2010). The simulation space is cut into two equal-sized sub-regions by a plane orthogonal to coordinate axes and this procedure is recursively applied until the desired number of sub-regions are reached. This approach generates more balanced load distribution compared to grid partitioning. For geographic partitioning methods, repartitioning can be easily done by sweeping the boundary lines. Dynamic geographic repartitioning has demonstrated better performance than their static counterparts (Zhang, Jiang, and Li 2007, Igbe 2010, Sun, Chen, Li, and Wang 2012).

Graph Partitioning does not rely on geographic information but the connectivity of the road network. The workload partitioning problem is formulated as as a graph partitioning problem by converting the road network into a graph and using the amount of workload as the weight of a vertex and the data volume between vertices as the weight of a edge. Graph partitioning algorithms consider both equalizing the weights between partitions and minimizing the edge cuts (Kernighan and Lin 1970, Karypis and Kumar 1998, Hendrickson and Kolda 2000). Thus, they usually generate better partitions than the geographic partitioning methods (Nagel and Rickert 2001, Suzumura and Kanezashi 2012). The disadvantage is that graph partitioning methods are more computationally expensive than geographic partitioning methods and a repartitioning does not guarantee new partitions to have similar shapes as the previous ones. Dynamic graph repartitioning can be achieved in two ways: scratch-remap and diffusive (a.k.a., incremental refinement) (Schloegel, Karypis, and Kumar 2001). *Scratch-remap* method invokes certain graph partitioning algorithms to generate new partitions from scratch, and then map the new partitions to the original partitions to reduce the data redistribution overhead. *Diffusive* methods use existing partitions and refine them incrementally or diffusively to achieve load balance. Scratch-remap method is generally able to generate better edge cuts but incurs higher data redistribution overhead compared to diffusive methods. Dynamic graph repartitioning has not been applied in traffic simulation.

Hypergraph Partitioning is a generalization of graph partitioning (Catalyurek and Aykanat 1999). A *hypergraph* is a generalisation of a graph in which an hyperedge can connect more than two vertices. A hyperedge can express the mutual data dependency among all vertices it connects. It is useful for scenarios where broadcasting of information plays an important role (Xu and Tan 2012). The disadvantage is a much higher computational cost of the algorithm, which may limit its use to offline partitioning only.

Scatter Partitioning is a method that distributes the workload nearby to different partitions. It usually results in a highly balanced workload distribution (Barceló, Ferrer, and Garcia 1998, Thulasidasan, Kasiviswanathan, Eidenbenz, and Romero 2010). However, the partitioning method incurs high communication overhead. It can be used in environments where high volume of communication does not incur significant overhead. This method might not be efficient in a distributed memory environment where the communication is nontrivial.

2.2 Dynamic Load-Balancing In Traffic Simulation

Dynamic load-balancing methods generally involve a number of steps: load measurement (load index), load information exchange, invocation strategy (whether a load-balancing operation is necessary), and load-balancing operation. The same applies to traffic simulation.

The most straight forward way of load measurement in agent-based traffic simulation is by counting the number of vehicles or agents (Zhang, Jiang, and Li 2007, Sun, Chen, Li, and Wang 2012). A more sophisticated way is to consider the average execution time of the agents, which takes into account both the load of the application and the possible influence of hardware utilization (Igbe 2010). Load measurement is usually performed periodically at run-time in a distributed manner. The subsequent three steps can all be conducted in a *distributed* way where all processes make individual decisions, or a *centralized* way where a master process makes all decisions. The load information exchange is also performed periodically. The exchange happens either between direct neighbors (Igbe 2010, Sun, Chen, Li, and Wang 2012) or between a master process and slave processes (Igbe 2010). After the load exchange, an invocation strategy analyzes the current workload distribution and decides whether a load-balancing operation should be performed. A load-balancing operation can be initiated either by the processes with heavy or light workload, or by a master process which oversees the global load distribution in a centralized way (Igbe 2010). As for the invocation criteria, a fixed imbalance threshold was used in some work (Sun, Chen, Li, and Wang 2012, Zhang, Jiang, and Li 2007). The value of the threshold is critical for the performance. With a small threshold, load-balancing operation is performed more frequently, thus results in better load balance; however, a worse overall speed up could be obtained as the result of the higher communication overhead (Sun, Chen, Li, and Wang 2012). To make an informed invocation decision, a profitability estimation can be performed to ensure that the gain of performing a load-balancing operation is greater than the cost and prevent non-beneficial operations (Igbe 2010). When strip and grid partitioning methods are used, dynamic load-balancing can be achieved easily by incrementally sweeping the boundary lines from heavily loaded simulation processes to the lightly loaded ones (Igbe 2010, Zhang, Jiang, and Li 2007, Sun, Chen, Li, and Wang 2012). However, these methods do not consider the communication overhead, which may render it non-beneficial for real world road networks that have irregular shapes. To the best of our knowledge, there has not been work demonstrating the effectiveness of dynamic load-balancing mechanism based on graph-partitioning in agent-based traffic simulations. Although graph partitioning is more computationally expensive, it considers the load imbalance as well as the communication cost.

3 PARTITION THE SIMULATION SPACE

3.1 Spatial Network and Agent Models

The simulation space of our agent-based traffic simulation is a road network. In a road network, the links of the network are containers/placeholders of agents. If links have multiples lanes, lanes are containers of agents. Nodes contain the connectivity information of links and lanes. The road network is a *spatial network* which is different from the network in network simulations where the nodes usually represent agents and the links represent the relationships or dependencies between agents. An illustration of the spatial network is shown in Figure 1. The dashed circles are network nodes, and the connections between them are network links. For simplicity, lanes are not shown, therefore, links are containers of agents.

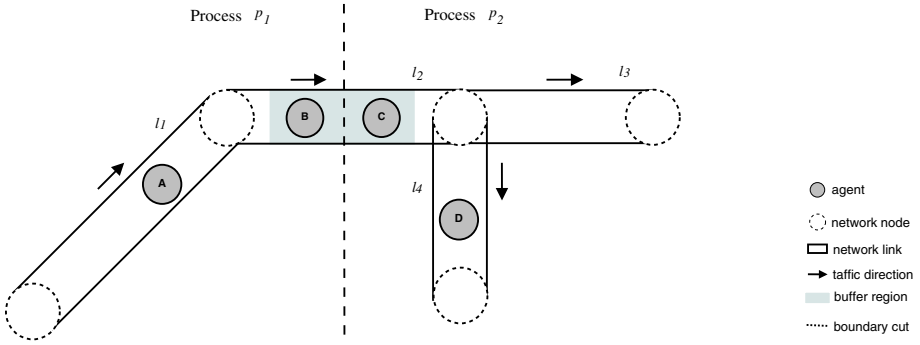


Figure 1: Partitioned spatial network container with buffered regions.

The agent in the simulation is a driver-vehicle-unit that contains driver behavior models, e.g., the acceleration model and the lane-changing model, and vehicle component models, e.g, the motor model and the battery model for electric vehicles. An agent has a *sensing range* which is the area around the agent within which the states of other agents have an effect on the agent's behavior. The sensing range is omnidirectional in the road network, even though the traveling direction of the agent is unidirectional, since the agent needs to examine the area in front for deciding appropriate accelerations, and the area both in front and behind for deciding safe lane-changes. Agents continuously scan the surrounding environment within the sensing range and perform certain behaviors during the simulation.

3.2 Agent Migration and Shared State Update

The partitioning of the simulation is performed on the spatial network. The spatial network is cut through *boundary links*. Boundary links are the links spanning two partitions. A boundary link is evenly divided between two partition, as depicted in Figure 1. Link l_2 is a boundary link. The left half belongs to process p_1, and the right half belongs to process p_2.

In the simulation, *migration* of an agent happens when the agent moves beyond the boundary of a partition and enters the domain of another partition. For example, in Figure 1 suppose agent B continues moving on link l_2 and exceeds the boundary of process p_1, it will be migrated to process p_2. Agents require the information of surrounding agents for certain behaviors. The states of the agents that affect neighboring agents in other processes are referred as *shared states*. They should be sent over to the neighboring processes and kept updated. A *buffered region* refers to the region within which the shared states of agents should be copied to another process and kept updated. Buffered regions are the areas directly next to the boundary cut on a boundary link. The size of buffer regions is determined by the sensing range of agents. In Figure 1, the shaded area on link l_2 is the buffered region. Agents B and C need to know the states of each other, therefore, shared states of agent B is sent to process p_2 and those of agent C to process p_1. *Non-local proxy* agents are created in the receiving processes which represent the actual agents in the sending process. For example, there is a proxy of B in process p_2 and a proxy of C in process p_1.

To reduce the communication overhead, the message size should be minimized. Since different information is required during migration and shared state update, the content to be transferred should be defined respectively. During migration it is necessary to transfer the complete state of the agent, which can lead to large message sizes. In contrast, during the update of shared states, only shared states (i.e., states that have influence on the behavior of surrounding agents) are necessary. Note that non-local proxy agents only contain shared-state information. To further reduce the overhead, creation and deletion of agents and their non-local proxies during agent migration is also optimized. When an agent enters the buffered region, a non-local proxy will be created in the neighboring process. After the agent is migrated to the neighboring process, the non-local proxy will be used to create the migrated agent. Since the migrated agent is still in the buffered region immediately after the migration, its copy in the original process is not removed. It is marked as the non-local proxy in the original process and deleted only after the migrated agent moves out the buffered region.

3.3 Barrier Synchronization and Message Passing

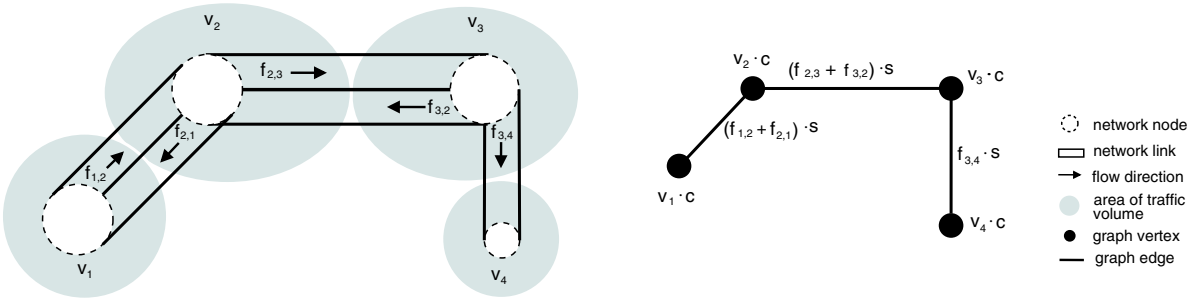
In the parallel simulation engine, events are scheduled by driver behavior models and vehicle models of the simulation. The main computational components are currently the driving behavior models, i.e., the acceleration and lane-changing models. Due to the characteristics of the driving behavior models, events scheduled by them are periodic with a fixed interval which is equivalent to a time-stepped execution. The necessity of migrating agents and updating shared states is checked during the simulation, and messages are sent only if necessary. Due to the time-stepped fashion of executing the agent models, the checking is performed at the end of each time step. Processes scan the boundaries of their partitions to determine the agents to be migrated and the shared states to be sent over. Then all the information to be sent is

wrapped in a single message and sent to the corresponding neighboring processes. This also serves as a global barrier that synchronizes the simulation time of the processes.

4 GRAPH-BASED DYNAMIC LOAD-BALANCING MECHANISM

4.1 Load Index and Communication Index

The spatial network needs to be abstracted as a weighted partitioning graph. One node of the spatial network corresponds to one vertex of the weighted graph. All links that connect two nodes in the spatial network are combined into one undirected edge that connects the two corresponding vertices in the weighted graph. An example is shown in Figure 2. A proper workload index and a communication index are required to assign appropriate weights to the graph. A *load index* measures the weight of vertices with workload, and a *communication index* measures the weight of edges with run-time communication overhead. In Figure 2, $f_{i,j}$ denotes the traffic flow from node i to node j, and v_i denotes the traffic volume at the neighborhood of node i. *Traffic flow* is the number of agents that travel through the link in a unit of time. *Traffic volume* is the number of agents in the neighborhood of a node at a time instant. The neighborhood of a node refers to the halves of the connecting links near the node (depicted as shadowed areas in Figure 2(a)).



(a) spatial network (b) partitioning graph

Figure 2: Using the traffic flow and traffic volume information on the spatial network to calculate the weights of the partitioning graph.

Suppose the traffic volume around node i is v_i and the average computational load per agent is c in terms of CPU time, then the load index of node i is $v_i \cdot c$. It is the computation time of all the agents in the neighborhood of node i. Suppose the overall traffic flow between node i and node j is $f_{i,j} + f_{j,i}$ and the average data size per agent migration is s, then the communication index is denoted as $(f_{i,j} + f_{j,i}) \cdot s$. Traffic flow is used as one of the factors for communication index because it measures the rate of agents passing through the link which corresponds to the rate of migrating agents if the link is a boundary link. In the simulations where agents have different computation times and data sizes, c and s may be different for different nodes and links. In the simulations where agents do not have significant computation time or data size differences, both t and s can be normalized to 1. In this case, the load index can be expressed as v_i, and the communication index can be expressed as $f_{i,j} + f_{j,i}$.

The dynamic load-balancing mechanism uses a master-slave approach. The workload is measured by all processes periodically. Then the slave processes send the workload information to a master process. The master process makes decisions on the invocation and executes the graph partitioning algorithm.

4.2 Invocation Strategy Analysis

The execution time of one time step of the simulation composes of a number of elements: the time for model calculation, the waiting time at a global barrier due to uneven workload distribution, and the time for message passing. Suppose the simulation has a set of I processes with ids $\{0, 1, ...I-1\}$, at one certain time step the total workload in terms of CPU time is t_{cmp}, of which the process i shares a portion t_{cmp_i}, the

waiting time of process i is t_{wait_i}, the message passing time of process i is t_{comm_i}, the extra time is t_{extra_i}, the execution time of one time step is t_i, then

$$t_i = t_{cmp_i} + t_{wait_i} + t_{comm_i} \tag{1}$$

Since there is a global barrier at the end of the time step, t_i is equal for all I processes. Because the processes that have less workload always wait for the one that has the most workload, we can deduce that $t_{cmp_i} + t_{wait_i} = Max(T_{cmp})$, where $T_{cmp} = \{t_{cmp_0}, t_{cmp_1}, ...t_{cmp_{I-1}}\}$ is the set of computation time of all processes. We can express the time spent on one time step for all processes as

$$t = Max(T_{cmp}) + t_{comm} \tag{2}$$

assuming that collective communication is done at the end of each step, which also serves as a barrier synchronization, so t_{comm_i} are the same for all processes.

If the workload is close to perfectly balanced, then $t_{wait_i} \approx 0$ and all processes have approximately the average amount of workload $t_{cmp_i} \approx Avg(T_{cmp})$, thus, $Max(T_{cmp}) \approx Avg(T_{cmp})$. However, in practice, it is not possible to achieve perfect balance at every time step. If a static partitioning method is used, it is unlikely that the workload distribution is optimized. For dynamic load-balancing, the balancing operation should not be performed at every time step considering the extra overhead it introduces. Therefore, an invocation strategy should be developed to perform load-balancing operations at the suitable time. Intuitively, a load-balancing operation is required when the workload reaches a very imbalanced state. We formulate the load imbalance at a time step as

$$L_{imb} = Max(T_{cmp}) - Avg(T_{cmp}) \tag{3}$$

which is measured with time. A load-balancing operation is invoked when L_{imb} is greater than a threshold L_{thres}. In addition, in an environment where agents are homogeneous which means they have approximately the same computational cost on average, the calculation of imbalance and threshold can be simplified by measuring the number of agents instead of time.

4.3 Repartitioning and Mapping

If a load-balancing operation is beneficial, a graph partitioning algorithm is executed by the master process. The graph partitioning algorithm utilized in our simulation is a multilevel k-way partitioning scheme (Karypis and Kumar 1998). The partitioning algorithm contains three main steps: graph coarsening, partitioning the coarse graph, and graph uncoarsening. After that a refinement step is performed. It is one of the fastest and most efficient graph partitioning algorithms.

New partitions generated may not have similar shapes as existing ones, therefore mapping is performed to map the new partitions to existing partitions to reduce the cost of redistributing the agents. Let I be the set of old partitions, J be the set of new partitions, with a similarity matrix $S = (s_{ij})$, $i = 0, 1, ..., I-1$ *and* $j = 0, 1, ..., J-1$. s_{ij} is the *similarity* between the old partition i and the new partition j. It is quantified with the number of agents that have an old partition id i and a new partition id j. Let P be the set of nodes in the road network. In the mapping process, the similarity matrix is firstly calculated by traversing the set of nodes of the road network. For all i and j, s_{ij} is initialized as 0. If node p has an old partition id i and a new partition id j, and there are v_p agents at the neighborhood of node p, the value of s_{ij} is increased by v_p. This process stops when the whole set of nodes is traversed. Therefore, the calculation of similarity matrix has a complexity of $O(|P|)$. The partitions that have the largest similarity values should be mapped together. To do this, the elements of the matrix S is stored in a queue and sorted by their value in descending order. Then the queue is traversed from the top. If the current item is s_{ij} and partitions i and j both have not been mapped by any other partition yet, map them, otherwise continue traversing to the next item. This process stops when the whole similarity queue is traversed. If there are unmapped partitions after this, which scarcely happens, map them randomly. In this way, the similarity

of new partitions and old partitions is maximized, thus the number of redistributed agents is minimized. The complexity of sorting and traversing the queue is $O(|I|^2)$. Therefore, the overall complexity of this mapping process is $O(|P|) + O(|I|^2)$. After mapping, new partitioning information is broadcast to slave processes. Subsequently, redistribution of agents is performed.

5 EXPERIMENTS AND RESULTS

The experiments in this paper investigate the performance of the graph-based dynamic load-balancing methods compared to static graph partitioning methods. Their performance is analyzed using the workload imbalance, communication overhead, and the overall speed-up. Since the simulation involves stochastic elements, the simulation is run a number of times for each method. Three different strategies are evaluated:

- *Static-nw* Partition only with network information at the beginning of the simulation. The weight of a vertex is the total length of links connecting to the corresponding node. The weight of an edge is the number of lanes inside the corresponding links.
- *Static-pg* Partition with pre-generated traffic flow information. This is the best achievable static partitioning. Considering the fact that a simulation with the same configuration is usually run multiple times, the average traffic flows of links and the average traffic volumes of nodes can be logged during an initial run and used for partitioning in subsequent runs.
- *DLB-th* Load-balancing operations are invoked if the load imbalance exceeds a threshold. The workload imbalance is checked periodically with a certain frequency.

The data used in our experiment are real world data, including the road network and traffic patterns. The experiment scenario is set up as follows: The road network is the network of whole Singapore that consists of 43392 nodes and 84343 links (124589 lanes). The trip distribution is derived from the data of the Household Interview and Travel Survey (HITS). The traffic of 24 hours is simulated. Approximately 1.2 million trips are generated and the number of agents during peak traffic hour of the day is approximately 75,000. The time step size is 0.5 second. The experiments are run on a cluster that is composed of 4 compute nodes each of which has following hardware configurations: 2 $Octa-coreIntel(R)Xeon(R)CPUE5-26700@2.60GHz192GBofRAM$. The compute nodes are connected via 56Gbps InfiniBand.

The agents in our experiment are homogeneous, therefore, we use the number of agents for the load index, load imbalance and threshold. By profiling the computational cost of agent models and the communication cost of message passing, and analyzing the dynamics of workload imbalance in *Static-nw* and *Static-pg* simulation runs, we estimated that the threshold should be at the magnitude of hundreds in terms of number of agents. So in the *DLB-th* strategy, thresholds 100, 500, and 1000 were used.

The first indicator of the effectiveness of the strategies is the workload imbalance of the simulation. The dynamics of the load imbalance of *Static-nw*, *Static-pg* and *DLB-th* methods is shown in Figure 3. Simulation runs with 4 processes are shown. Each plot is the average of multiple simulation runs.

Figure 3 has shown that:

1) There are two peak load imbalance periods in the static partitioning methods. They correspond to the morning and afternoon traffic peak hours, which are from 6am to 9am and from 6pm to 9pm. This is because during traffic peak hours, the total number of agents rapidly increases in the simulation. This leads to much more workload changes than other periods.

2) The *DLB-th* method is able to keep the imbalance relatively much lower, because the load-balancing operations are triggered to balance the workload many times.

3) In the *DLB-th* method, load-balancing operations are performed more frequently during the traffic peak hours, and less frequent during other periods. The workload changes faster and more unevenly during peak hours, therefore, the imbalance exceeds the threshold more frequently.

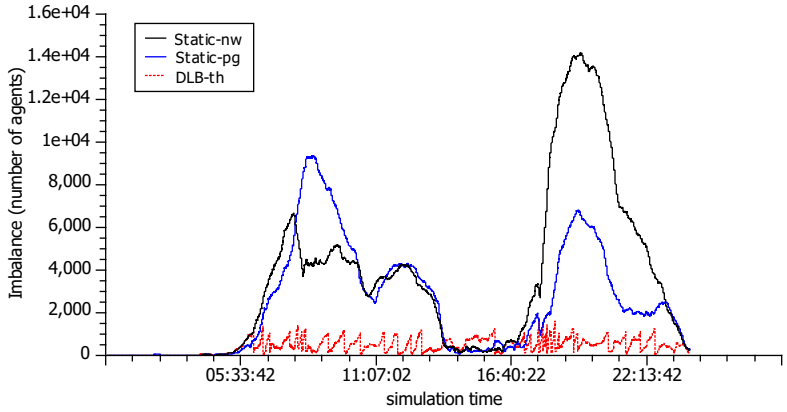


Figure 3: Comparison of workload imbalance of 4 processes throughout the simulation.

The average load imbalance throughout the simulation is shown in Table 1, averages are taken from multiple simulation runs.

Table 1: Average load imbalance per time-step in terms of number of agents (unit=1).

No. of processes	Static-nw	Static-pg	DLB-th-1000	DLB-th-500	DLB-th-100
4	3243.4	2281.6	419.6	229.2	107.3
8	3154.9	2059.4	407.5	251.9	93.6
16	2511.7	1820.3	446.1	225.2	76.2
32	2821.2	1377.8	439.1	226.4	93.3

Table 1 has shown that the *DLB-th* methods achieved better work balance than static methods, and the lower the threshold is, the better workload balance is achieved. This result is expected, since load-balancing operations will improve the load balance, and the lower the threshold, the more frequently load-balance operations are invoked.

The second indicator is the communication overhead. Since all simulation runs have the same duration of simulation time, the total number of migrated agents is used as the indicator of the communication overhead. A larger number of migrated agents indicates larger message sizes and thus more overhead. The numbers of migrated agents of different methods with different number of processes are shown in Table 2. Note that the agents redistributed during load-balancing operations in *DLB-th* methods are not counted.

Table 2: Total number of migrated agents in the whole simulation (unit=million).

No. of processes	Static-nw	Static-pg	DLB-th-1000	DLB-th-500	DLB-th-100
4	0.9	0.8	0.7	0.7	0.7
8	1.6	1.5	1.3	1.2	1.3
16	2.6	2.3	2.1	2.1	2.2
32	3.8	3.5	3.3	3.3	3.5

It can be observed from Table 2 that the number of migrated agents slightly decreased in the *DLB-th* methods. This is because the graph-partitioning algorithm is able to minimize the communication overhead according to run-time information which is not possible for static methods. Additionally, using the pre-generated traffic flow information for static partitioning also reduced the number of migrated agents.

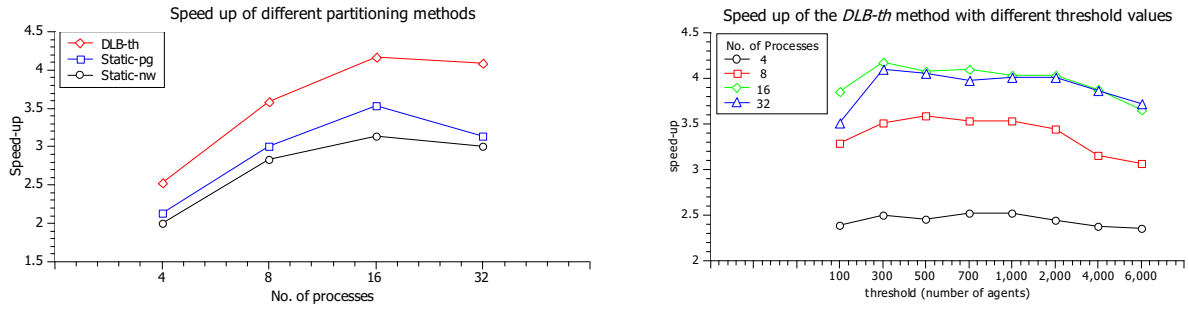
The third indicator is the total extra overhead introduced by the *DLB-th* methods. The average overhead of simulation runs is shown in Table 3.

Table 3: Total extra overhead of dynamic load-balancing operations (unit=second).

No. of processes	DLB-th-100	DLB-500	DLB-th-1000
4	637.5	85.5	38.5
8	599.8	72.7	36.8
16	496.0	55.5	31.3
32	555.1	48.3	24.6

From Table 3, it can be observed that the smaller the threshold is, the higher the load-balancing operation overhead will be. This is because smaller thresholds invoke the load-balancing operation more frequently.

The speed-up of the overall simulation against sequential execution is shown in Figure 4a.



(a) (b)

Figure 4: Comparision of speed-ups of different configurations.

Figure 4a shows that better speed-up is achieved by the *DLB-th* methods than static partitioning methods, which results from better workload balance, less communication overhead, and affordable extra overhead. The maximum speed-up achieved by *DLB-th* methods is 4.2 with 300 being the threshold, by *Static-pg* is 3.54, and by *Static-nw* is 3.14. All maximum speed-up is achieved using 16 processes. Comparing the maximum speed-ups, *DLB-th* has a 18.6 percent improvement over *Static-pg*, and 33.7 percent over *Static-nw*. The thresholds in Figure 4a for 4, 8, 16, and 32 processes are 700, 500, 300, and 300 respectively. This infers that the same threshold may not be the optimal threshold for simulations with different number of processes. The speed-up of the simulation using different thresholds is evaluated which is shown in Figure 4b. It can be observed that the speed-ups are similar when the threshold is at the range from 300 to 1000. However, when the threshold value is too small (e.g., 100), the load-balancing overhead becomes too high thus the speed-up drops. When the threshold value is too large (e.g., 6000), the load-balancing operations are performed infrequently and the load may become imbalance, thus, the speed-up also drops. For example, in the case of using 16 processes, the *DLB-th* method using a threshold value of 6000 has a similar performance as the *Static-pg* method.

From the experiment results above, we can infer that the graph-based dynamic load-balancing is able to improve the overall efficiency of parallel agent-based traffic simulation significantly. The limitation of our approach is that the estimated value of the threshold might not be the optimum value. However, based on our analysis on the threshold above, an approximate value is sufficient to achieve a good speed-up.

6 CONCLUSION AND FUTURE WORK

In this paper, a graph-based dynamic load-balancing mechanism is designed. The mechanism is implemented in the traffic simulator SEMSim Traffic and its effectiveness is evaluated. Experiments have shown that compared to static partitioning methods, the dynamic load-balancing mechanism has improved load balance and the overall performance of the simulation significantly.

There are two pieces of possible future work. Firstly, a diffusive dynamic graph partitioning method could be evaluated. The potential benefit is less number of agents that need to be redistributed during load-balancing operations. Thus, load-balancing operations can be performed more frequently. The disadvantage is that the resultant partitions may not be optimal in terms of balanced load and inter-partition communication. Secondly, the weights of vertices and edges of the partitioning graph is currently calculated using the available traffic flow and traffic volume information. The actual traffic flow and traffic volume after the partitioning may change. Therefore, the partitioning might be improved by using predicted future traffic flow and traffic volume.

ACKNOWLEDGMENTS

This work was financially supported by the Singapore National Research Foundation under its Campus for Research Excellence And Technological Enterprise (CREATE) programme. Michael Lees is also affiliated with ITMO University, St. Petersburg, Russian Federation and his work is partially supported by Russian Scientific Foundation, Project #14-21-00137.

REFERENCES

Barceló, J., J. L. Ferrer, and D. Garcia. 1998. "Microscopic traffic simulation for ATT systems analysis. a parallel computing version". *25th Aniversary of CRT*:1–16.

Berger, M., and S. Bokhari. 1987. "A partitioning strategy for nonuniform problems on multiprocessors". *IEEE Transactions on Computers* C (5): 570–580.

Catalyurek, U., and C. Aykanat. 1999. "Hypergraph-partitioning-based decomposition for parallel sparse-matrix vector multiplication". *IEEE Transactions on Parallel and Distributed Systems* 10 (7): 673–693.

Fujimoto, R. 1990. "Parallel Discrete Event Simulation". *Communications of the ACM* 33 (10): 30–52.

Gipps, P. 1981. "A behavioural car-following model for computer simulation". *Transportation Research Part B: Methodological* 15 (2): 105–111.

Hendrickson, B., and T. G. Kolda. 2000, November. "Graph partitioning models for parallel computing". *Parallel Computing* 26 (12): 1519–1534.

Igbe, D. 2010. *Dynamic load balancing of parallel road traffic simulation*. Phd thesis, University of Westminster.

Karypis, G., and V. Kumar. 1998. "Multilevel k-way Partitioning Scheme for Irregular Graphs". *Journal of Parallel and Distributed Computing* 48:96–129.

Kernighan, B. W., and S. Lin. 1970. "An Efficient Heuristic Procedure for Partitioning Graphs". *The Bell system technical journal* 49 (1): 291–307.

Klefstad, R., Y. Zhang, and M. Lai. 2005. "A distributed, scalable, and synchronized framework for large-scale microscopic traffic simulation". In *Transportation*, 813–818.

Lee, D.-H. 2002, July. "A framework for parallel traffic simulation using multiple instancing of a simulation program". *Journal of Intelligent Transportation Systems* 7 (3): 279–294.

Lees, M., B. Logan, and R. Minson. 2005. "Modelling environments for distributed simulation". In *Environments for Multi-Agent Systems*, 150–167.

Lighthill, M. J., and G. B. Whitham. 1955, May. "On Kinematic Waves. II. A Theory of Traffic Flow on Long Crowded Roads". *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 229 (1178): 317–345.

Nagel, K., and M. Rickert. 2001. "Parallel implementation of the TRANSIMS". *Parallel Computing* 27:1611–1639.

Ni, D. 2003. "2DSIM: a prototype of nanoscopic traffic simulation". In *Intelligent Vehicles Symposium, 2003. Proceedings. IEEE*, 47–52: IEEE.

Paveri-Fontana, S. 1975. "On Boltzmann-like treatments for traffic flow: a critical review of the basic model and an alternative proposal for dilute traffic analysis". *Transportation Research* 9 (4): 225–235.

Richards, P. I. 1956. "Shock Waves on the Highway". *Operations Research* 4 (1): 42–51.

Schloegel, K., G. Karypis, and V. Kumar. 2001. "Wavefront diffusion and LMSR: Algorithms for dynamic repartitioning of adaptive meshes". *IEEE Transactions on Parallel and Distributed Systems* 12 (5): 451–466.

Sun, X., F. Chen, X. Li, and X. Wang. 2012. "An improved dynamic load balancing algorithm for parallel microscopic traffic simulation". In *2012 International Conference on Measurement, Information and Control (MIC)*, 600–604.

Suzumura, T., and H. Kanezashi. 2012, October. "Highly Scalable X10-Based Agent Simulation Platform and Its Application to Large-Scale Traffic Simulation". In *2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, 243–250: IEEE.

Thulasidasan, S., S. P. Kasiviswanathan, S. Eidenbenz, and P. Romero. 2010, May. "Explicit Spatial Scattering for Load Balancing in Conservatively Synchronized Parallel Discrete Event Simulations". In *2010 IEEE Workshop on Principles of Advanced and Distributed Simulation*, Number i, 1–8: IEEE.

Wei, D., F. Chen, and X. Sun. 2010. "An improved road network partition algorithm for parallel microscopic traffic simulation". In *2010 International Conference on Mechanic Automation and Control Engineering (MACE)*, 2777 – 2782.

Xu, C., and F. Lau. 1997. "Load Balancing in Parallel Computers Theory and Practice". Volume 381 of *The Springer International Series in Engineering and Computer Science*, 1–19. Boston: Springer US.

Xu, Y., H. Aydt, and M. Lees. 2012. "SEMSim: A Distributed Architecture for Multi-scale Traffic Simulation". In *26th ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS)*, 178–180.

Xu, Y., and G. Tan. 2012. "hMETIS-Based Offline Road Network Partitioning". *AsiaSim 2012*:221–229.

Zhang, D., C. Jiang, and S. Li. 2007. "Research on dynamic load balancing algorithms for parallel transportation simulations". *Advanced Parallel Processing Technologies*:560–568.

AUTHOR BIOGRAPHIES

YADONG XU is a PhD candidate in the School of Computer Engineering at Nanyang Technological University (NTU), Singapore. His thesis will be centered on parallel and distributed agent-based simulation, and nanoscopic traffic simulation. His email address is xuya0006@e.ntu.edu.sg.

HEIKO AYDT received his Ph.D. degree in Computer Science from Nanyang Technological University (NTU), Singapore and his M.Sc. degree from the Royal Institute of Technology (KTH) in Stockholm. He is currently a Principal Investigator at TUM CREATE. His current research interests are agent-based simulation, complex adaptive systems, symbiotic simulation, evolutionary computing and simulation-based optimisation. His email address is heiko.aydt@tum-create.edu.sg.

WENTONG CAI is a Professor in the School of Computer Engineering at Nanyang Technological University (NTU), Singapore. He is also the Director of the Parallel and Distributed Computing Centre. His expertise is mainly in the areas of Modeling and Simulation (particularly, modeling and simulation of large-scale complex systems, and system support for distributed simulation and virtual environments) and Parallel and Distributed Computing (particularly, Cloud, Grid and Cluster computing). His web page is http://www.ntu.edu.sg/home/aswtcai/ and his email address is aswtcai@ntu.edu.sg.

MICHAEL LEES is an Assistant Professor at the University of Amsterdam in the Section Computational Science. Before this he was an Assistant Professor at Nanyang Technological University (NTU), Singapore. His research interests are primarily in modelling and simulation of large scale complex systems. He is particularly interested in understanding the effect that human behavior has on such systems and the important role that individual behavioral interactions have on system level dynamics. His email address is m.h.lees@uva.nl.