Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

PERFORMANCE OPTIMIZATION FOR AGENT-BASED TRAFFIC SIMULATION BY DYNAMIC AGENT ASSIGNMENT

Hiroki Kanezashi

Graduate School of Information Science and
Engineering, Tokyo Institute of Technology
2-12-1 Oo-okayama
Meguro Tokyo, JAPAN

Toyotaro Suzumura

IBM Thomas J. Watson Research Center
1101 Route 134 Kitchawan Rd,
Yorktown Heights,
New York, USA, 10598

ABSTRACT

It is indispensable to make full use of parallel and distributed systems with increasing demands for large-scale traffic simulation, but problems remain about insufficient scalability due to costs of synchronization by load unbalancing among compute nodes. To tackle this problem, we propose performance optimization method for traffic simulations to underlying road networks preprocessed by graph contraction introducing dynamic re-assignment vehicles and cross points to threads and nodes based on time-series traffic congestion. By applying the optimization and running the simulation of the real-world Dublin city on 16 compute nodes of TSUBAME 2.5, the simulation performance has improved by 4 times with the proposed graph contraction method. We compared the effect of optimizations between the agent assignment method and existing adaptive synchronization method with comparison to regular 1 synchronization per step.

1 INTRODUCTION

In recent years, demands of large-scale traffic simulation are increasing from disaster preventions and city plannings. Such simulations are required to be faster especially about evacuation simulation against sudden disaster. With increasing demands for larger simulations, it is hard for a standalone computer to simulate large-scale social behavior in that all agent objects exceeds memory capacity. Therefore, parallel distributed computation environments are usually used to handle large-scale simulation objects in shorter time by distributing these objects to each computation nodes.

As a means to apply an agent-based simulation program to distributed computation nodes, however, there are some problems related to performance we have to overcome. First, distributions of agents and other objects must be distributed equally to each computation nodes for better speed-up and scalability. Unfortunately, such distribution will change dynamically in traffic and other ITS simulations according as running simulations. For the reason, the scalability will be sometimes worse while conducting such simulations for a long time. Second, communication cost between CPU cores and computation nodes must be considered. Agent objects will have to be distributed to each node in order to they can handle in-memory, but some agents will send data others at the different nodes. The effect will be larger by increasing the number of nodes and processors. Moreover, all nodes must synchronize with each others frequently to keep consistency. When some nodes simulate their own role area very faster and the other do slower, inconsistent outcomes may be generated.

In this research, we executed agent-based traffic simulation in distributed parallel simulation base and showed that the main overheads are load imbalance between nodes and synchronization time. Then we proposed and applied a method to counter these overheads to our simulation base, and evaluated effects of the method. We also discussed the most suitable scale of computation environments with input data.

Our contributions are following.

- We found the primary overhead of agent-based traffic simulation in parallel distributed environment.
- We proposed more effective method for agent-based simulation and implemented it to our agent simulation platform.
- We evaluated the method in several clusters up to 16 computation nodes and found limitations of speed-up by the number of nodes.

The rest of the paper is organized as follows. Section 2 describes our agent-based simulation platform for parallel distributed environment. Section 3 describes an agent traffic simulator we used. In section 4, we explain problem findings about performance overhead. In section 5, we present our proposed method. In section 6, we present evaluation conditions and results of our implementations. Section 7 describes related work. In section 8, we conclude and show future work.

2 XAXIS: AGENT-BASED SIMULATION PLATFORM

As previous work, we developed XAXIS (X10-based Agent eXecutive Infrastructure for Simulation)(Suzumura and Kanezashi 2012, Suzumura and Kanezashi 2013a, Suzumura and Kanezashi 2013b) as agent-based simulation middleware. Simulation applications run on each X10 places independently, and XAXIS supports multi node execution like message exchanges, agent migrations and synchronizations for all places. XAXIS is based on ZASE(Yamamoto, Tai, and Mizuta 2008) simulation base, which can run on a single node. We extended ZASE using X10 so that ZASE applications can be executed distributed parallel computation system to handle larger environment and gain scalability.

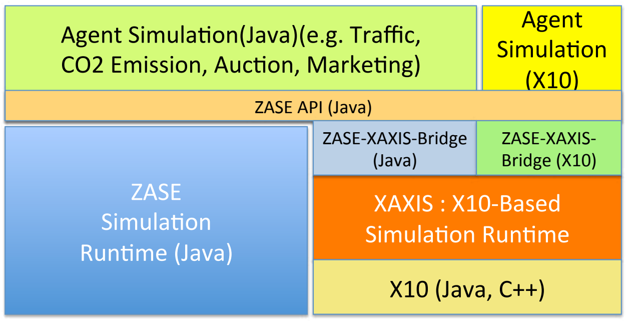
Agent Simulation(Java)(e.g. Traffic, CO2 Emission, Auction, Marketing)
Agent Simulation (X10)
ZASE API (Java)
ZASE Simulation Runtime (Java)
ZASE-XAXIS-Bridge (Java)
ZASE-XAXIS-Bridge (X10)
XAXIS : X10-Based Simulation Runtime
X10 (Java, C++)

Figure 1: XAXIS Software Stack.

XAXIS executes following procedures: handles agents, exchanges messages, and exchanges migrating agents at a simulation step and repeats them. XAXIS entrust these simulation procedures with all X10 places, but in order to keep accuracy, it has some check points where places will stop until rest places reach this point in every simulation steps. Some places will wait until all places handling agents by X10 Team library so that they can send messages and agents to correct destination.

3 MEGAFFIC: IBM MEGA TRAFFIC SIMULATOR

Megaffic (IBM Mega Traffic Simulator)(Osogami, Imamichi, Mizuta, Morimura, Raymond, Suzumura, Takahashi, and Ide 2012) is agent-based traffic simulation application. It can run in the distributed node system on the XAXIS simulation middleware. Megaffic used to run on ZASE API, so we improved methods about agent behavior a little.

In the Megaffic, cross points handles incoming roads to move vehicles on the roads. Moving vehicle method is implemented at Java classes related to cross point, and vehicle repositories is implemented at road class. When vehicles are about to run on the roads, destination cross point of the running road extract vehicles object from road and adjust speeds and positions of the vehicles one after another. If some vehicles reached to a cross point, the cross point calculates the next road and cross point, and then leave the vehicle with the cross point. In this simulation program, each cross point is correspond to Driver Agent at XAXIS, and vehicle is corresponded to Citizen Agent because the road network is always stable but vehicles will be handled different cross points in other places.

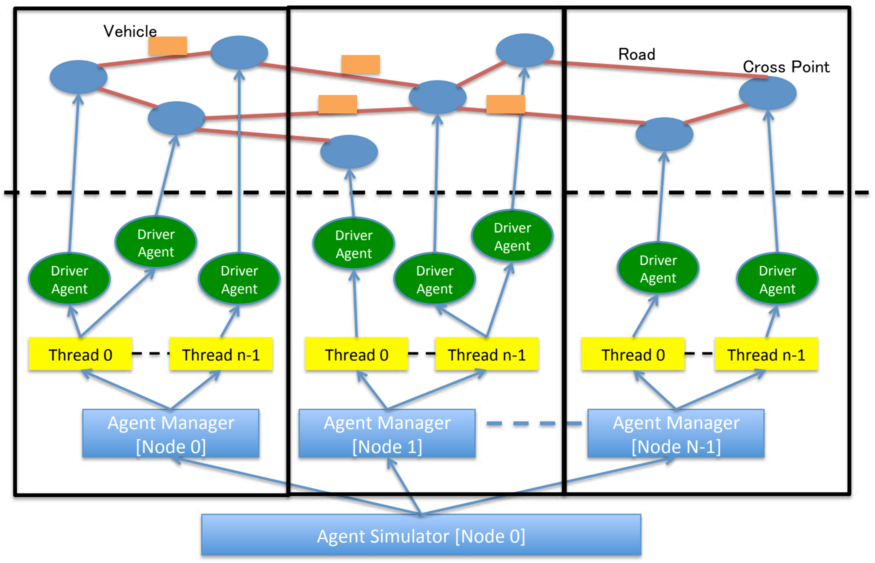

Figure 2: Agent assignment and handling from XAXIS to Megaffic.

In XAXIS simulation platform layer, Agent Simulator will be invoked at first when simulation program is launched. Agent Simulator creates Agent Manager at each computation node (X10 Place), and invoke them to run Driver Agent objects. These Agent Manager objects start several Java threads (X10 Activities) to handle Driver Agent objects. Each thread call run method at Driver Agent, and Driver Agent objects will make Megaffic application objects work.

In Megaffic application layer, all cross points will call method to move vehicles running on inner roads. Actually the Megaffic class of vehicle is treated as subclass of Citizen Agent on XAXIS and managed by Agent Manager in XAXIS, but vehicles will be moved by Driver Agent via cross points and roads.

In traffic simulation at multinode execution, the road network as simulation subject will be divided to sub-graphs to be stored each node. Some vehicles will be handled by a cross point in the other nodes when running Megaffic application at distributed environment. In this situation, the vehicle objects should be sent to the destination nodes. XAXIS supports Citizen Agent migration like message-passing methods. First, the application creates a message object and store the vehicle (subclass of Citizen Agent) to this message. The message is sent from application, and restored as vehicle objects.

4 PRELIMINARY EVALUATION AND PROBLEM FINDINGS

The execution speed should increase as the program use more computation nodes because the number of agents (cross points) each thread or node handles will decrease. However, some problems will happen when agent-based traffic simulation program is executed in parallel distributed systems. Many vehicle objects run in a long distance, and sometimes they will have to reach other cross points in the different nodes. In this situation, vehicle object implemented as Java class must be serialized and deserialized. And all nodes must synchronize each other to keep consistency.

Furthermore, some specified nodes will have to handle many vehicles while the others will handle few of them because vehicles move to them in some simulation scenarios. Even all vehicles are distributed to all nodes equally, congested and clear areas will be separated with simulation running. It causes a lack of scalability because of workload imbalances between nodes and more wait time for synchronizations.

To detect the main performance overheads, we evaluated by preliminary execution with some computation nodes. The result of simulation is Figure 3. MoveVehicles means average elapsed time that each Java thread handles cross point and have vehicles to run. SendMessages means the elapsed time that each vehicle sends messages to neighbor vehicles. Migration means the elapsed time that all computation nodes send vehicle data to other nodes.

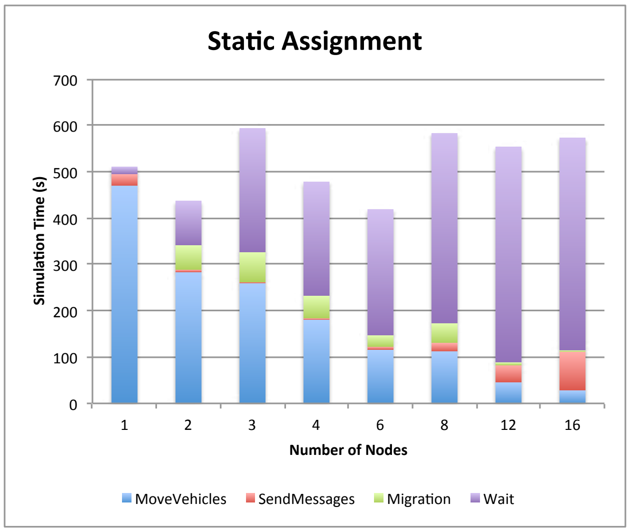

Figure 3: Simulation time details in preliminary execution.

Our simulation program got the best performance at 6 nodes, and there are no performance improvements at more than 8 nodes. Procedure time of cross points' moving vehicles was decreased as increasing the number of nodes and the average time at 16 nodes is about 1/16 of 1 node execution. However, the time is calculated as average, and about 80% of total simulation time is wait time for finishes of other nodes. Communication overheads by migrating vehicles to other nodes are also remarkable in multi-node execution.

5 OPTIMIZATION METHODS

We propose several performance optimization methods: dynamic agent assignment inner nodes (to threads) and assignment across nodes (to threads and nodes). We compare effects of these methods to reducing synchronization frequency method.

5.1 EXTEND SYNCHRONIZATION INTERVAL

At first, we implemented our existing idea (Suzumura and Kanezashi 2013a) to XAXIS.

In the XAXIS simulation base, all computation threads and nodes have to synchronize each other in order to send and receive messages and agents correctly. In this system, however, most nodes will be idle while some nodes take a long time to manage many agents. And what is worse, when all threads and nodes synchronize, each node will have to communicate to the other nodes. Those result lack of scalability.

To reduce these synchronization overheads, we added frameworks to adjust constant synchronization interval to XAXIS. If the interval is set to longer, each node can execute independently and does not need to communicate except sending messages and agents asynchronously.

5.2 DYNAMIC AGENT ASSIGNMENT

We found that the main overhead of agent-based simulation is load imbalances by distribution of agents.

To overcome the load imbalance problem, we propose dynamical Driver Agent assignment methods as one of the optimization method types. One is completed at each node, only re-assign threads to agents. The other is more costly, but also more powerful method re-assigning agents among X10 Places.

5.2.1 DYNAMIC AGENT ASSIGNMENT AT EACH INNER NODE

The first assignment method changes only the mapping between threads and Driver Agents. It is completed in each node, and no vehicles will be migrated during this assignment. In order that the workload will be equal among threads at each place, the mapping what thread will handle cross points in this place regularly. There are no change among road network and vehicles.

In the whole of a simulation, the number of cross points handled by each node is not changed. That means load imbalance among distributed nodes will not be resolved. However, the procedures of re-assignment are very simple and no extra vehicle migration is not necessary.

5.2.2 DYNAMIC AGENT ASSIGNMENT ACROSS NODES

The dynamic agent assignment inner nodes will keep load balance in each node, but load imbalance between nodes is not yet resolved. The reason is that the total number of vehicles in each node may change drastically. To tackle that problem, we also proposed and implemented agent assignment across nodes together with handled vehicles.

In this method, all nodes have whole road network. It will require more heap memory and extra runtime cost, but the usage of memory from cross points and roads is much less than that of vehicles.

The agent assignment procedure is shown at Figure 4.

1. When some cross points should move to other nodes, deactivate them and no threads will handle such cross points.
2. In order to continue moving vehicles, migrate these vehicles to other nodes where these handling cross points will be activate.
3. Activate assigned cross points to threads in the destination nodes.

By re-assigning cross points with vehicles, all vehicles handled by the cross points need to migrate to destination node. It causes other performance overheads for communications and calculations. However, by the communication among all nodes, load imbalance will be resolved in the whole simulation environment.

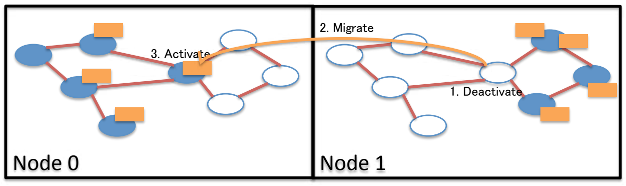

Figure 4: Agent assignment across nodes.

6 EVALUATION

6.1 Input Data Set

For preparing input files for traffic simulation, we obtained Open Street Map (OSM)(Haklay and Weber 2008) XML file and General Transit Feed Specification (GTFS)(Google 2012) files as road network data and person trip file as vehicle trip data. However, Megaffic application can only input specify formatted CSV file, so we had to convert these XML and GTFS text files to the CSV files.

Because curved roads in OSM road network data are represented as many connected short roads, it will be hard to handle whole network as it is with fewer computation nodes. In order to improve efficiency, we connected such roads to one straight one. In this optimization, curved roads are treated as straight ones. However, the affect of the optimization is little because road network structure will never change and the lengths of connected roads are sum of short straight ones. After the contraction, the number of cross points is reduced from 118,856 to 19,062 and the number of roads is from 269,869 to 40,965.

After aggregating networks from input data, we divided the road network into the number of calculation nodes or threads. In default, XAXIS reads a definition mapping file what agents are assigned to each X10 place. We used METIS(Karypis and Kumar 1999) as network partitioning program to generate a mapping file from agent ID to X10. In the default option, METIS divides network into the non-contiguous network. It has option to try to divide each network cluster contiguous, but this option can be applied only when input network is contiguous. We extracted largest Strongly Connected Component (SCC) as input data of METIS to apply ”contig” option. Original OSM network has many isolated cross points and roads, but we removed them because no vehicles will run on these roads and they are not related to traffic simulation.

We prepared 234,378 person trip data about commuting in Dublin city for vehicle routes. Most of trips will go from suburbs to Dublin City Centre. For privacy preserving, we obtained limited data: the location of origin, destination and departure time for each person. Megaffic application needs full path data including all cross points to be passed, so we have to generate trip paths for all people from our generated road network.

In the Megaffic application, one simulation step is corresponded to one second at default setting. In our experiments, simulation steps are set to 14,400, which is corresponded to 4 hours. The start time of trips is based 6:00 AM, so this experiment is expected people commuting simulation from 6:00AM to 10:00AM.

6.2 Environment

We executed simulation program at TSUBAME 2.5 supercomputers Thin-nodes at Tokyo Institute of Technology . The specification data is Table 1. These Thin-nodes have GPU cores, but we used only CPU cores because of XAXIS architecture.

CPU	Intel Xeon X5670 2.93 GHz (6 cores) x 2 (Hyperthreading enabled)
Operating Systems	SUSE Linux Enterprise Server 11 SP3
Memory	PC3-10600 54GB
Network	QDR InfiniBand x 2 (80Gbps)

Table 1: TSUBAME 2.5 Thin-Node Specification

Megaffic application and XAXIS simulation base are running on Java and X10. The version of Java and X10 is following.

The parameters are shown at Table 2.

Number of X10 Places	1, 2, 3, 4, 6, 8, 12, 16
Number of X10 Threads	12
Maximum Java Heap Memory	48GB
Minimum Java Heap Memory	24GB
Java Stack Size	64MB
X10 Version	2.5.1
Java Version	java version ”1.7.0_09” Java(TM) SE Runtime Environment Java HotSpot(TM) 64-Bit Server VM

Table 2: Multi-Node Execution Parameters

In X10, more than one places can be launched in a node. For example, if we launch XAXIS application with setting environment valuable X10_NPLACES to 2 at single node, two Java Virtual Machines will launch at the same node. But we assigned one place for each node in this time because we wanted to use sufficient heap memory.

6.3 Performance with Reduction of Synchronization Frequency

When we reduce synchronization frequency, the simulation time reduced at more than 4 nodes. Especially at 16 nodes it reduced by 35% while there was little effect at less than 3 nodes. However, the best number of nodes was 6 and the simulation time at more than 6 nodes became worse. The main reason is that the number of message objects including vehicle objects sent between physical computation nodes were increased and our method could not cover these message handling overheads.

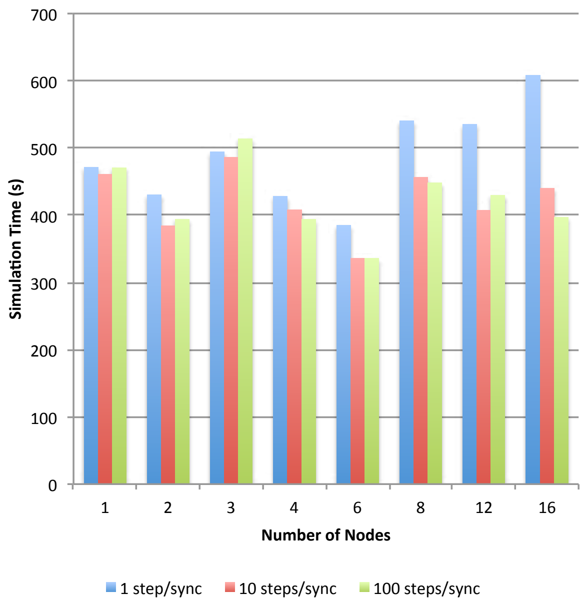

Figure 5: Simulation time by changing synchronization interval.

6.4 Performance with Dynamic Agent Assignment

We performed experiment with each our optimization method, dynamic agent assignments inner and among nodes. The result of reduction of dynamic agent assignment method is Figure 6.

7 RELATED WORK

As our previous work (Suzumura and Kanezashi 2013a), we implemented dynamic adaptive synchronization method to XAXIS and Megaffic. The synchronization overheads are remarkable, so we added a framework to control synchronization according to congestion of road network. However, the origins and destinations for trip of vehicles are selected randomly, so load imbalances were hidden. In this our paper, we used real origin-destination data and found load imbalances as new kind of overhead.

MATSIM(Nagel 2012) is one of the most popular traffic simulator. It is on large-scale agent-based system, and many simulation models can be applied like mixture of public and private vehicles. There are many related researches about MATSIM, most of which are case studies on many cities or model extensions. In (Rashid A. Waraich and Axhausen 2009), the authors tried to gain scalability of MATSIM

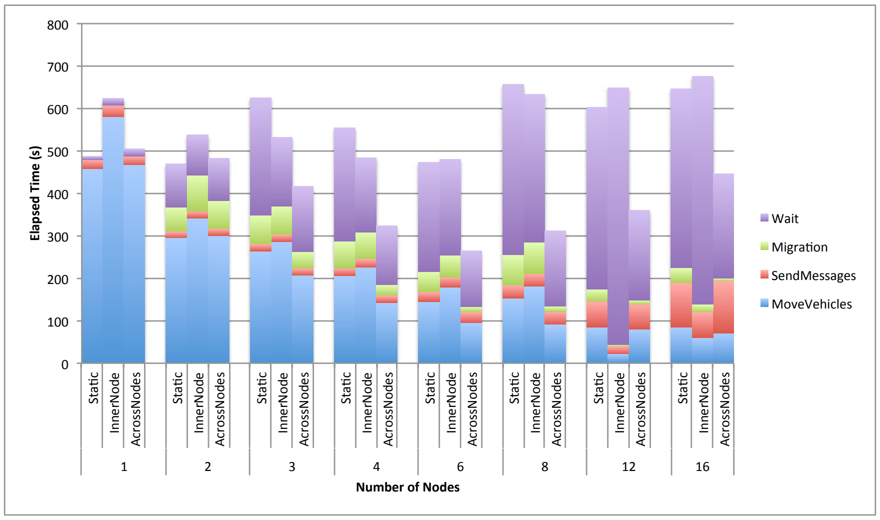

Figure 6: Simulation time with dynamic agent assignment method.

by Java thread. They reached 6 times speed-up using 8 threads. But these evaluation simulations are performed standalone system, not distributed computation nodes.

SUMO(Krajzewicz, Hertkorn, Rssel, and Peter 2002) is also microscopic traffic simulator, implemented in C++. It is said that there is no limitation the volume of input data like network size and the number of vehicles, but not mentioned that it will work on distributed parallel system. dSUMO(Quentin Bragard and Murphy 2013) is an extended version SUMO of distributed feature, and it is evaluated using Dublin road network. The author implemented Container and interfaces such as Server and Clients. However, the authors used only primary roads, and evaluated with up to only 3080 vehicles, and they compared performance up to 2 machines and 4 CPU cores.

8 CONCLUDING REMARKS AND FUTURE WORK

We will have to evaluate our simulation program with these optimization methods for larger road networks and more vehicles. In these experiments, we used whole Dublin city road network and found that the result is the best at 6-node and 72-core execution, and that performance was saturated and become worse at more than 6 nodes. The reason is that the problem size is fixed and these experiments are performed at strong scaling. Using too many computation nodes caused increase of communication costs like vehicle migrations. In order to show these optimization methods are effective, we need further experiments with various road network or trips.

Our optimization methods leave room for improvements. In our current implementation, whole agent assignment rules are decided before running simulation. The same preliminary simulation have to be executed and obtain log files about the number of running vehicles on roads and cross points. It causes waste of simulation time and computation resources, so we will have to re-assign agents automatically while running simulations by monitoring workloads at nodes or threads.

It also needs much more sophisticated improvement on XAXIS simulation base to synchronize only partial calculation nodes at our current simulation base, so we will re-design XAXIS to be available more elastic model as our future work.

Furthermore, it needs additional evaluations with different traffic data sets to verify our proposed methods. Traffic maps of various scales of cities are available from OSM, but it is difficult to obtain public real vehicle trip data because of privacy issues. Our optimization method assumes real traffic situation with unbalanced road congestion, so we have to consider how we can generate trip data from limited traffic information and apply to our simulation program.

ACKNOWLEDGEMENTS

This research was partly supported by the Japan Science and Technology Agency's CREST project.

REFERENCES

Google 2012. "What is GTFS? - Transit - Google Developers".

Haklay, M. M., and P. Weber. 2008, October. "OpenStreetMap: User-Generated Street Maps". *IEEE Pervasive Computing* 7 (4): 12–18.

Karypis, G., and V. Kumar. 1999. "A Fast and Highly Quality Multilevel Scheme for Partitioning Irregular Graphs". *SIAM Journal on Scientific Computing* 20 (1): 359 – 392.

Krajzewicz, D., G. Hertkorn, C. Rssel, and W. Peter. 2002. "SUMO (Simulation of Urban MObility) - an open-source traffic simulation". In *Proceedings of the 4th Middle East Symposium on Simulation and Modelling (MESM2002)*. De Montfort University.

Kai Nagel 2012. "Agent-Based Transport Simulations — MATSIM".

Osogami, T., T. Imamichi, H. Mizuta, T. Morimura, R. Raymond, T. Suzumura, R. Takahashi, and T. Ide. 2012. "IBM Mega Traffic Simulator". Technical report, IBM Research Report, RT0896.

Quentin Bragard, A. V., and L. Murphy. 2013. "dSUMO: Towards a Distributed SUMO". In *The first SUMO User Conference (SUMO2013)*. German Aerospace Center, Institute of Transportation Systems.

Rashid A. Waraich, David Charypar, M. B., and K. W. Axhausen. 2009. "Performance Improvements for Large Scale Traffic Simulation in MATSim". In *9th Swiss Transport Research Conference*. IMP-HSG, IRE, IVT, SVWG/ASET and TRACE.

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

Suzumura, T., and H. Kanezashi. 2013a. "Accelerating Large-Scale Distributed Traffic Simulation with Adaptive Synchronization Method". In *20th ITS World Congress 2013*. US ITS committee.

Suzumura, T., and H. Kanezashi. 2013b. "A Holistic Architecture for Super Real-Time Multiagent Simulation Platform". In *Simulation Conference (WSC), Proceedings of the 2013 Winter*. IEEE.

Yamamoto, G., H. Tai, and H. Mizuta. 2008. "A Platform for Massive Agent-Based Simulation and Its Evaluation". In *Massively Multi-Agent Technology*, edited by N. Jamali, P. Scerri, and T. Sugawara, Volume 5043 of *Lecture Notes in Computer Science*, 1–12. Springer Berlin Heidelberg.

AUTHOR BIOGRAPHIES

HIROKI KANEZASHI is a Ph.D. student at the Graduate School of Information Science and Engineering of the Tokyo Institute of Technology. He graduated from the Tokyo Institute of Technology Computer Science Department in March 2015. He has been researching large-scale data processing in high-performance computing environments. The main research objects is performance optimization of traffic simulation application and agent-based framework in parallel distributed computation nodes. He has also researched precision evaluation of the result of the simulation by open GIS data analysis. His current research interests from Ph.D. course include machine learning, especially deep learning in supercomputers. His email address is kanezashi.h.aa@m.titech.ac.jp.

TOYOTARO SUZUMURA is a research staff member at IBM Thomas J. Watson Research Center in the U.S. as well as a visiting associate professor of University College Dublin in Ireland. He received his Ph.D. in Computer Science from Tokyo Institute of Technology in 2004. He joined IBM Research - Tokyo in 2004 and had involved with several projects such as high performance XML processing, the PHP scripting language, large-scale stream computing, the X10 programming language and so forth. Since 2010, he has started to develop an X10-based agent simulation platform and its application to large-scale traffic simulation. He had served as a visiting associate professor at the Graduate School of Information Science and

Engineering of Tokyo Institute of Technology since April 2009 until October, 2013 and explores research on high performance computing and large-scale graph analytics. In October 2013, he joined the Smarter Cities Technology Center of IBM Research located in Dublin, Ireland and in April 2015 he has joined IBM T.J. Watson Research Center in the U.S as a research staff member. His e-mail address is suzumura@acm.org.