2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications

Highly Scalable X10-based Agent Simulation Platform and its Application to Large-scale Traffic Simulation

Toyotaro Suzumura[1,2,3] and Hiroki Kanezashi [2,3]
[1] IBM Research – Tokyo, [2] Tokyo Institute of Technology , [3] JST CREST

This paper describes highly scalable X10-based agent simulation platform called XAXIS. XAXIS is designed to handle millions or billions of agents on recent highly distributed and parallel computing environments with more than hundreds of CPU cores. To make the runtime scalable on such environments, we need to redesign and implement the simulation middleware. In this paper, we propose the software design, implementation on X10, one of the state-of-the-art PGAS language, and then application to large-scale traffic simulation. By using 192 CPU cores in distributed memory computing environment, the performance scalability is achieved with a traffic simulation.

I. INTRODUCTION

In the era where sensors are deployed for sensing and reflecting the real-world events to digital world, we need an infrastructure for predicting the near future so that we could conduct the better action and better business decision for the next step. Such a prediction can be done in a wide variety of ways such as machine learning on time-series if the relevant entity is not that large. However, the problem that we are tackling is not straightforward in the case of predicting the future at city-wide or social system-wide level. It involves millions of entities have more or less effect on each other and each entity behaves in different ways. The examples include traffic simulation, social network simulation, pandemic infection simulation, CO2 emission simulation, economic simulation, and so forth. For example, the traffic simulation involves more than 1 million vehicles and the traffic flows on roads are dynamically changed by various factors such traffic lights and car accidents, and so forth.

Moreover, social internet medias such as Twitter and/or Facebook have become important information sources to obtain the real-world phenomenon and an industry business shows high interest on how they can use those medias to maximize their investment on the marketing. To make a better decision on how, when, and where they should send the marketing message for those medias, it is better for us to be able to do the simulation before actually sending the marketing messages to understand the marketing effect. Just like the traffic simulation, a huge number of entities, each of who represents individual account in social media, are involve for such understanding.

An agent-based simulation has been known to be a solution that solves these kinds of problems for a while since 1990s. Also, several efforts have been proposed to conduct the large-scale agent simulation, but we have to stand at the stage of designing and implementing new simulation middleware by leveraging the state-of-the-art technologies. Agent-based modeling and simulation (ABMS) is recognized as an effective technology to understand social systems and social phenomenon. ABMS has been applied in many areas to support decision-making and to do research. The scale of agents is expanding more day by day. We need a large-scale agent simulation that can handle millions, billions of agents. Large-scale agent simulation has become a popular way to model and analyze big multi-agent system. Large-scale Agent Simulation technology is developing steadily nowadays due to demands to analyze and model the bigger and more complex system and the advancing computation power.

In this paper, we propose a highly scalable X10-based agent simulation middleware that can leverage the post-peta or exa-scale computing environment in the upcoming 10 years. Our prior art in [25] described the overall description of the middleware, but this paper describes more detailed design and implementation. The technical challenge that we tackle in this paper is as follows:

High Scalability on Modern Distributed Systems

- How can we re-design the software architecture of highly scalable agent based systems towards post-peta or exa-scale computing environment ?
- What programming language would be suitable for building such a system ?
- How can we concurrently process multiple agents in a scalable manner ? For traffic simulation examples, one solution would be to decompose road network and assign each domain to one compute node, and then make multiple agents run in each road network.
- How can we remove obstacles to prevent the node scalability ? Since we are targeting more than hundreds of CPUs, it would not be straightforward to obtain linear scalability without worrying anything.

Seamless Transition from Legacy Agent-based Applications

- What is a to-be software design that allows legacy agent-based applications to run them on new agent simulation platforms ?

1550-6525/12 $26.00 © 2012 IEEE
DOI 10.1109/DS-RT.2012.44

For the above technical challenges, our contributions in this paper are listed as follows:

- We propose a scalable X10-based agent simulation middleware called XAXIS. As far as we know, this is the first approach of leveraging the PGAS language for the agent-based simulation middleware.
- We demonstrate the performance scalability in a large-scale computing environment using 5th ranked super computer named TSUBAME.

The remainder of the paper is as follows. First, the overview of the X10 language as the underlying language of XAXIS is presented in Section II. Section III describes the design and implementation of the XAXIS system. We also describe an application example using a large-scale traffic simulation on XAXIS in Section IV. Section V describes the performance evaluation. Related work is reviewed in Section VI, followed by our conclusions and future directions in Section VII.

II. X10 – Highly Productive Parallel and Distributed Language

Since we use X10 as the underlying programming language of our propose simulation platform, XAXIS, we give you the overview of the X10 language in this section.

X10 [4] is a novel parallel distributed programming language that IBM Research is developing as an open source project. It enables development of highly efficient applications at high throughput in a distributed system in a mixed environment with various models comprising multiple cores, accelerators, etc. In an execution environment equipped with many execution cores, it is important to demonstrate the parallelism and memory configuration to programmers. X10 offers a global address space that is partitioned into multiple places, called Partitioned Global Address Space (PGAS).

A place is the abstraction of the locality of memory, and it typically corresponds to one compute node with more than one CPU cores. Activity can be generated dynamically in each place as an asynchronous execution subject equivalent to a lightweight thread. Activity can be generated using the syntax of an "*async*" sentence, and can be moved to other places using "*at*" sentence. Conventionally, to achieve a simulation using a massive computer with each node comprising multiple cores and being combined in the network, the Message Passing Interface (MPI) is used as a messaging mechanism among nodes. Programming models such as OpenMP and POSIX thread are employed for thread parallel in the 1 node. However, the concept of "*Place*" and "*Activity*" of X10 enables one to build an executive operation system on a massive computer cluster using a unified programming model at high throughput.

X10 supports parallelism by both concurrent and distributed code. A parallel code uses asynchronous un-named activity created by the *async* struct and it is possible to wait for activities to finish with the "*finish*" construct. Distributed code is run over multiple places that do not share memory, therefore objects are deeply copied from one place to another when captured by the "at" struct. Copying is done by first serializing the object into a buffer, sending a buffer to the other place and then de-serializing the buffer at the other place.

Currently X10 compiler is implemented as a translator to other programming languages. X10 can be compiled to Java, C++ environments in Figure 1. X10 environment running on Java is called Managed X10, while X10 on C++ is called Native X10. X10 compiler has two parts: front-end and back-end. The front-end analyze X10 source, check types to create AST (Abstract Syntax Tree). The backend generates code from the AST to Java code or C++ code.

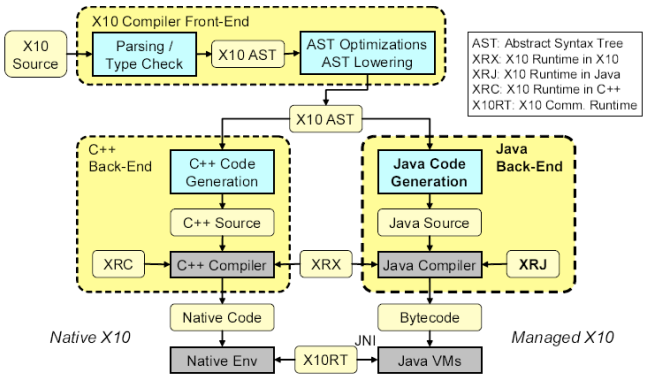

Figure 1 X10 Overview

III. XAXIS: Scalable X10-based Agent Simulation Middleware

XAXIS (X10-based Agent eXecutive Infrastructure for Simulation) is a highly scalable multi-agent simulation middleware that enables application developers to make their application runnable on large-scale environment in a productive manner. The programming model of XAXIS is derived from Java-based agent simulation middleware called ZASE [1], and we also aim at building a middleware that has a compatible API interface with ZASE so that the existing ZASE-based application runs on top of ZASE as well. This section describes the overview of the agent model, its system architecture, and the simulation model of XAXIS.

A. XAXIS Agent Model

ZASE is based on the agent model of [1]. Each agent of ZASE has internal states, handles messages and updates its own states as needed. Messages are sent by a simulator or by other agents. When a message object is delivered to an agent, a callback method of the agent is called. A returned message object will be sent back to the sender. The ZASE framework provides functions to create and to delete agents. Developers can add services to an agent runtime. Agents can look up the

reference to a service and can directly invoke methods of services. A service object also has functions similar to those of agents. Messages are exchanged among agents, simulation runtimes, and agent runtimes. ZASE provides point-to-point messaging and multicast messaging. ZASE also provides a message distribution and aggregation function.

When a simulator sends a multicast message to an agent runtime, the agent runtime will distribute the message to agents. An agent runtime aggregates reply messages from agents into an aggregated message and sends the message to a simulator.

B. *XAXIS System Architecture*

The whole software stack of XAXIS is illustrated in Figure 2. As previously described, we developed Java-based multi agent simulation platform called ZASE [1] and it has a set of APIs ("ZASE API" in the figure) that follows the agent model explained in previous subsection. We also have developed certain amount of applications on top of ZASE such as traffic simulation, auction, marketing, social system, etc.

As one of the design philosophy of XAXIS, the API set of XAXIS should be compatible with ZASE API so that legacy applications runnable with ZASE can also work on top of XAXIS as well without modification. In order to realize this, we have introduced the bridge layer - denoted as the "ZASE-XAXIS-Bridge" layer in the figure - that provides ZASE API. A set of APIs is implemented with X10. The beauty of adopting X10 is that those APIs in XAXIS is implemented in high productive manner without implementing thread management and messaging layer for distributed environment since the X10 language itself includes those functionalities at the language level. The software stack in Figure 2 includes two bridges of "ZASE-XAXIS-Bridge" depending on either Java or X10. Java-based bridge is intended for Java-based simulation on XAXIS. Especially Java is still more commodity language than X10 when implementing the simulation for developers. However, we also provide X10-based bridge (denoted as "ZASE-XAXIS-Bridge(X10) so that the whole software stack is all implemented in X10 and can be compiled into X10 C++ backend with high performance communication libraries such as MPI in the case that simulation developers implement their simulation in X10.

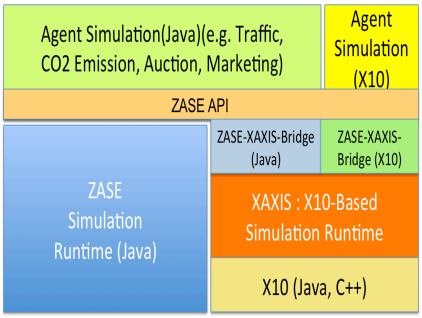

Figure 2 XAXIS Software Stack

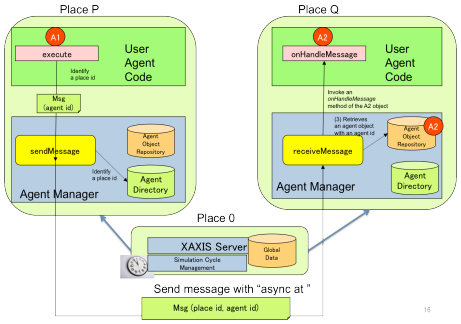

Figure 3 Simulation Model Implementation

C. *XAXIS Simulation Model Implementation with X10*

Figure 3 shows how the simulation model of XAXIS on distributed environment is implemented with X10. An agent manager manages agents asynchronously executed by the activity of X10 in each place in an agent process of a simulation. In case of communication with agents in other places, a place of which the agent manager is in charge is chosen based on an agent's identifier, and a communication message is sent thereto. This communication message is hidden from users, and simulation developers are expected to concentrate only on mounting the logic of the agent. A message that is developed using X10 can be transmitted and received in XAXIS without explicit message communication in a distributed computer, but by calling the method of the agent manager in other places using *at* syntax. A mechanism for communication between places using activity that implements an asynchronous thread and at syntax enables us to mount the simulation execution environment itself easily.

IV. Large-scale Traffic Simulation on XAXIS

In this section we describe how a large-scale traffic simulation is implemented on top of XAXIS.

A. *Large-scale Traffic Simulation*

In the city, there is a demand to optimize the resources of human, cars, and energies. We need a large infrastructure that can meet the demand. The system that reduces wastes and predicts disasters or traffic jam can make a contribution to the whole humanity. In this meaning, IBM Research developed an agent-based traffic simulation modeling

system called IBM Mega Traffic Simulator (Megaffic) [24] to optimize a city traffic system as an important research in recent years.

The up-to-now system has many advantages such as, showing cross point saturation, daily traffic volume. IBM traffic simulation project can be applied to large scale road networks to represent behaviors of vehicles. Megaffic simulation helps to evaluate the efficiency of transport policy, road structure. The system helps administrators to make a policy such as traffic rules, one-way traffic, charges for traffic jam, road user charge, traffic demand management, etc. The system has three parts, mathematical modeling technology, simulation base and visualization base. In mathematical model, the way drivers make a driving behaviors, especially routes choice, is deter-mined by linear function of expected utility which allows to set up the utility weights such as the rates, travel time and distances. Driver behavior model can be expressed in the Gipps [23] model. Route selection rule can be determined by rule types such as minimal cost route, the shortest path algorithm or the time of route selection. Simulating road network of the whole city when use mathematical to model drivers behavior takes much computational cost.

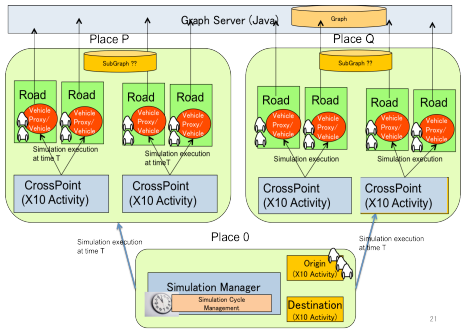

Figure 4 Traffic Simulation

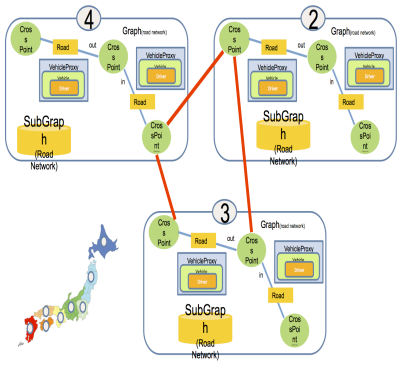

Figure 5 Multi-Node Simulation

B. *Megaffic on XAXIS*

Megaffic runs on XAXIS as shown in Figure 4. A central server at Place 0 invokes an execution method of all the cross points. Each cross points holds a set of incoming and outgoing roads and responsible for moving vehicles on incoming road. If a vehicle reaches up to the cross point, the cross point determines whether it puts the vehicle to the next road of given trips holded by each vehicle if there is sufficient space to make the vehicle enter the road. If the next road is congested, the vehicle waits on the cross point until the sufficient space.

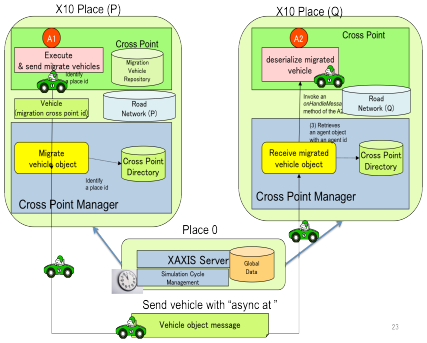

Figure 6 Vehicle Migration

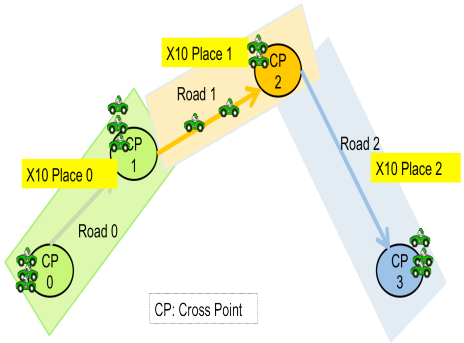

Figure 7 Migration Example

C. *Design for Vehicle Migration Among X10 Places*

Other design point for running Megaffic on XAXIS is to consider how vehicle migration is realized. Figure 6 and Figure 7 illustrates the architecture overview of the migration. First of all, a road object has information on the identifiers of origin and destination cross points. When looking into the cross point identifier, it is possible to know which X10 place the cross point is located since the identifier is assigned by the X10 DistArray construct. A road object also exists at a place where its destination cross point is located. As shown in Figure 7, For instance, if certain trip takes CP1 as origin and CP3 as destination, a graph server returns CP1, CP2, and CP3 as a shortest path. When a vehicle firstly enters into a road, it checks whether a road exists at the same X10 place. If not, a manager migrates the vehicle to the next road exists at different X10 place.

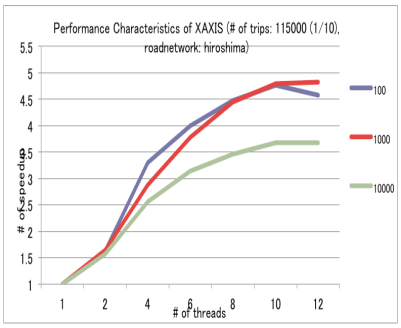

Figure 8 Single Node Performance with up to 12 threads and the Hiroshima road network

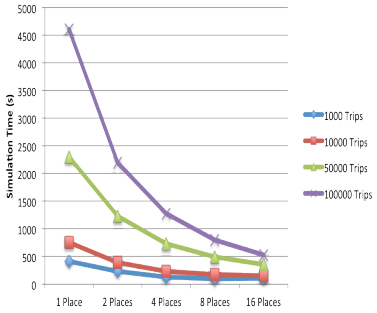

Figure 9 Multi-Node Performance with up to 16 Places and the Tokyo road network

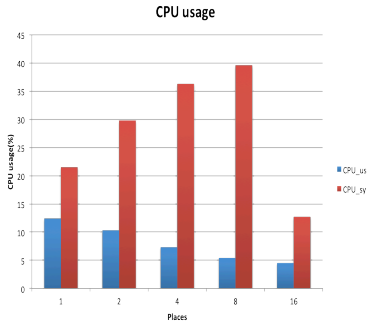

Figure 10 CPU Usage

V. Performance Evaluation

In this section we conduct performance evaluation of XAXIS-based traffic simulation with TSUBAME, the 5th ranked (as of 2011/11) super computer located at Tokyo Institute of Technology and present the performance results.

A. Evaluation Environment

A traffic simulation was conducted on XAXIS using the TSUBAME 2.0 super-computer at the Global Scientific Information and Computing Center, Tokyo Institute of Technology. TSUBAME 2.0 is a production supercomputer operated by Global Scientific Information and Computing Center (GSIC) at the Tokyo Institute of Technology. TSUBAME 2.0 has more than 1,400 compute nodes inter-connected by high-bandwidth full-bisection-wide Infiniband fat nodes, which can differ in their local memory capacity. All of the compute nodes share a scalable storage system with a capacity of 7 PB.

Each TSUBAME 2.0 node has two Intel Westmere EP 2.93 GHz processors (Xeon X5670, 256-KB L2 cache, 12-MB L3), three NVIDIA Fermi M2050 GPUs, and 50 GB of local memory. The operating system is SUSE Linux Enterprise 11. Each node has a theoretical peak of 1.7 teraflops (TFLOPS). The main system consists of 1,408 computing nodes, and the total peak performance can reach 2.4 PFLOPS. Each of the CPUs in TSUBAME 2.0 has six physical cores and supports up to 12 hardware threads with Intel's hyper-threading technology, thus achieving up to 76 gigaflops (GFLOPS).

The interconnect that links the 1,400 computing nodes with storage is the latest QDR Infiniband (IB) network, which has 40 Gbps of bandwidth per link. Each computing node is connected to two IB links, so the communication bandwidth for the node is about 80 times larger than a fast LAN (1 Gbps). Not only the link speed at the end-point nodes, but the network topology of the entire system heavily affects the performance for large computations. TSUBAME 2.0 uses a full-bisection fat-tree topology, which accommodates applications that need more bandwidth than provided by such topologies as a torus or mesh. One of the key features of the TSUBAME 2.0 architecture is high-bandwidth hardware that transmits data efficiently. To sustain high rates of intra-node communications, the memory bandwidth is 32 GB/s to each CPU.

B. Performance Evaluation with Single Node

We firstly show the performance speed up ratio with one single node and varying number of threads and simulation overall steps using the Hiroshima road network that has 19124 cross points. The number of trips (trip equals to a vehicle moving from given start point and end point) is 115000 and the number of simulation steps is 100, 1000, 10000. The result is illustrated in Figure 8. The promising speed-ups are achieved with 1000 and 10000 steps. By making use of 12 threads in one node, around 5 fold speed-ups is achieved. However with larger steps, 10000 do not show such speed ups. As the CPU usage is shown in Figure 10, most of the CPU usage is used in the kernel level for waiting for incoming messages with busy wait, and not used for the computation if the number of trips per each time step is not sufficient.

C. Performance Evaluation with Multi-Node Environment: Decomposing Road Network for Parallel Processing

For the performance evaluation, we use the road network of the bay area in Tokyo which is obtained from Open Street Map [3]. The network is comprised of 161364 cross points and 203,363 roads which is roughly 8 times larger than the Hiroshima network used in a single-node experiment. To enable the parallel processing, we partition the network into multiple domains. The graph partitioning tool METIS [5] was used for partitioning the network into 16 domains so that cross points were distributed equally to each domain. Figure 10 shows the illustration on how the bay area of Tokyo is divided by allocating different color to different domain. One group (corresponding to one partitioned domain) is allocated to one compute node on TSUBAME.

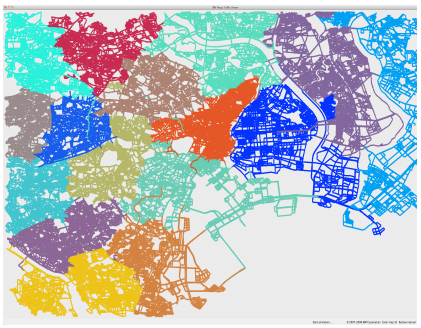

Figure 11 Graph partition of the Tokyo road network with 16 Places

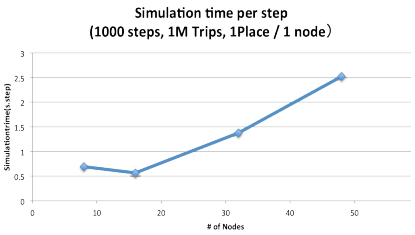

Figure 12 Synchronization Overhead

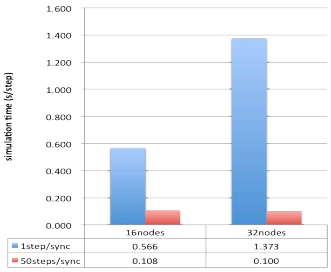

Figure 13 Performance Comparison with different synchronization frequency

One group consisted of about 10,000 cross points in average, and parallel distributed processing was executed using 16 nodes and 192 CPU cores in total. A simulation was conducted in conditions of 1,000 steps and varying number of trips, 1,000 trips, 10,000 trips, 50,000 trips, and 100,000 trips. With larger number of trips, the processing for each step becomes busier and consumes more CPU time. Figure 9 shows the result, where the horizontal axis denotes the number of places where one place corresponds to one physical compute node with 12 CPU cores, and vertical axis represents the execution time in seconds.

In the case of 100,000 trips, the simulation was completed within around 4500 seconds for all domains with one node (place) and 12 CPU cores. Meanwhile with 192 CPU cores and 16 nodes, it is successfully reduced to around 500 seconds. Other cases with less than 100,000 trips shows less performance scalability because the computational load is not sufficient for such scenarios. From these results, relatively heavy computation or larger number of trips per each step is required for linear scalability by leveraging the full power of such computing resources.

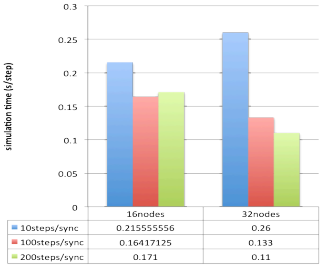

Figure 14 Performance Comparison with different synchronization frequency

D. *Measuring Synchronization Overhead*

Given the results in previous experiments, the linear performance speedups are not obtained if the number of trips per 1 simulation step is relatively smaller since the CPU usage becomes quite low and most of the time is spent on the synchronization with other computing activities. Although we can see linear speed-ups with 100000 trips per 1000 simulation steps (100 trips per steps), it might not be realistic number in some time period such as mid-night. In this sub-section, we firstly show preliminary experiments to show how synchronization at high frequency hinders the performance scalability in multi-node environment.

In this experiment, we measured the synchronization overhead in large-scale environment using 8, 16, 32, 48 nodes on the TSUBAME super computer. This experiments use the whole Japanese road network with 993,731 cross points and 2,552,160 roads. As shown in Figure 12, the elapsed time for each time step is unexpectedly increased with larger number of nodes from around 0.5 seconds to 2.5 seconds. The simulation parameters are 1000 simulation steps and 1 million trips.

To understand how synchronization hinders the performance scalability, we ran two experiments with different simulation workloads. The first experiment shown in Figure 13 is conducted with 1000 steps and 100,000 steps, and the second experiment shown in Figure 14 is done with 1000 steps and 1 million steps. We used 16 and 32 nodes and also tried different synchronization frequency, 1 and 50. With the latter frequency, simulation synchronization is conducted every 50 steps. Surprisingly the result shows dramatic performance speed up with the synchronization frequency of 50 by reducing 0.566 seconds to 0.108 seconds in the case of using 16 nodes. However if the computational load is sufficiently high in the second experiment with 16 nodes, the result in Figure 14 demonstrates that it is not that effective like the first experiment.

To conclude the finding in this experiment, it is highly required to introduce a method of avoiding the synchronization overhead. We will report our optimization method for avoiding this synchronization overhead in different paper.

VI. Related Work

In this section, we explain the overview of prior arts on large-scale agent-based simulation. In [17], the paper explains about the system helps to implement ABMs on GPU by presenting data parallel algorithms for simulating agent based model to gain massive parallelism of GPU. The system can gain updates per second to 10.2 with the 16.7 millions of agents. The purpose of the system is to gain large-scale of simulation and simulation time rather than to give a detail and interaction of agents which has agent

replication of O (1) average time.

In [8], the paper describes a platform AGENTFLY used for large-scale high-detail simulation of air traffic by using trajectory planning and several collision avoidance algorithms. The system can be evaluated by the performance of simulation when it changes the number of agents and system resources. The system can make a simulation involving the total of 52799 flights with up to 6500 airplanes.

In [18], the paper discusses performance modeling by run a simulation for different values of the input parameters with the scale of hundreds of agents and 5 nodes and makes a prediction of multi-agent based simulation.

In [19], the paper uses Level of Detail techniques to find a good balance between visual credibility and computational requirements. The goal of the system is to compare CPU gain and the behavior consistency. System's scale gains the agent number of 10000.

In [20], the paper aimed at improving the overall performance of large-scale un-manned aerial vehicle (UAV) applications on distributed execution by minimizing agent communication cost and making a load sharing amongst platforms. Simulation has a the agent number of 10000 and it uses 4 computers. Simulation time takes a few hours to reach a runtime ratio up to 5.

With regards to traffic simulation, the papers [21] [22] use the so-called queue model as the base of the traffic simulation. The system uses a parallel implementation to speed up the computation. The simulation uses a road network with 10564 nodes and 28624 links and have the trip number of 991471 trips from 6:00AM to 9:AM. The largest number of vehicles in the simulations is 162464 vehicles at 8.00 AM. The runtime ratio is nearly 800, it means that simulating 24 hours of all car traffic in less than two minutes.

VII. Concluding Remarks and Future Work

In this paper, we design and implement a highly scalable X10-based agent simulation middleware, and then demonstrate how large-scale traffic simulation is developed on top of the middleware. The experimental results demonstrate that it only takes around 800 seconds to conduct 12000 simulation steps for the road network of Tokyo with 192 CPU cores and 16 compute nodes, which enables super real-time simulation. As far as we know, our effort presented in this paper is the first effort that designs and implements multi agent simulation middleware with the state-of-the-art PGAS language and conduct the large-scale experiment on recent supercomputers. Future work includes the optimization for avoiding the synchronization overhead observed in the experiments and other work includes the following items:

Native X10 integration: In this paper, we only introduced the effort of running Java-based traffic simulator on XAXIS, but we are currently in the process of porting it to pure X10-based applications so that the entire software stack would be written in X10. After the porting is completed, we could obtain an option of compiling the en-tire software stack to X10 C++ backend. C++ backend allows us to use high performance communication libraries such as MPI, proprietary communication libraries for supercomputers such as BlueGene, and so forth. Thus we will be able to achieve more real-time simulation.

Other applications than traffic simulations: Many agent simulations follow time-based model and as observed in our experience, the synchronization overhead when advancing to next time step would prevent the performance scalability in large-scale simulations. Our proposed performance optimization technique - adaptively changing the synchronization frequency while maintaining the allowed simulation accuracy – could be generally applied to such time-based agent simulation.

References

[1] Yamamoto, G., Tai, H., and Mizuta, H. A Platform for Massive Agent-based Simulation and its Evaluation. AAMAS 2007, 900-902.

[2] Kawachiya, K. X10: A Programming Language for Multicore Era. Information Processing, 52(3) (2011), 342-356.

[3] Open Street Map, http://openstreetmap.jp/

[4] Saraswat, Vijay A., Sarkar, Vivek, and von Praun, Christoph. 2007. X10: concurrent pro-gramming for modern architectures. In Proceedings of the 12th ACM SIGPLAN symposi-um on Principles and practice of parallel programming (PPoPP '07). ACM, New York, NY, USA, 271-271.

[5] Karypis, G., and Kumar, V. Multilevel k-way Partitioning Scheme for Irregular Graphs. Journal of Parallel and Distributed Computing, 48 (1998), 96-129.

[6] Florian, M. A Traffic Equilibrium Model of Travel by Car and Public Transit Modes. Transportation Science, 11(2) (1977), 166-179.

[7] Wardrop, J.C. Some Theoretical Aspects of Road Traffic Research. Proc. Institute of Civil Engineers Part 2, 9 (1952), 325-378.Baldonado, M., Chang, C.-C.K., Gravano, L., Paepcke, A.: The Stanford Digital Library Metadata Architecture. Int. J. Digit. Libr. 1 (1997) 108–121

[8] David Sislak, Premysl Volf, et al, Distributed Platform for Large-Scale Agent-Based Simulations, Springer 2009,

[9] Gorgious Theodoropoulos, Yi Zhang, et al, Large Scale Distributed Simulation on the Grid, CCGrid 2006

[10] Sanjay Nayer, Rick Loffredo, et al, Large-Scale Multi-Agent-Based Simulation using Exemplars, AAMAS 2005

[11] Dan Chen et al. Large scale agent-based simulation on the grid, Journal Future Generation Computer Systems

[12] Yi Zhang et al, Grid-aware Large Scale Distributed Simulation of Agent-based Systems, 2005

[13] Jon Parker, A flexible, large-scale, distributed agent based epidemic model, WSC (Winter Simulation Conference) 2007

[14] Comparison of agent-based modeling software, http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software
[15] Takahashi, T. and Mizuta, H., Efficient Agent-based Simulation Framework for Multi-Node Supercomputers, The proceedings of the 2006 Winter Simulation Conference, 2006
[16] Traffic Pattern Census Data: http://www.mlit.go.jp/road/census/h22-1/index.html
[17] Mikola Lysenko, Roshan D'Souza and Keyvan Rahmani, Framework for Megascale Agent Based Model Simulations on the GPU, Parallel and Distributed Computing, 2007
[18] Dawit Mengistuet, Lars Lundberg, and Paul Davidsson, Performance Prediction of Multi-Agent Based Simulation Applications on the Grid, WASET 2007
[19] Lauren Naverro, Fabien Flacher, and Vincent Corruble, Dynamic Level of Detail for Large Scale Agent-Base Urban Simulations, AMMAS 2011
[20] Myeoung-Wek Jang et al, Scalable Agent Distribution Mechanisms for Large-Scale UAV Simulations, 2005 International Conference on Integration of Knowledge Intensive Mult-Agent Systems, 2005
[21] Nurhan Cetin et al,A Large-Scale Agent-Based Traffic Mircosimulation Based on Queue Model, 3rd Swiss Transport Research Conference 2003
[22] Bryan Raney, Nurhan Cetin et al,Large Scale Multi-Agent Transportation Simulations, 42nd ERSA Conress, Dortmund 2002
[23] Gipps, P.G. A behavioural car-following model for computer simulation. Transportation Research Board Part B, 15, 105-111, 1981
[24] Megaffic (IBM Mega Traffic Simulator) http://www.research.ibm.com/trl/projects/socsim/project_e.htm
[25] Toyotaro Suzumura, Sei Kato, Takashi Imamichi, Mikio Takeuchi, Hiroki Kanezashi, Tsuyoshi Ide, and Tamiya Onodera. 2012. X10-based massive parallel large-scale traffic flow simulation. In*Proceedings of the 2012 ACM SIGPLAN X10 Workshop* (X10 '12). ACM, New York