Project CASSIA — Framework for Exhaustive and Large-scale Social Simulation —

Itsuki Noda, Yohsuke Murase, Nobuyasu Ito, Kiyoshi Izumi, Hiromitsu Hattori, Tomio Kamada, and Hideyuki Mizuta

1 Overview

Project CASSIA (Comprehensive Architecture of Social Simulation for Inclusive Analysis) aims to develop a framework to administer to execute large-scale multi-agent simulations exhaustively to analyze socially interactive systems. The framework will realize engineering environment to design and synthesize social systems like traffics, economy and politics.

The framework consists of:

- MASS Planning Module: a manager module conducts effective execution plans of simulations among massive possible conditions according to available computer resources.
- MASS Parallel Middleware: an execution middleware provides functionality to realize distributed multi-agent simulation on many-core computers.

Itsuki Noda
AIST, 1-1-1 Umezono, Tsukuba, Japan, e-mail: i.noda@aist.go.jp

Yohsuke Murase
RIKEN Center for Computational Science, 7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Japan, e-mail: yohsuke.murase@gmail.com

Nobuyasu Ito
Univ. Tokyo & Riken

Kiyoshi Izumi
Univ. Tokyo

Hiromitsu Hattori
Ritsumei Univ.

Tomio Kamada
Kobe Univ.

Hideyuki Mizuta
IBM

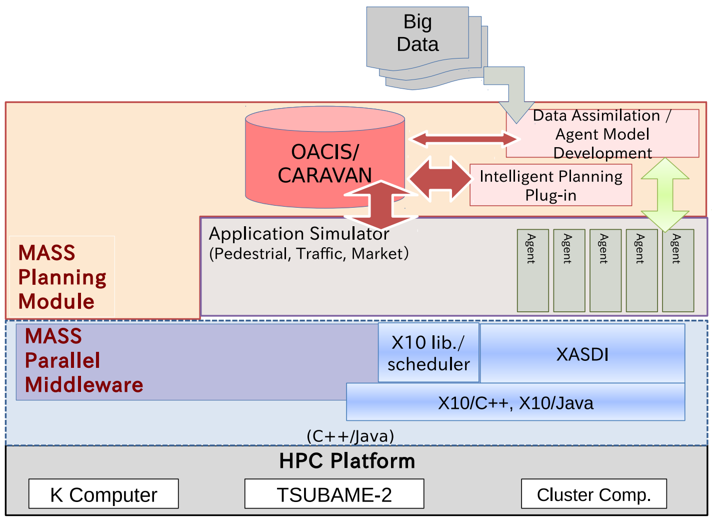

Fig. 1 Cassia Framework

2 MASS Planning Module

In this section, we give an overview on the two frameworks for parameter-space exploration, OACIS and CARAVAN. Although both of these frameworks are designed in order to conduct parameter-space exploration making full use of HPC resources, they differ in the target scale of each job. In one hand, a certain class of research issues requires to use the maximum computing power to solve a single large problem, so called capability computing. On the other hand, other researches require to do many small- or medium-scale simulations for parameter-space explorations, i.e., capacity computing. We categorized these problems into four classes depending on the scale of a single simulation job as summarized in Table 1. In this Table, it is assumed that the total amount of computation is order of ten exa floating-point operations. The left most column, which we call class A, corresponds to typical capability computing. The number of independent jobs is at most 10^2. In class B, a typical single job is an MPI-parallel program using $10^3 \sim 10^5$ CPU cores. The number of jobs amounts to $10^3 \sim 10^5$. In this class, a naive manual management of jobs is no longer possible and a framework for managing jobs is necessary. When typical job scale is serial or shared-memory parallel application, as labelled class C, the number of jobs expands up to $10^6 \sim 10^9$, which requires an even harder job management. In this class, the parameter selection and interpretation of the results for each job must be done algorithmically. Finally, on the right most class, where a single job becomes a function level, the number of jobs is more than 10^{10}. Thus, for capacity computing ranging from class B to D, the demands for frameworks can be totally different depending on the granularity of jobs. This is why we developed two frameworks, OACIS and CARAVAN for classes B and C, where majority of the social simulations are found.

Table 1 Categorization of the problems according to the scale of single simulation job. To calculate the required number of operations, FLOPS, the number of CPU cores for each job, we made assumptions that an exa-flops computer is available, the efficiency of each job is 10%, and the duration of each job is order of 10^2 seconds. From left to right, the typical scale becomes finer while the typical number of jobs gets greater. In the last two lines, we showed an example of social simulations and a framework used for parallel job execution.

class	A	B	C	D
# of jobs	$10^0 \sim 10^2$	$10^3 \sim 10^5$	$10^6 \sim 10^9$	$10^{10} \sim$
# of operations / job	$10^{19} \sim 10^{17}$	$10^{16} \sim 10^{14}$	$10^{13} \sim 10^{10}$	$10^9 \sim$
FLOPS / job	$10^{18} \sim 10^{16}$	$10^{15} \sim 10^{13}$	$10^{12} \sim 10^9$	$10^8 \sim$
# of cores / job	$10^8 \sim 10^6$	$10^5 \sim 10^3$	$10^2 \sim 10^{-1}$	$10^{-2} \sim$
typical job scale	large-scale MPI	medium scale MPI	SMP or serial	function
parameter selection	manual	manual or auto	auto	auto
social simulation application	-	traffic in metropolitan area	traffic in a city	data-driven model
frameworks		OACIS	CARAVAN	Map-Reduce

OACIS, which stands for Organizing Assistant for Comprehensive and Interactive Simulations, is a job management framework for problems in class B[**?**]. It is available as an open-source software under the MIT license. (http://github.com/crest-cassia/oacis). This class of problems require researchers to carry out many simulation jobs changing models and parameters by trial and error. This kind of trial-and-error approach often causes a problem of job management because of a large amount of repetitive works. Such repetitions are not only troublesome and tedious but prone to human errors. OACIS is designed to let researchers conduct their research in an efficient, reliable, and reproducible way, helping management of simulation jobs and results.

The system architecture of OACIS is depicted in Fig. 2. It is a web application developed based on the Ruby on Rails framework, which provides an interactive user-interface. The application server is responsible for handling requests from users. When a user creates a job using a web browser, the record of the job is created in the database. Another daemon process, which we denote as "worker", periodically checks whether a job is ready to be submitted to a remote host. If a job is found, the worker generates a shell script to execute a job and submits it to the job scheduler on the remote host (which we call "computational hosts") by SSH connection. The worker process then periodically checks the status of the submitted jobs and, when the jobs are finished, it downloads the results and stores them into designated storage and database appropriately. Hence, users do not have to check the job status by themselves and the simulation results are kept in an organized and traceable way. Various logs, including the values of parameters, executed dates, elapsed times, the version number of the simulator, are automatically kept as well. A simulator on OACIS is registered as a command line string to execute the simulation, not as the execution program itself. By this design, OACIS can run simulators in various research fields, which may be written in different programming language.

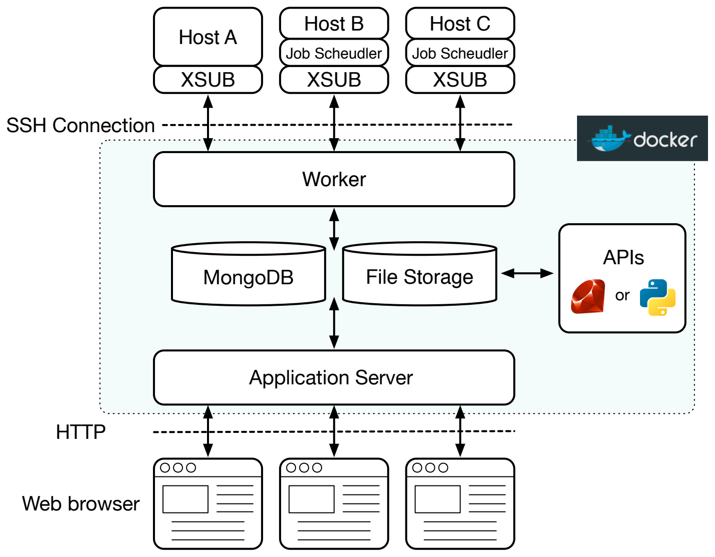

Fig. 2 A system overview of OACIS.

In addition to an interactive user-interface, OACIS provides application programming interfaces (APIs) in Ruby and Python programming languages. Any set of operations on OACIS is programmable using the APIs, which can be used for various types of parameter-space explorations including parameter sweeps, sensitivity analysis, and optimization of parameters.

CARAVAN is another framework designed for class C jobs. It is also available as an open-source software under the MIT license.(https://github.com/crest-cassia/caravan)

Figure 3 shows the whole architecture of CARAVAN. It consists of three parts: search engine, scheduler, and simulator. “Simulator” is an executable program prepared for each use case. Once a user integrate a simulator into CARAVAN, it is executed in parallel. Since a simulator is executed as an external process, a simulator may be implemented in any language as in OACIS. “Scheduler” is a part which is responsible for parallelization. It receives the commands to execute simulators from the search engine, distributes them to available nodes, and executes the simulator in parallel. This part is implemented in X10 programming language using MPI for a communication layer. “Search engine” is a part which determines the policy on how parameter-space is explored. More specifically, it generates a series of commands to be executed in parallel, send them to scheduler. It also receives the results from the scheduler when these tasks are done. Based on the received results, search engine can generate other sets of tasks repeatedly as many as a user wants. This part is written in Python. A simulator and a search engine must be prepared by each user while the scheduler does not have to be modified once it is built.

When writing a simulator and a search engine, users do not have to explicitly take care of the parallelization. The scheduler is designed so that the whole application can scale up to tens of thousands of processes. To evaluate the performance of the scheduler on the K computer, we tested an embarrassingly parallel problem, in which each task takes about 20 seconds. We obtained a result that the efficiency of the task scheduling remains more than 99% even when the number of MPI processes is scaled up to 18432.

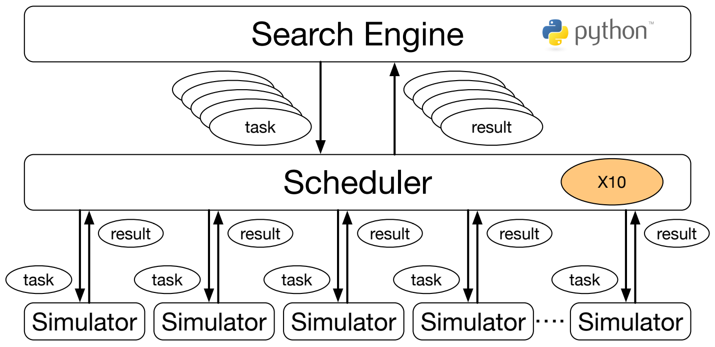

Fig. 3 A system overview of CARAVAN

3 MASS Parallel Middleware

3.1 X10 Extentions and Plham

(Kamada)

Plham is a platform for large-scale and high-frequency artificial market simulation. It consists of models of markets for each stocks and three types of agents (high-freq. traders, short-term and long-term traders).

In order to enhance parallelism of computation, we introduce asynchronous computation in agents and communication between agents/markets, and provide high-level library to program them.

3.2 XASDI

In this section, we describe the overview of X10-based Agent Simulation on Distributed Infrastructure (XASDI). This framework is published as an open source software (https://github.com/x10-lang/xasdi) under the Eclipse Public License (EPL)

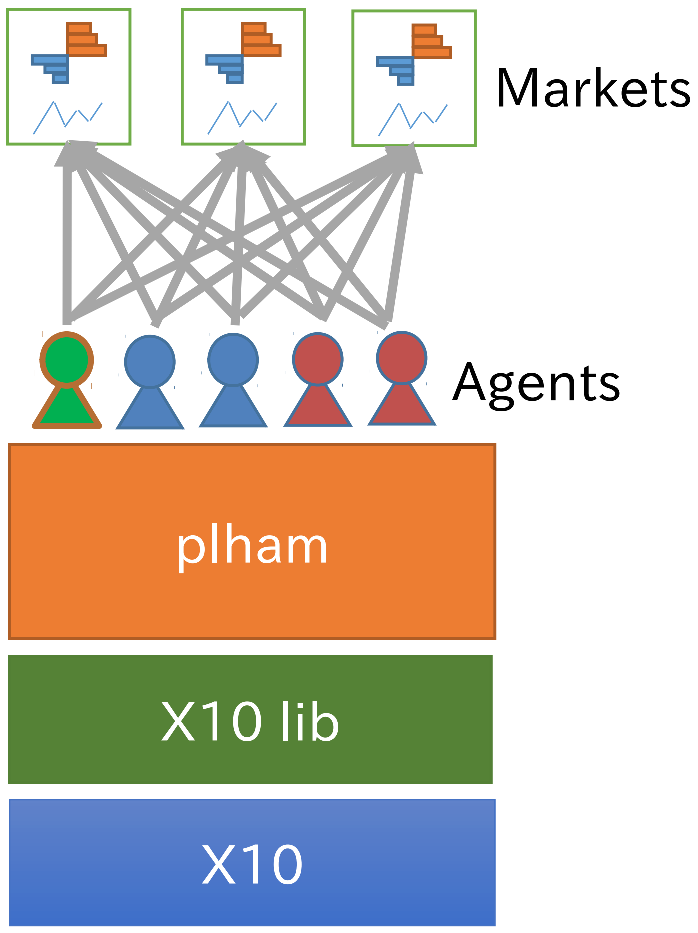

Fig. 4 Plham

XASDI is large-scale agent-based social simulation framework with enormous number (billions) of agents to represent citizens in cities or countries. XASDI enables distributed simulation with the X10 language for post-Peta Scale machines. The X10 programming language (http://x10-lang.org/) is the APGAS (Asynchronous, Partitioned Global Address Space) language that provides highly parallel and distributed functionalities with Java-like syntax [**?**]. X10 application can be compiled and executed on Java environment [**?**, **?**] With this feature, XASDI provides easy-to-use API with Java that is familiar to application programmer of social simulations and can be developed with powerful IDE functionalities (e.g. Eclipse refactoring and debugger).

XASDI software stack contains core runtime written in X10 language for distributed agent and execution management and Java API bridge to enable application programmer to utilize familiar Java languages (Figure 6). By utilizing XASDI

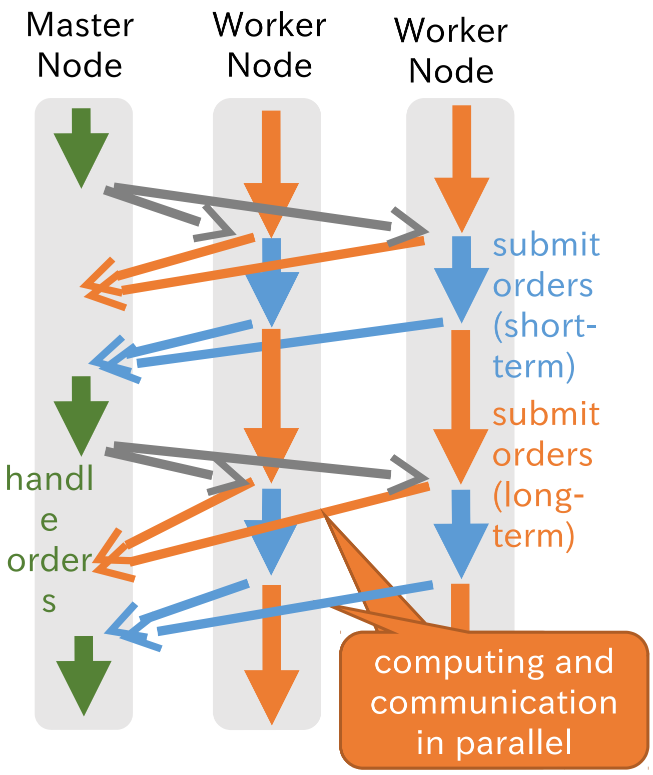
Master Node
Worker Node
Worker Node
submit orders (short-term)
submit orders (long-term)
handle orders
computing and communication in parallel

Fig. 5 Parallel Execution of Market Simulation

framework users can easily develop their social simulator with Java on distributed parallel environment without studying the new X10 language.

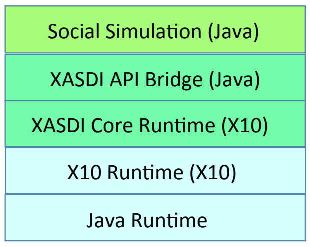

Fig. 6 XASDI Software Stack

The agent in XASDI is referred to as Citizen and Citizen has corresponding CitizenProxy that is managed in the simulation environment to exchange messages. To manage CitizenProxy, XASDI provides a hierarchical container structure called

Place, Region and World (see Figure 7. CitizenProxies belong to a Place and Places belong to a Region. World can contain several Regions, but usually there is only one Region in the World.

Here, we need to note that the confusing terminology of the X10 programming language and the framework. The X10 use the term “Place”, too, but the meaning of the term is different. The Place of X10 is used to denote the distributed execution environment for multi-core or multi-node. For this meaning, we will use “X10 Place (node)” in distinction from the Place container of agents. Only one World instance exists in one X10 Place and manages lists of entities in the world including Region and Citizen. The world can also contain IDs of Citizens in other nodes.

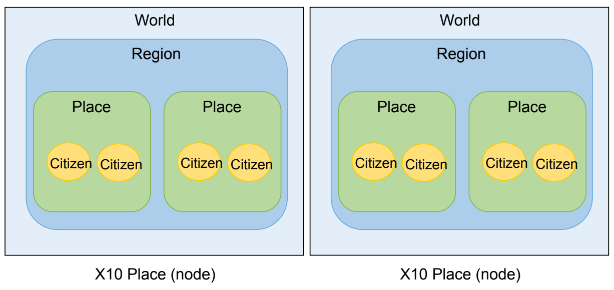

Fig. 7 XASDI hierarchical structure to manage agents

Other important classes in XASDI are Message, MessageRepository and Driver. MessageRepository manages Message exchange among CitizenProxy and environment and this class also works as interface between Java environment and X10 environment to exchange Messages in distributed X10 Places. Driver manages execution of the simulation with a corresponding thread. Each Driver is related to Places (and Citizens in the Places) where it has a responsibility for execution.

Finally, XASDI provides a logging mechanism. By preparing log definitions for the application, it can output the simulation log at each X10 Places.

Application users of XASDI need to develop their own simulation application by utilizing these classes and execute the application with XASDI library on Java and X10. We describe the execution process on XASDI and the simple customization for the sample application bundled with XASDI.

A user starts the simulation by executing the shell script to invoke the X10 core runtime. After the preparation of X10 environment, Launcher for the simulation written by Java is called at each X10 Place. Launcher reads the initialization file and generates a Region, Places and Drivers. If agents are needed to exist from the beginning of the simulation, Launcher or Driver generates Citizens. Citizens can also be generated by Driver through the method of the Region during the simulation at given time or randomly. For example, consumer agents are generated when they enter a shopping mall.

The main simulation process is executed through Drivers. Simulation managers in the X10 core environment generate threads and invokes call back method in each

Driver in parallel. One simulation time step can be divided into phases. The number of phases are determined at the beginning of the simulation with initialization file. The method of the Driver looks at the time and phase given by the environment and determines the action that the corresponding Places and Citizens should perform.

In one node, there is one Region instance that manages the execution of the simulator. The core framework written using X10 manages the Regions and Message Repositories of distributed nodes. The Message Repository supports several kinds of messages such as individual message, broadcast message, and control message to move agent between X10 Places. An individual message is a standard message from one agent to another. A broadcast message is sent to all nodes and received by the Region or agents corresponding to the type of message. A move message is a control message for X10 core runtime to remove the Citizen at the source node and restore it at the destination node with serialized field data stored in the message.

As applications for large-scale simulation on XASDI, we developped a large-scale traffic simulator for city [**?**] and a shopping mall simulator with integrated model of walking and purchasing behavior [**?**]. Fig. 8 shows the weak scaling performance of XASDI by using our application for shopping mall simulation with distributed consumer agents [**?**].

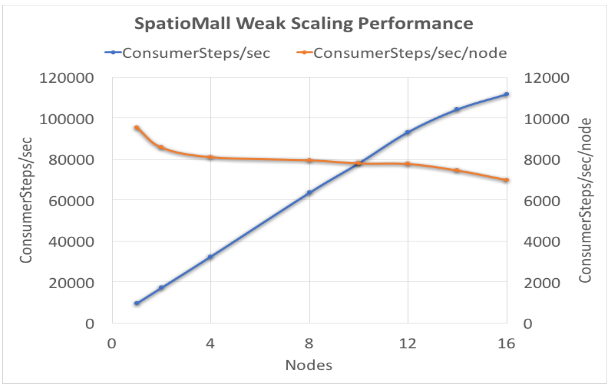

Fig. 8 Weal scaling performance of XASDI with shopping mall application.

4 Applications

4.1 Market Simulation

(Izumi)

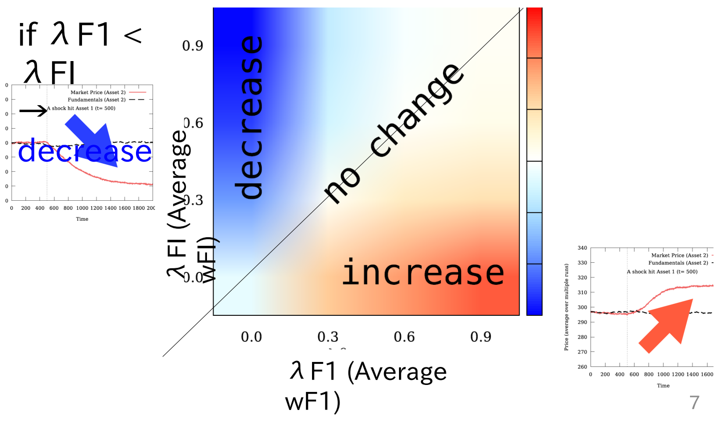

Fig. 9 Phase Diagram of Market Simulation

4.2 *Pedestrian Simulation*

(Noda)

CASSIA Framework can illustrate a trade-off structure in planning of evacuations from disasters. Optimization in disaster responses is serious requirements for local governments. But, such optimization includes multiple objective functions. So, the important issue is how to understand trade-off structures of such multi-objective functions over large number of policy options.

We apply our framework to evaluate evacuation plans, which have over 300 control parameters, to find out such trade-off. We implement NSGA-II algorithm to search optimal structures over large parameter spaces, and utilize the performance of K-computers to speed-up the process.

4.3 *Traffic Simulation*

(Hattori, Ito?)

5 Computational Roadmaps of Social Simulations

As described above, our purpose is to determine how HPC contributes to the advancement of research on social simulation or to clarify the computational power required for real applications of social simulation. In this section, we focus on three applications and try to develop roadmaps for them.

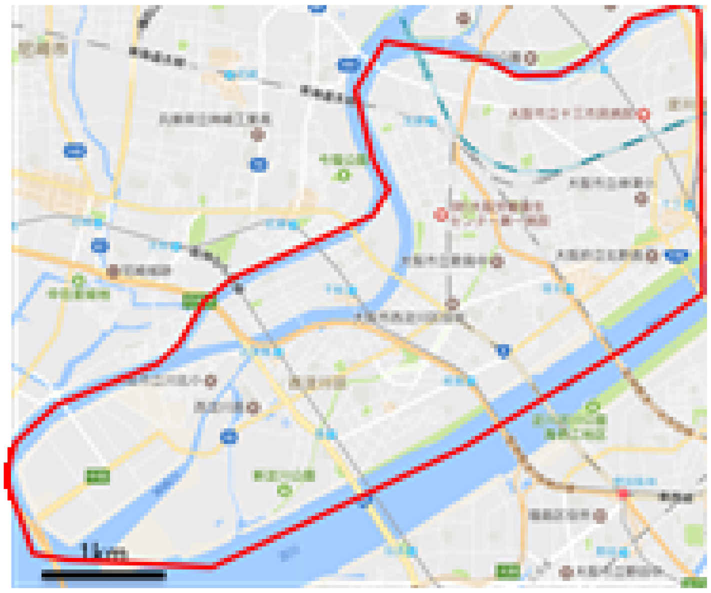

Fig. 10 Nishiyodogawa Area used in Pedestrian Simulation

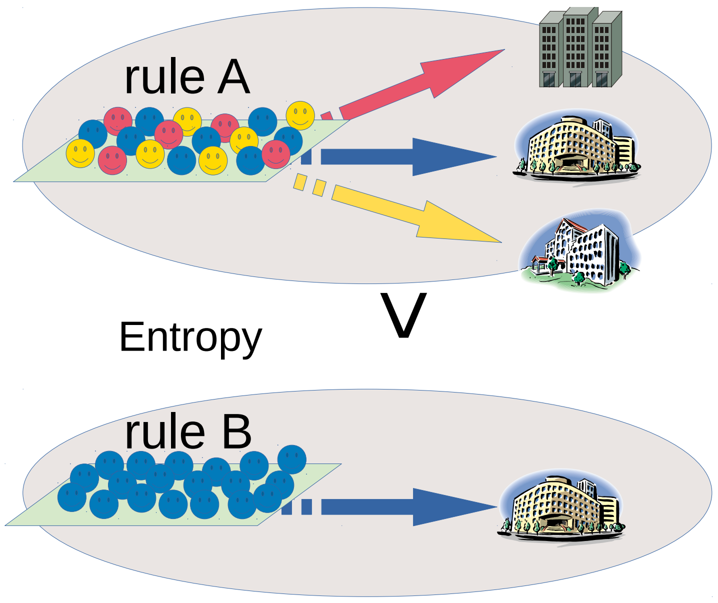

Fig. 11 Rule Entropy

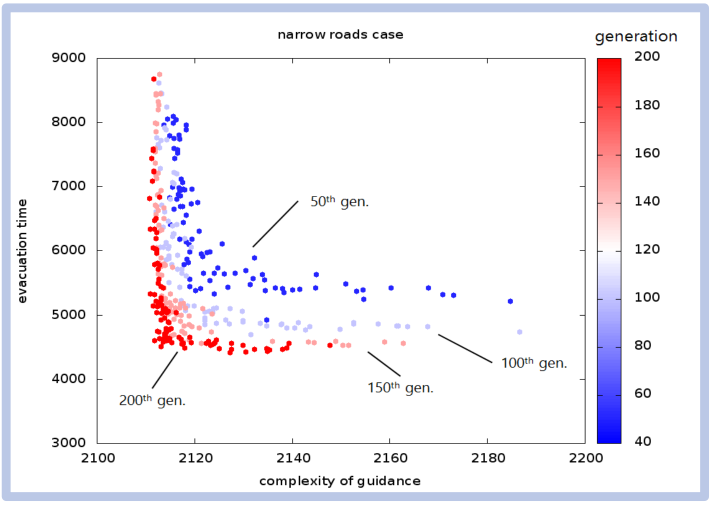

Fig. 12 Result of Evacuation Simulation (narrow road)

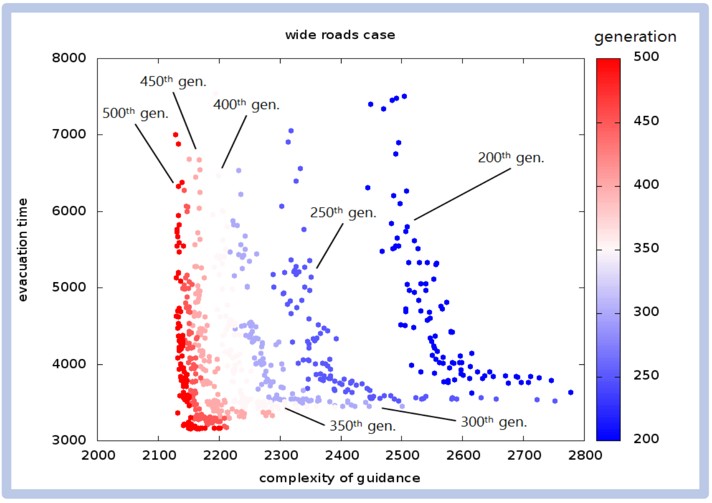

Fig. 13 Result of Evacuation Simulation (wide road)

In the development of these roadmaps, we adopted two indexes to measure the computational cost, “number of situations” and “complexity of one simulation session”. As discussed in section **??**, we considered exhaustive evaluation by simulation as a key methodology of social simulation. Therefore, to evaluate the model, examining many conditions and models is important. The index of “number of situations” indicates this number. Meanwhile, ordinal computational cost of a simulation, which is determined by the number of entities and the number of interactions among the entities, is important. In addition, in multiagent simulation, the computational cost of thinking of each agent is significant. In the following discussion, we integrate these complexities as “complexity of one simulation session”.

5.1 Evacuation/Pedestrian Simulation

The main target of evacuation simulation is not to find an optimal plan of evacuation for a given disaster situation, but to evaluate the feasibility and robustness of executable candidates of evacuation plans or guidance policies. Because of natures of disasters, it is difficult to acquire complete information to determine the conditions of evacuations in the event of a disaster. Therefore, it is almost impossible to validate optimality for each disaster. Instead, local governments should strive to establish feasible plans that will work robustly in most situations of disasters. This means evaluation of evacuation plans should be done under widely varying disaster scenarios. A massively parallel computer simulation will make such evaluations easy and effective.

Several simulations have been performed for evaluating such evacuation plans [**?**][**?**][**?**][**?**]. For example, a simulation of an evacuation from a Tsunami struck city in Tokai area in Japan was performed, where a massive damage is expected to occur due to the great Tokai-Tonankai earthquake. To help understand the importance of the relationship between evacuation scale (populations of evacuees) and effectiveness of evacuation plans, we conducted the following exhaustive simulations considering various sizes and evacuation policies (evacuee’s origin-destination (OD) plans). The simulation results indicate that the scale of evacuation can be grouped into two categories, namely, “large” ($>$ 3,000 evacuees) and “small” ($<$ 3,000 evacuees). Each evacuation plan has similar relative effectiveness in each category. The actual evacuation size (population) may change based on various factors such as daytime/nighttime, number of visitors/travelers, weather, and special events. This implies that citizens and local governments should consider at least two plans for large- and small-scale evacuations.

We execute the evacuation simulation described above to arrive at a reference point for illustrating computational costs of various actual applications. In the above simulation, we considered the following scenarios:

- 2,187 OD plans and
- 8 cases of evacuation population (70–10,000 agents).

Therefore, in total, 17,497 simulation scenarios were executed over about 30 days when using a single process on Xeon E5 CPU (2.7 GHz). We denote this reference point as the rectangle “city zone, TSUNAMI” in figure 14.

We can easily extend the simulation scale. Although a population of only 10,000 is considered in “city zone, TSUNAMI”, we can extend the simulation to a more densely populated area such as in Tokyo. For example, we performed a similar simulation analysis in the Kanazawa area, which is located on the coast along the Japan Sea and experiences snowfall in the winter. In this case, the population size is similar (about 6,000 agents), but the number of combinations of scenarios increases to 4,194,304 (2^{22}). The rectangle “city zone, TSUNAMI and HEAVY SNOW” in figure 14 denotes this calculation cost.

We can further extend the simulation to a large scale with a larger number of scenarios. Kitasenju area, a large transfer station surrounded by rivers, has a population of 70,000, and the computational cost of simulating this area is denoted by “dense-population zone, complex disaster” in figure 14. Because this area is densely populated and complex, we have combinations of 44 policy candidates, that is 2^{44} scenarios. In the case of Tokyo, we need additional computational power. In figure 14, “megacity” corresponds a huge city such as Tokyo. In this case, the size of evacuation and the number of possible scenarios is very large. Therefore, peta- or exa-scale HPC is required to handle such simulations.

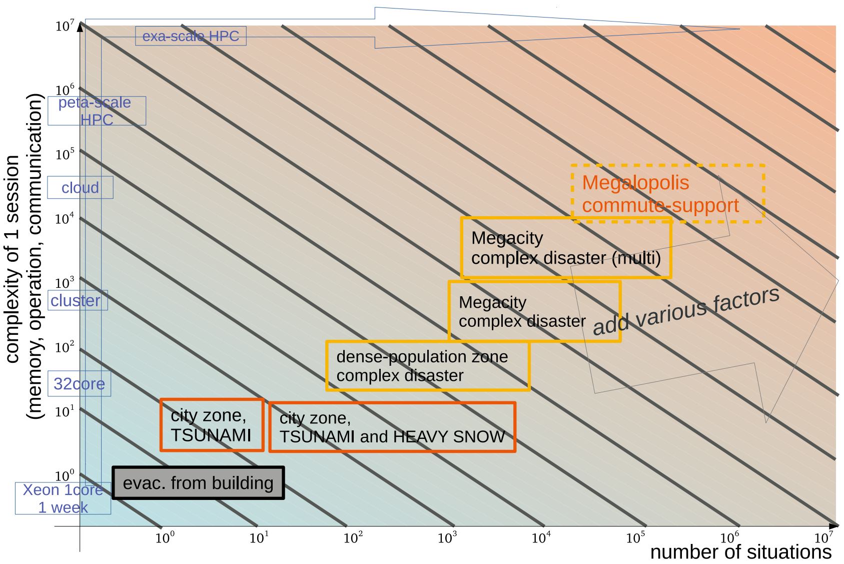

Fig. 14 Roadmap of Evacuation Simulation

5.2 *Traffic Simulation*

Road traffic is an important domain from the viewpoint of applying social simulation. Traffic simulation has been extensively researched over a long period, and recently, the focus has been on multiagent simulation, in which each agent behaves according to its own preferences and inference rules. Big data advances in computational power enable us to perform such detailed simulations.

[**?**] have been developing a traffic simulator called IBM Mega Traffic Simulator that can run large scale traffic simulations on XASDI middleware. The main feature of this simulator is its ability to reflect individual drivers' preferences. Using this feature, according to big-data, we can adapt parameters in the simulation that cause differences in drivers' tendency.

To create a reference point for the roadmap of the traffic simulation, we considered the case of evaluating road restriction policies for road construction in the Hiroshima area[**?**]. In this case, we performed simulations of the following scales:

- 70,000 agents (trips), 120,000 road links, and 15 hours and
- 20 cases

In this case, the calculation required about one day when using a single process on Xion E5 CPU. We denote this reference point as "million city, road plan" in figure 15.

We can draw out the roadmap from this reference point. When considering the Tokyo area, the number of agents increases up to about 2 million and the number of road links increases to about 610,000. Moreover, if we consider a larger area such as the Tokyo metropolitan area, the population increases about 4 million and the number of road links increases to 2.5 million. These calculation costs are plotted as "Tokyo, traffic control" and "metropolis, traffic control" in figure 15.

When we consider a big event, we must list a large number of cases to evaluate the robustness of road traffic to accidents, whereas the scenarios mentioned above pertain to normal situations that are repeated every day. Because various situations affect traffics, the number of situations increases quickly. These costs are plotted as "Tokyo, big event", "metropolis, big event" and "whole Japan, big event" in figure 15, and they require exa-scale computational power.

5.3 Market Simulation

Market simulations are another important application of multiagent simulations, in which agents directly affect each other by selling/buying stocks and/or currencies [**?**]. Compared with evacuation and traffic simulations, market simulations are not constrained by physical space. Therefore, the time cycles of agents' interactions may be quite short. Moreover, the ways of thinking of agents show large variations. This means that the market simulations also require huge computational cost.

As the reference point of the calculation cost in market simulations, we present the case of "tic size" evaluation. In this scenario, we conducted a simulation of multiple markets having different tic sizes, which is the minimum price unit for trading stocks. Market companies such as Japan Exchange Group internationally compete with each other by providing attractive services to traders. A small tic size is one of such services that considerably increases cost. Therefore, such organizations need evaluations of changes to such services in advance. In collaborative works with Japan Exchange Group, we conducted a simulation experiment to find key condi-

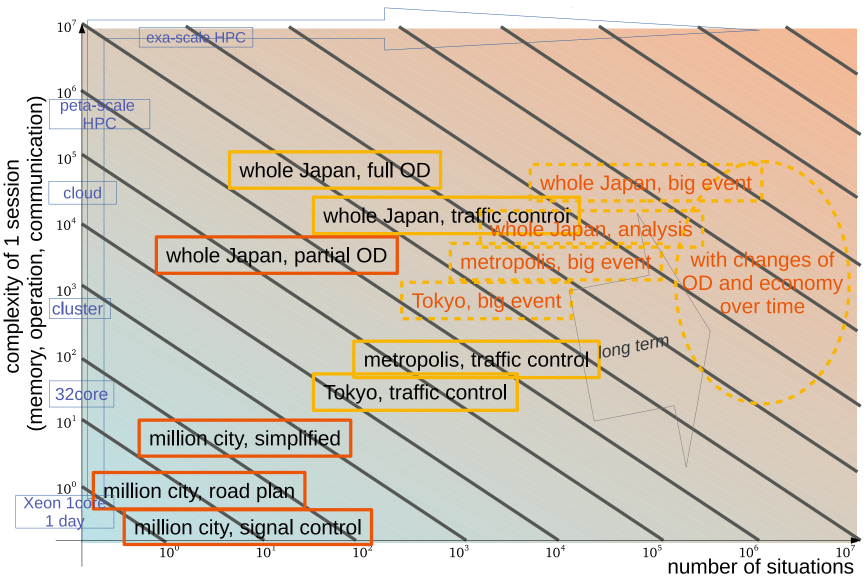

Fig. 15 Roadmap of Traffic Simulation

tions that determine market share among markets. In the simulation, we considered the following scenario:

- one good in two markets, 1,000 agents, and 10 million cycles
- five cases of tic size and 100 simulation runs per case

This simulation takes about one day when using a single thread on a Xeon E5 CPU. As the reference point, we plot this as "tic size" in figure 16.

We are considering extending the market simulations to various applications used for stock market analyses. For example, it is in the interest of market companies to determine "daily limit" and "cut-off" prices[**?**]. In this case, the simulation must handle 10–20 goods. Moreover, evaluating the effects of "arbitrage" [**?**], which involves trading rather quickly in intervals of milliseconds, is important from the viewpoint of maintaining sound market conditions. This will increase the computational cost, as plotted in figure 16. Another topic is the evaluation of "Basel Capital Accords", which deal with the soundness of banks in markets. In the present study, we executed the case of three names for the Basel Accords, but we will extend it to 100 names in the real application.

The evaluation of "systemic risks of inter-bank network" is an important issue in market evaluation. However, currently, the computational cost of a naive simulation exceeds exa-scale HPC.

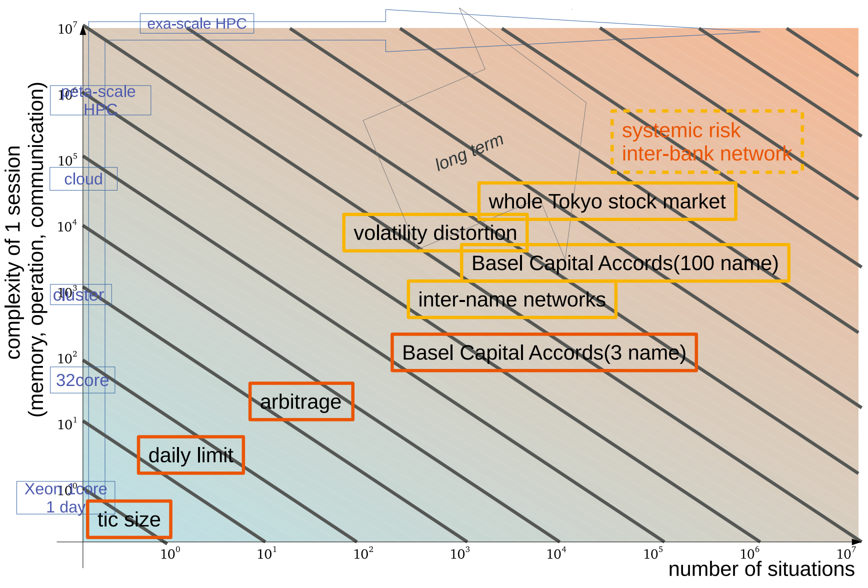

Fig. 16 Roadmap of Market Simulation

References