An Adaptive VM Provisioning Method for Large-Scale Agent-based Traffic Simulations on the Cloud

Masatoshi Hanai*§, Toyotaro Suzumura†‡§, Anthony Ventresque†§¶, and Kazuyuki Shudo*
*Tokyo Institute of Technology, 2-12-1 Ookayama, Meguro, Tokyo, Japan
Email: {hanai.aa, shudo}@{m, is}.titech.ac.jp
†IBM Research, Damastown Industrial Estate, Mulhuddart, Dublin 15, Ireland
Email: suzumura@acm.org
‡JST CREST
§School of Computer Science and Informatics, University College Dublin, Ireland
Email: anthony.ventresque@ucd.ie
¶Lero, the Irish Software Engineering Research Centre

***Abstract*—Using the Cloud for large-scale distributed simulations, such as agent-based traffic simulations, sounds like a good idea, as it is possible to provision and release easily processing nodes (e.g., virtual machines) in the Cloud. However, the question is complex as it involves users' objectives, such as, time to process the simulation and cost of the simulation, and because the workload evolves in distributed simulations, in each node and the whole system, and this impact the resource provisioning plans. This paper proposes two main contributions: (i) a method for efficient utilization of computational resources for distributed agent-based simulations, providing a mechanism that adapts the resource provisioning to users' objectives and workload evolution; and (ii) a staged asynchronous migration technique to limit the migration overhead when the number of workers change. Our preliminary experimental results on a 24 hour scenario of traffic in the city of Tokyo show that our system outperforms a static provisioning by 12% in average and 23% during periods when workload changes a lot.**

***Keywords*—Large-scale Agent-based Simulation, Traffic Simulation, Cloud Computing, Resource Provisioning, Monetary Cost.**

I. Introduction

Research on agent-based simulation has been essential for areas such as, transportation, environmental protection, prediction of global economic evolution [24]. Working on realistic situations, e.g., large urban areas or large social systems, is a challenging problem giving the scale considered. Several simulating methods and simulators have been proposed in the past to run the simulations on distributed systems [15], [27], each computing node of the distributed system hosting a simulator and a partition of the simulation model.

There are two main problems with distributed simulations though: (i) depending on users' preferences and objectives, finding a right combination of number and characteristics of computing nodes can be difficult; this is particularly evident in the Cloud, where there are numerous (different) options and provisioning is easy; and (ii) workload keeps changing in large-scale simulation models, at a global level (e.g., more vehicles at peak commuting times than during the night) and at a local level (e.g., vehicles moving from one partition to another); previous work show that this can lead to computing imbalance [26] and/or increased communication between computing nodes.

In this paper, we propose a framework for adaptive resource provisioning, which enables to increase and decrease the number of machines during the execution of the simulation, in order to achieve an efficient utilisation of computational resources. Our framework makes prediction on the evolution of the workload and decides on the characteristics and the number of virtual machines that need to be provisioned or released. We also propose a novel solution for efficient migration of workload between computing nodes. Our staged asynchronous migration achieves the very fast migration by overlapping the migration to the original simulation processing while the simulating results are exactly same as non migrating simulation. Our proposal finally achieves to reduce the payment cost by 12 % in average and 23 % during periods when workload changes a lot in all of Tokyo traffic simulation.

The rest of paper is organized as follows. In Section II, we describe the context of our research. In Section III, we give an overview of our proposed system. In Section IV, we discuss some more efficient way to migrate simulation object at run time. In Section V, we describe the implementation of our system and some evaluation based on a 24 hour scenario of traffic in the whole urban area of Tokyo. Section VI discussed some related work and finally we conclude in Section VII.

II. Background

This section aims at providing the reader with a detailed description of two important elements of the context of our research, i.e., IBM Mega Traffic Simulator and the Cloud's cost model (extended recently to supercomputers).

A. The IBM Mega Traffic Simulator

In our study we use *Megaffic*, a large-scale distributed agent-based traffic simulator developed by IBM [22], [23], [27]. The advantage of using Megaffic for traffic simulations is twofold. On the one hand Megaffic optimises some of the agents (i.e., drivers) decisions by estimating some of the parameters of the model from probe-car data, differentiating Megaffic from many other agent-based traffic simulator which

need to calibrate these parameters during the simulation. In short, Megaffic precomputes some of the simulation data, such as, road segments and lanes chosen by the drivers on their route, speed of the vehicles on the road. On the other hand Megaffic is designed for running on massively parallel computers and has proven to be able to simulate microscopic (i.e., agent-based) models of vehicular traffic in several cities and even the whole island of Japan. To do so, Megaffic is built on top of *X10-based Agent eXecutive Infrastructure for Simulation* (XAXIS), a platform for distributed agent-based simulations acting as a middleware between the complexity of the distributed systems and the applications (Megaffic in our case). XAXIS is based on *X10* [14], a parallel programming language suitable for multi-core architectures, which is being developed by IBM Research.

In Megaffic, an agent represents a driver of a vehicle, who travels along the road. There are three elements defining the simulation model that need to be set up before execution: route selection, speed selection and lane selection. Execution steps are divided in two: pre-iteration and iteration. During the pre-iteration phase, the origin, the destination and the departure time of each agent are generated according to the model. The iteration phase then starts, agents interact with other agents according to the defined behavior model. Agents select a route from the origin to the destination, change speed and select a lane. Finally when an agent reaches its destination, it is removed from the simulation. Megaffic also creates new agents at their origin whenever their departure time is reached.

In our research, the computation model and the agent behavior are based on Megaffic. Thus each agent has a tentative path, driver preference and origin-destination data in advance.

Other microscopic traffic simulators have been proposed in the past, with the objective of simulating large-scale urban areas. Bragard et al. [11] have defined dSUMO, a distributed version of the free and open traffic simulation suite SUMO [9]. SUMO offers tools and algorithms for developers of transportation scenarios and helps them to import road networks (e.g., extracted from OpenStreeMap data [19]), generate traffic and define some of the parameters such as, vehicle following model and driving behaviors. dSUMO runs several instances of SUMO on different computing nodes and manages the interactions between them: mostly the transfer of vehicles from one node to another and the border between nodes (a.k.a., interest management). Another example is Matsim [4], which achieves a large-scale microscopic traffic simulation on a single computer. For example, the traffic in all of Switzerland was simulated using Matsim, but some of the details were omitted from their simulation models for scalability [25].

B. Pay-as-you-go Cost Model for Computing Resources

Renting computing resources as you need them or as you use them, sometimes referred to as *pay-as-you-go*, is a recent trend in computing that started with the advent of cloud computing. Amazon, through its Amazon Elastic Compute Cloud (Amazon EC2 [1]) service, has been a pioneer in this area and offers to run computing nodes (virtual machines) and pay only for the time for which they are in use (i.e., the time between their launch and release. This economic model for computing infrastructure has spread widely since and it is now also implemented in the older high performance computing world.

1) IaaS in the Cloud:

Infratructure as a Service, or IaaS, is one of the way to consume services in the Cloud. It consists in virtual machines, accessible over the Internet: some specific commands (or APIs) allow to start virtual machines, interact with them and shut them down. In Amazon EC2, you can rent virtual machine environment in hourly units. You can choose memory size and networks bandwidth as well as the performance of CPU. You can also go for supercomputer-like characteristics, with high performance CPU and sometimes GPU, and high bandwidth network.

There are other IaaS with finer cost models than Amazon EC2, such as Microsoft Azure [5], Google Compute Engine [2] and Rackspace [6] that charge by minutes. Thus you can increase and decrease the resources and can reduce the total cost by using resources according to your exact needs.

2) Supercomputers with Flexible Cost Models:

Recently, supercomputer have entered the pay-as-you-go model that has been popular in the Cloud. We give here two examples that we use in our work (not in this paper).

Tsubame 2.5 [8] is one of the fastest supercomputer in Japan: it is ranked eleventh in the of TOP 500 supercomputers ranking in November 2013, with peak performance of 5609.4 TFLOPS [7]. It is operated by the Global Science Information and Computing Center (GSIC) of the Tokyo Institute of Technology. Tsubame includes various kind of machines, from single node with middle performance to large number of nodes with high performance including GPU; you are billed by the second for your work on Tsubame.

K computer [3] is also one of the fastest supercomputer in Japan: it was ranked fourth at TOP 500 in November 2013 with peak performance of 11280.4 TFLOPS [7]. It is produced by Fujitsu and operated by RIKEN Advanced Institute for Computational Science (AICS). The billing interval is an hour.

III. System Overview

This section focuses on a description of our system for adaptive VM provisioning for large-scale agent-based traffic simulations.

A. General Description

Our system shares many of its concerns with other large-scale computing systems on commodity machines (e.g., the Cloud or supercomputers) and we do not pretend we built it from scratch. However, our solution systematises various approaches and methods and proposes to organise them in a coherent set of modules, addressing what we consider the three main concerns of such a large-scale computing on a variable number of running nodes:

- *Workload prediction*: This is logically the first step of any distributed computing when the number of machines can vary depending on users' preferences and objectives, and the context (e.g., variability in price or complexity of the processing). An accurate prediction of what is required/available overall and for each machine will help determine the characteristics and number of machines needed.
- *Data partitioning*: Given users' preferences and objectives, the question is now to determine the best plan, in terms of number of machines to run and partitioning

of the data. This covers a series of crucial issues that have been addressed extensively in the literature.

- *Resource management*: This is the last concern, the more applied too. It regards the implementation of the plan defined at the previous step in accordance with the objectives of the users. The issue here is that the implementation of the plan needs to be looked after well or it will lead to inefficient solutions that do not scale.

Figures 1 and 2 give a more detailed description of how we want to achieve our objective of adaptive resource provisioning for large-scale distributed traffic simulation. Figure 1 shows the fundamental elements of our system, and in particular the three modules that manage the resources provisioning: the *workload predictor*, the *state partitioner* and the *resource controller*. The system is totally composed by master-worker architecture. The master node controls resource provisioning and simulation itself. The workers process the actual simulating scenario. Our framework is constructed as a extension of the master in the existing simulator.

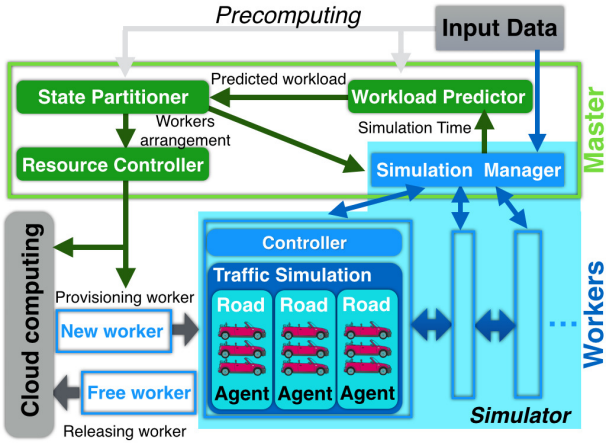

Fig. 1: System Overview

Figure 2 shows the order in which each module is applied by the master node. Before processing simulation scenarios, in the workload predictor and the state partitioner, the precomputation occurs to predict the simulation workload and to partition the simulation state in advance. After that, the actual simulation starts, which consists of the series of iterations. In the iteration, the simulator's master passes the simulation progress to the modules. According to the progress information, the modules defines appropriate resources in the next iteration.

In the next subsections we give a detailed overview of the three main modules in our solution.

B. Workload Predictor

The workload predictor predicts next steps' workload of each worker based on the precomputed result of the input trips data, such as, departing time and precise route of each vehicle. It is then possible for the predictor to totally predict the workload in advance by using only the input trip data if we assume followings:

```
1  // Initializing and precomputing
2  TrafficSimulator trafficSim = new TrafficSimulator();
3  WorkloadPredictor predictor
4    = new WorkloadPredicter(inputScenairoData);
5  StatePartitioner statePartitioner
6    = new StatePartitioner(inputScenarioData);
7  ResourceController controller = new ResourceController();
8  // Iteration
9  while (time < TIMETOFINISH) {
10   WorkloadInfo workloadInfo
11     = predictor.predictWorkload(time);
12   ResourceArrangement arrangement
13     = statePartitioner.partition(workloadInfo);
14   resourceController.provision(arrangement);
15   CrossPointsMeta cps = trafficSim.migrate(arrangement);
16   trafficSim.refreshCPsMetaData(cps);
17   time = trafficSim.runIteration();
18   resourceController.release(cps);
19 }
```

Fig. 2: The Computation Flow in Master

- The workload strongly depends on the number of departing vehicles.
- Individual vehicles' trip patterns do not impact the total workload.
- Only statistical properties (e.g., total number of vehicles, average trip path length) impact the workload.

C. Simulation State Partitioner

The simulation state partitioner decides on the number and characteristics of simulation workers required. Partitioning a model's data for distributed simulation is another critical issue that attracted a lot of research in the area (see for instance [29]). Recently, some flexible partitioning has been proposed by Bragard et al. [12], [13] and they claim it is applicable to traffic simulation. We believe it would be possible to use such adaptive partitioning in our own system and we actually consider it for some future work.

In the current version of our system, we assume the following property about the input data (see Section V for more details):

- The distribution of vehicles' departing points is uniform, thus the workload pattern depends on only the number of departing vehicles and not their location (in practice we use a randomization).
- We run the simulation many times with very minor changes, which does not impact the total performance. Thus, we can totally find the execution time from the second time. For example, we simulate what happens if the new road are added, what happens if the road are blocked, and so on. And the execution time is hardly changed in such minor changes.

Based on these assumption, we can define the appropriate number of workers at each iteration in following ways. In advance, we find elapsed time saturation numbers of workers by analysing the strong scaling with each workload pattern. For

example, if the elapsed time does not improve by adding the worker from 4 to 5 with 500 departing vehicles, the saturation number of 500 departing vehicles is 4. After that, in the traffic simulation, the partitioner receives predicted workload from the predictor and defines the appropriate number of workers. According to the number of workers, the partitioner partitions the road map by the same way as Megaffic, which use *k*-ways graph partitioning algorithms.

D. Resource Controller

The resource controller controls physical or virtual machines, depending on the environment. What we have in mind is an Infrastructure as a Service (IaaS) model with virtual machines provisioned when needed and released when not required anymore; but obviously other systems can be envisaged.

In this IaaS context, the controller gets its orders from the partitioner and either launches new machines via the relevant IaaS API if it requires the extra machine, or releases unused machines. Resource provisioning procedures such as starting VMs and setting up their configuration is independent of the simulation. Therefore it can be done in the background.

It is important to notice that the resource controller works better if the prediction is made in advance, and not at the last minute (e.g., need for new VM decided when more computing power is already needed and not before it is required). The time required for provisioning and configuring machines has to be included in the state partitioning plan.

IV. Efficient Simulation State Migration for Traffic Simulation

This section focuses on synchronisation and communication for a large-scale agent-based traffic simulation such as Megaffic. We first detail the synchronisation mechanisms in Megaffic. Then we discuss some naive, synchronous, solution which does not destroy consistency of the simulating result. After that, we describe two techniques to make the migration faster than using naive synchronous migration: *asynchronous* and *staged asynchronous migration*. The key idea of the techniques is overlapping the migration cost with the simulation execution, where there is workload imbalance between workers and some workers idle until all workers finish processing.

A. Synchronisation and Migration in Megaffic

As we said above, our system is based on a Java implementation of Megaffic (see also Section V for more details). Megaffic uses a *bulk synchronous* processing, with one iteration representing 10 seconds in the real world. Each iteration consists of two phases, *individual tasks* first and then *communication* of vehicles between workers. In the individual task phase, each worker processes vehicles situated on roads assigned to it and computes where each vehicle shall be at the next iteration. After the vehilces' positions computation comes the communication phase: vehicles are exchanged between workers according to where vehicles should be at the next iteration point. In order to keep the simulation internal clock synchronised and keep consistency for the simulating result, a barrier synchronization occurs among the workers at the end of individual task phase and of communication phase. See for example iteration 1 in Figure 3 for an illustration of the three steps mentioned above: individual tasks (computation of vehicles' position), communication and synchronisation barrier.

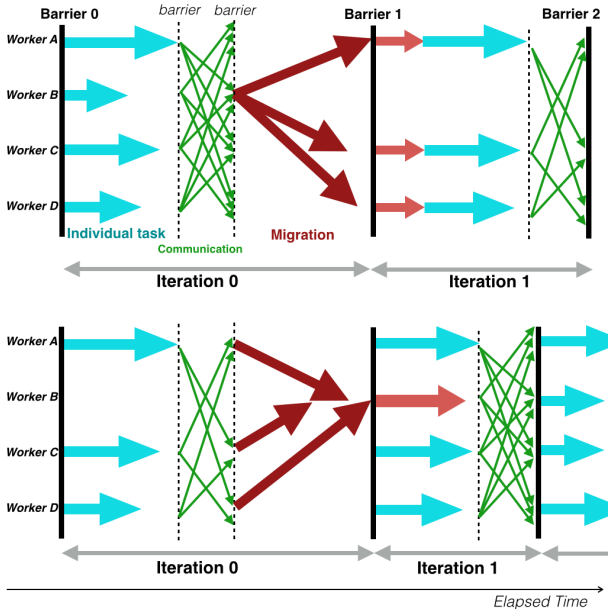

Fig. 3: Naive Synchronous Migration

Beside, in order to change the number of workers during the simulation while keeping consistency of the simulating result, we need to migrate part of the simulation state between workers. But migration cost is high as it requires a lot of communication between workers, and serialization and deserialization of simulation objects, which increases execution time and CPU cost. Suppose C_a is the cost to migrate 1 agent, C_r is the cost to migrate 1 road, $N_a(i)$ is the number of migrating agents in $road_i$, and N_r is the number of migrating roads. The total cost of migration C_{total} is:

$$C_{total} = C_r \times Nr + \sum_{i=0,1,2,..Nr} \{C_a \times N_a(i)\} \quad (1)$$

Each road migration is independent, thus you can easily parallelize it and if there are enough processes and enough network bandwidth, the execution time is:

$$T_{paraTotal} = \max_{i \in \{0,1,2,\cdots,N_r\}} \{Time(C_r + N_a(i) \times C_a), \} \quad (2)$$

where $Time(x)$ gives the time required to process the corresponding cost.

B. Basic Ideas for a Migration Process in Megaffic

Now, to migrate the simulation state safely, there are two requirements.

First, migrations cannot go at a more fine-grained level than the road segment level. In Megaffic, the simulation state is represented by a set of roads with cross points and the simulation is processed by road segments. Thus if a road is divided by the migration, the simulation state is changed and the minimum unit of processing is changed, which we do not want here.

Second, migrations have to occur after all vehicles are exchanged between workers during the communication phase, i.e., when no more vehicle needs to move between workers. Otherwise, if there are still vehicles which need to move while workers and partitions are being modified, there are clear risks of inconsistencies.

In our system, the resource controller has to wait for all the workers to finish the synchronisation of all the vehicles. The resource controller can then repartition the model and crosspoints, roads and vehicles are exchanged between workers before the next iteration. Figure 3 shows a basic example of such a migration process, that we call *naive synchronous migration*: the scenario on the top shows a passage from 4 workers to 3 workers, while the scenario on the bottom of the figure represents a simulation with 3 workers that grows to 4 workers. In both scenarios the migration happens after the processing of individual tasks and the communication/synchronisation. This makes sure that everything is coherent in the simulation model, and that, for instance, worker B's data is given to the other workers correctly (top scenario) or that workers A, C and D's data is shared with worker B (bottom scenario).

C. Basic and Staged Asynchronous Migration

To reduce the migration overhead that we can observe in Figure 3 we propose two *asynchronous migration* schemes that overlap migration and individual tasks' processing, hence limiting the waiting time of the different workers.

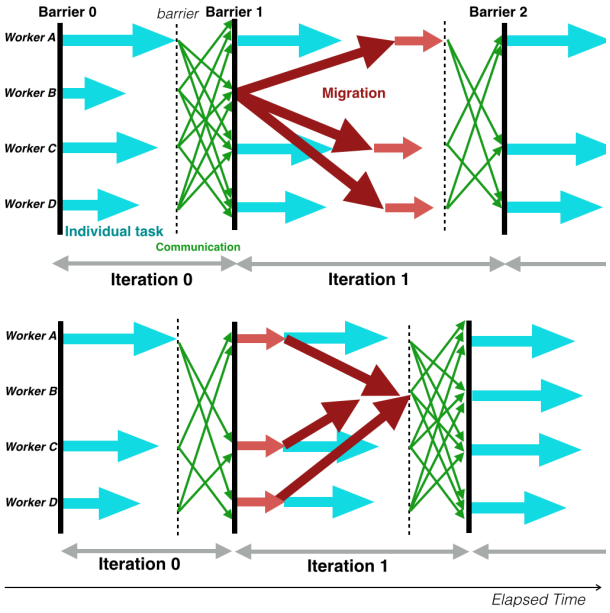

Fig. 4: Asynchronous Migration

A basic asynchronous migration (see Figure 4) (i) happens after the individual tasks' processing when the number of workers decreases and gives to each worker the new data to process before the synchronisation (top example); or (ii) happens before each workers' own processing and after the processing of the data that is going to be exchanged, when a new worker is introduced in the system (bottom example). In both cases the processing of the new/old data is done in parallel.

A *staged asynchronous migration* is another scheme where the migration is processed step by step during several iterations. This allows the migration to overlap better with the individual tasks' processing and to be faster. Figure5 shows the example of a staged asynchronous migration divided in 3 iterations.

V. Evaluation

This section has two distinct objectives:

- The first one is a study of the performance of the distributed simulation when the number of workers varies, through a strong scalability evaluation. This gives us an interesting insight into what the simulator is able to cope with depending on the number of workers and the load.
- The second objective consists in a performance evaluation of the adaptation mechanism, in particular the migration techniques and the gains in simulation time offered by the modification of the number of workers.

A. System Setups

Our simulator is based on a Java implementation of Megaffic. The Java standard serialization is known to be a bottleneck when there are a lot of serialisation; we then decided to use Messagepack (version 0.6.6), an efficient binary serialization format. Moreover, as serialisation in our system occurs during the migration (of states between workers) and the communication (of vehicles between workers), we need an efficient communication library, to make sure the performance of the simulation is not impacted. We then use Netty (version 3.2), an NIO communication framework that has good throughput and low latency.

We use Google Compute Engine as our Infrastructure as a Service solution (see Table I), with a maximum of 17 n1–standard–1 instances (16 workers and 1 master). The n1–standard–1 instances have 1 virtual CPU and 3.8 GB of Memory. All instances belong to the same zone (Europe-West1-b) and run a CentOS–6–v20140619 image with Java SE 7 installed.

Service	Google Compute Engine
Instance Type	n1–standard–1 (1 vCPU, 3.8GB Mem)
Zone	Europe-West1-b
OS image	CentOS–6–v20140619
Java VM	Java SE 7 Update 65 (OpenJDK 64-Bit Server)

TABLE I: Virtual machines' configurations.

Table II shows the simulation setups of our main scenario. It consists of 24 hrs of traffic data of Tokyo collected by the MLIT (Ministry of Land, Infrastructure, Transport and Tourism) in 2011.

Figure 6 shows the typical number of departing vehicles per 10 seconds of real time in Tokyo (it corresponds to the workload for the simulation). We use the full road network of Tokyo, i.e., 770,192 cross points and 2,089,374 road segments. One thing that is missing in the data collected by the MLIT is the OD matrix (origin and destination for each trip). We generate it randomly, as well as the exact route of every

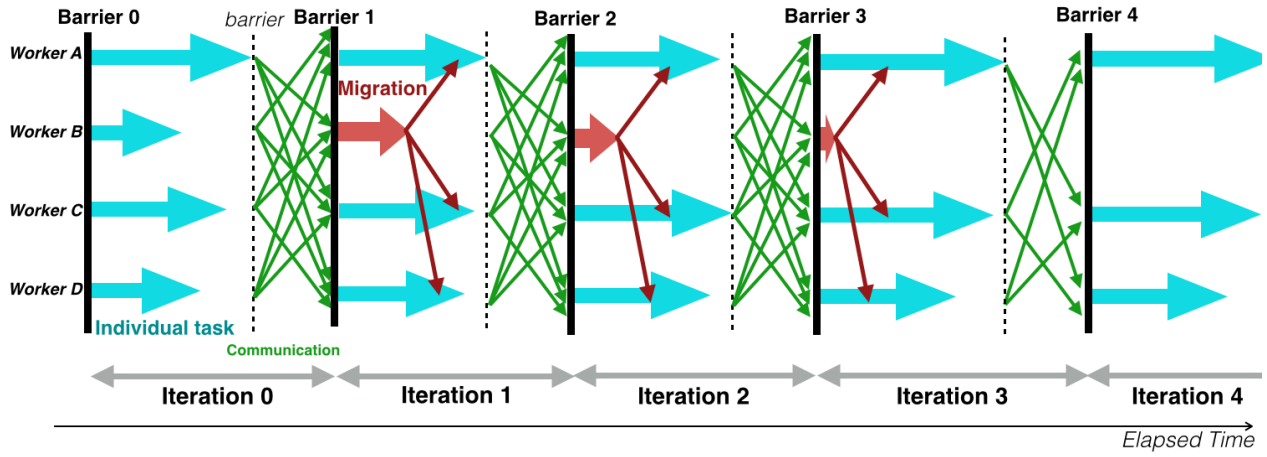

Fig. 5: Staged Asynchronous Migration

Road Map	Full road network of Tokyo
– # of Cross Points	770,192
– # of Roads	2,089,374
Scenario	24 hrs of Tokyo's traffic
– Sum of Departing Vehicles	10,000,000
– Trips Origin/Destination	Random
– Time Unit	10 sec/iteration

TABLE II: Scenario's input data.

vehicle. Each iteration of the simulation corresponds to 10 seconds of real time, hence the 24 hour scenario corresponds to 8,640 iterations in the simulator.

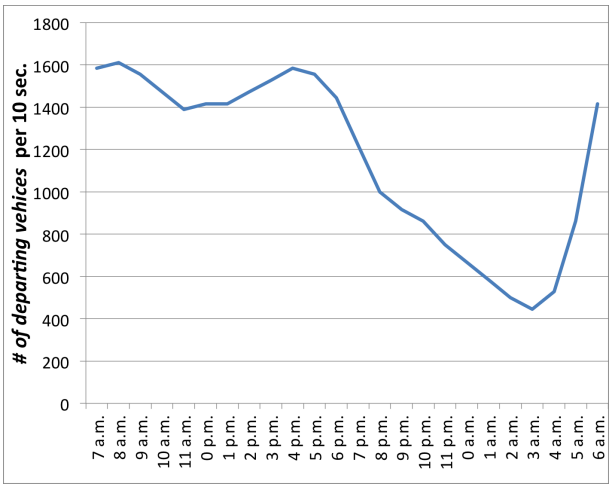

Fig. 6: Number of departing vehicles in a typical 24 hr scenario of Tokyo.

B. Evaluation of the Optimal Number of Workers

Here we want to study the impact of the number of vehicles (i.e., the load) on the system depending on the number of workers. This will give us an evaluation of the optimal number of workers required by the system depending on the load. We create a strong scaling evaluation of the system, with 5 to 16 workers, and 400 to 1,700 departing vehicles at each step (the extreme values given by the scenario, see Figure 6). The results of this evaluation are given in Figure 7, with the corresponding saturation points, i.e., when adding workers does not impact performance anymore. Note that saturation points for more than 1,000 vehicles is higher than the 16 workers limit that we have so we stick to this value of 16. From that study we extract the optimal values of number of workers per load as given in Table III.

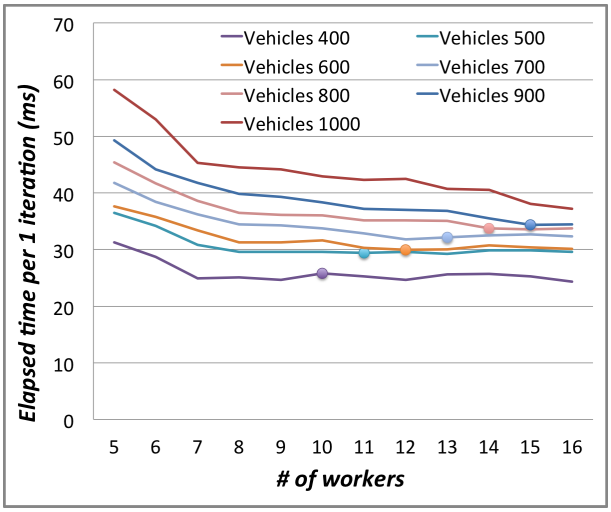

Fig. 7: Strong scaling evaluation and saturation points for each rate of departing vehicles.

C. Performance Evaluation of the System

Here we evaluate the performance of the migration techniques and our system as a whole using the number of departing vehicles given by the Tokyo 2011 scenario.

# of departing vehicles	# of workers
400 to 499	10
500 to 599	11
600 to 699	12
700 to 799	13
800 to 899	14
900 to 999	15
1000 to 1700	16

TABLE III: # of workers suggested depending on the load.

Figure 8 shows the average elapsed time of one iteration (including staged asynchronous migration) when changing the number of workers. In staged migration, the more migrating smaller parts, the more overlapped with individual tasks. And finally, each migrating execution is almost all overlapped. For example, in the migration from 10 workers to 11 workers, the execution is approximately fully overlapped in 50 separations and finally became 26 milliseconds at one iteration, which is approximately 3% of naive synchronous migration (876 milliseconds), as mentioned in Section IV. In the migration, even if the staged migration, some overhead for the migration remains. The overhead is influenced by the number of migrating workers (not by the number of vehicles) and the more workers are migrating, the more overhead it causes. Thus, the elapsed time are increasing from "10 to 11" to "15 to 16".

Fig. 8: Time of one iteration including (staged) migration.

Figure 9 shows the cost performance of the system by 50 separated staged migration compared to the non elastic way with 10, 12, 14 and 16 workers. "Cost" is defined as the sum of instances' running time from starting to releasing. In the 24 hour scenario, the system can reduce the cost of the simulation by up to 12 % with 16 workers compared to a non elastic way. If we focus to the 8 p.m. to 6 a.m scenario, where the number of departing vehicles changes a lot, the reduction is even bigger (23%) compared to the non elastic execution with 16 workers.

VI. Related Work

The potential of the high performance application on the cloud computing environment has been investigated in several

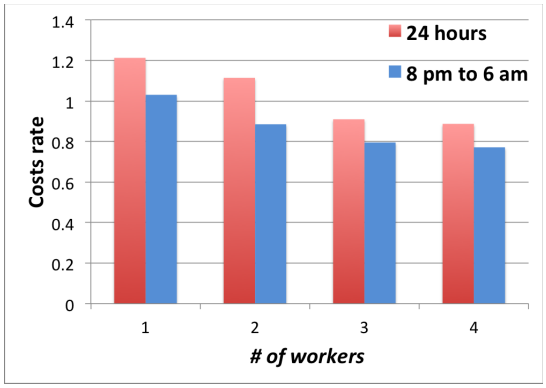

Fig. 9: Cost reduction.

groups. For example in [20], they analyzed the typical scientific applications on the Amazon EC2 and they compared the performance with their supercomputer and computer cluster. They showed the detail performance characteristic and limitation of Amazon EC2 compared to the supercomputer and computer cluster. Also as with our proposal in [16], they proposed the payment cost reducing way for scientific application. They used the resource provisioning based on the task scheduling technique. They proposed the two novel heuristic scheduling algorithms for reducing the payment cost of scientific application.

Shengming Li, et al. [21] proposed the multi-VM provisioning technique similar to our approach. Their proposal is a combination of TBAMP (Time-based Billing Aware Multi-VM Provisioning algorithm) and ARIMA (autoregressive integrated moving average) and enables to reduce the cloud computing payment cost. This provisioning method is composed of prediction module and partitioning module, which is very similar to our proposed way. The main difference between our research and their research is that in our traffic simulation, the workers have simulation state during execution while their target applications do not have any state in process, for example, cache server or web server. In such application, it is not important to keep consistency between workers or each worker has same state during execution. In contrast, our traffic simulation requires to keep consistency even if the number of worker is changed during execution, and to move the simulation state of the workers.

Recently, a lot of works about efficient computational resource allocation are proposed by the resource provider. For example, in the research [17], they proposed a new method to reduce the energy cost and provision resources as long as keeping some SLAs. This technique consists of prediction workload scheme and reactive allocating scheme like ours. In this research they divide the workloads into two types, coarse time scale (e.g., hours or days) and finer time scales (e.g., minutes). And then if the workload is typed as "coarse", resources are provisioned based on the result of prediction; if it is typed as "finer", resources are provisioned based on reactive allocation technique. There are several similar approaches using combination of prediction and reactive provisioning [10] [18] [28]. Unlike these research, our research is for users or customers of computer resources, not for the computer resource provider. Therefore, there are

some differences in assumptions. First, we can use application specific and semantic data. For example, we can get the number of vehicles in advance. Thus we can predict based on traffic semantics such as "it is the noon" or "it is the early morning". These accurate the more detail and easier prediction than using general profiling. Second, we assume unlimited use of resources. We can provision as many machines as we pay.

VII. Conclusion

In this research, we presented the framework for adaptive resource provisioning in traffic simulation, which provides the method to reduce the utilization cost of computer resources. Also, we proposed the technique to migrate the simulation objects efficiently by overlapping the simulating processes.

For future works, the modules of the system should be sophisticated. In the predictor, we should make the prediction more accurate with mathematical ways or machine learning techniques. In this proposal, we assumed that the simulation workload depends on only statical property of departing vehicles. Such assumption is not accurate, for example, in the case that there is traffic congestion in the simulation. In such cases, individual vehicle trips impact the total performance. To solve the problem, for example, we can predict traffic flows more precisely by using ARIMA (autoregressive integrated moving average), which has been proposed in previous researches. To sophisticate the simulation state partitioner, we should solve the optimization problem. Actually, the partition problem can be represented as the cost minimum problems subject to the admissible maximum execution time, the execution time of individual tasks, the execution time of vehicle communication and the elapsed time of migration. To solve this problem, for example, by using incremental graph partitioning technique, we can optimize the cost performance more accurately and effectively than this proposal.

Acknowledgment

This work was supported, in part, by JST CREST and JSPS KAKENHI, grant numbers 25700008 and 26540161, by Science Foundation Ireland grant 10/CE/I1855 to Lero (the Irish Software Engineering Research Centre, www.lero.ie) and by Science Foundation Ireland Industry Fellowship grant 13/IF/12789.

References

[1] Amazon EC2. https://aws.amazon.com/ec2/.

[2] Google Compute Engine. https://cloud.google.com/products/compute-engine/.

[3] K computer. http://www.kcomputer.jp/en/.

[4] Matsim. http://www.matsim.org/.

[5] Microsft Asure. http://azure.microsoft.com/.

[6] Rackspace Cloud. http://www.rackspace.com/.

[7] TOP500. http://www.top500.org/.

[8] Tsubame 2.5. http://www.gsic.titech.ac.jp/en/tsubame/.

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

[10] M. N. Bennani and D. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC*, pages 229–240. IEEE, 2005.

[11] Q. Bragard, A. Ventresque, and L. Murphy. dSUMO: towards a distributed SUMO. In *SUMO Conference*, 2013.

[12] Q. Bragard, A. Ventresque, and L. Murphy. Global Dynamic Load-Balancing for Decentralised Distributed Simulation. WSC, 2014.

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

[14] P. Charles, C. Grothoff, V. Saraswat, C. Donawa, A. Kielstra, K. Ebcioglu, C. Von Praun, and V. Sarkar. X10: an object-oriented approach to non-uniform cluster computing. *ACM SIGPLAN Notices*, 40(10):519–538, 2005.

[15] N. Collier and M. North. *Repast HPC: A platform for large-scale agent-based modeling*. Wiley, 2011.

[16] H. M. Fard, T. Fahringer, and R. Prodan. Budget-constrained resource provisioning for scientific applications in clouds. In *CloudCom*, volume 1, pages 315–322. IEEE, 2013.

[17] A. Gandhi, Y. Chen, D. Gmach, M. Arlitt, and M. Marwah. Hybrid resource provisioning for minimizing data center SLA violations and power consumption. *Sustainable Computing: Informatics and Systems*, 2(2):91 – 104, 2012.

[18] D. Gmach, S. Krompass, A. Scholz, M. Wimmer, and A. Kemper. Adaptive quality of service management for enterprise services. *ACM Transactions on the Web*, 2(1):8, 2008.

[19] M. Haklay and P Weber. Openstreetmap: User-generated street maps. *Pervasive Computing*, 7(4):12–18, October 2008.

[20] K. Jackson, L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H. Wasserman J, and N. Wright J. Performance analysis of high performance computing applications on the amazon web services cloud. In *CloudCom*, pages 159–168. IEEE, 2010.

[21] S. Li, Y. Wang, X. Qiu, D. Wang, and L. Wang. A workload prediction-based multi-vm provisioning mechanism in cloud computing. In *Asia-Pacific Network Operations and Management Symposium*, pages 1–6. IEEE, 2013.

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

[23] T. Osogami, T. Imamichi, H. Mizuta, T. Suzumura, and T. Ide. Toward simulating entire cities with behavioral models of traffic. *IBM Journal of Research and Development*, 57(5):6:1–6:10, Sept 2013.

[24] M. Paolucci and et al. Towards a living earth simulator. *The European Physical Journal Special Topics*, 214(1):77–108, 2012.

[25] B. Raney, N. Cetin, A. Völlmy, M. Vrtic, K. Axhausen, and K. Nagel. An agent-based microsimulation model of swiss travel: First results. *Networks and Spatial Economics*, 3(1):23–41, 2003.

[26] T. Suzumura and H. Kanezashi. Accelerating large-scale distributed traffic simulation with adaptive synchronization method. In *ITS World Congress*, 2013.

[27] T. Suzumura, S. Kato, T. Imamichi, M. Takeuchi, H. Kanezashi, T. Ide, and T. Onodera. X10-based massive parallel large-scale traffic flow simulation. In *ACM SIGPLAN X10 Workshop*, page 3. ACM, 2012.

[28] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier internet services and its applications. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 291–302. ACM, 2005.

[29] A. Ventresque, Q. Bragard, E. S. Liu, D. Nowak, L. Murphy, G. Theodoropoulos, and J. Qi Liu. SParTSim: A space partitioning guided by road network for distributed traffic simulations. In *DS-RT*, pages 202–209. IEEE, 2012.