Towards a Framework for Adaptive Resource Provisioning in Large-Scale Distributed Agent-Based Simulation

Masatoshi Hanai[1,2], Toyotaro Suzumura[2,4], Anthony Ventresque[2,3,4], and Kazuyuki Shudo[1]

[1] Tokyo Institute of Technology, Dept. of Mathematical and Computing Sciences 2-12-1 Ookayama, Meguro, Tokyo, 152-8552 Japan
{hanai.aa,shudo}@{m,is}.titech.ac.jp
[2] School of Computer Science and Informatics, University College Dublin, Ireland
anthony.ventresque@ucd.ie
[3] Lero, the Irish Software Engineering Research Centre
[4] Smarter Cities Technology Centre, IBM Research, Damastown Industrial Estate, Mulhuddart, Dublin 15, Ireland
suzumura@acm.org

Abstract. Large scale distributed agent-based simulations run on several computing units (e.g., virtual machines in the Cloud, nodes in a supercomputer). Classically, these systems try to (re-)load-balance the nodes as overloaded nodes slow down the process.However another challenge in large scale distributed simulations is that the *overall load evolves*. In this paper we leverage on commodity computing to adapt resource provisioning (number of computing units) to the load during the execution of the simulation. We also propose an asynchronous migration mechanism that migrate workload between computing nodes efficiently when nodes wait for synchronisation barriers to happen. We validate our implementation on a scenario simulating one day of vehicular traffic in Tokyo, running on 2 to 8 machines depending on the demand. Our evaluation shows a 26% reduction in data migration time compared to a naive migration approach between computing units.

1 Introduction

Agent-based simulation is an important field of research that has led to important and promising findings in areas such as transportation, environmental protection and economy [17]. Bringing agent-based simulations up to the scale required by large systems (e.g., large urban areas, social systems) is a challenging problem and several methods and simulators have been proposed [10,20]. In short, large scale simulations need to run on several computing nodes in parallel to speed-up their processing, and the challenges are usually load-balancing and minimisation of communication between nodes.

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part I, LNCS 8805, pp. 430–439, 2014.

In this paper we are interested in a different problem: *how can we efficiently adapt the number of computing resources to the load?* Our idea comes from two simple observations: (i) overall simulation load evolves over time [19], e.g., for a urban traffic simulation, there are less vehicles to process during the night than during peak commuting hours; and (ii) the number of computing nodes required during low and peak demands should vary accordingly. Figure 1 shows the traffic pattern (number of cars) of a day in Tokyo; in the system we describe below (Megaffic) 100,000 vehicles per computing node is a good number and it is clear from the figure that while we sometimes need 8 machines to run the simulation, there are other moments when less are required.

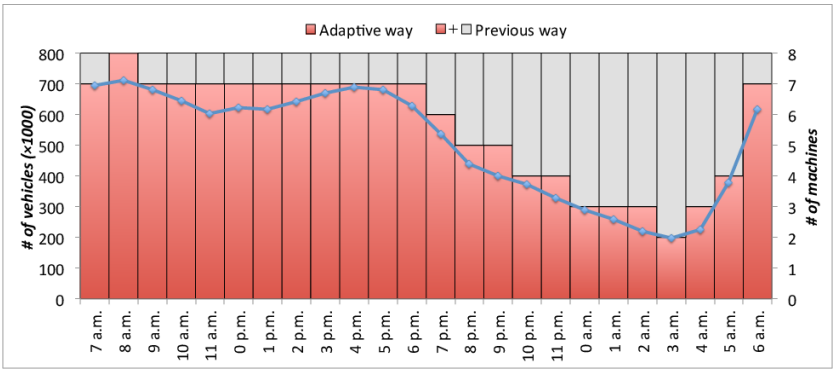

Fig. 1. Typical day of traffic (number of vehicles) in Tokyo and number of machines required to simulate it with our adaptive system

This provisioning and releasing of resources according to demand is challenging in a high-performance computing setting, i.e., the classical context for large scale simulations. However the situation is different nowadays, with the advent of pay-as-you-go mechanisms in both Cloud computing and supercomputers (see Section 2 for details). This is why we propose in this paper a new method to adapt resource provisioning, which enables to increase and decrease the number of machines during execution and achieves an efficient utilization of computational resources. Our main contribution are:

1. The description of a framework which can control the number of machines during execution.
2. An asynchronous migration mechanism for simulated objects, keeping the simulation consistent when the number of executing machines is changing.

The rest of paper is organized as follows: Section 2 describes the context of our research; Section 3 gives a description of our framework, Section 4 details our efficient object migration technique and Section 5 summarizes our implementation decisions; we evaluate the performance of our system in Section 6; Section 7 compares our work to related efforts; we finally conclude in Section 8.

2 Background

This section gives a description of two of the main elements of our work: the traffic simulator and the Cloud computing infrastructure.

2.1 IBM Mega Traffic Simulator

IBM Mega Traffic simulator, or *Megaffic* [15,20,16], is a large-scale distributed traffic flows simulator. Megaffic is built on top of the platform XAXIS (*X10-based Agent eXecutive Infrastructure for Simulation*), a highly scalable multi-agent simulation distributed middleware based on X10, the parallel computing programming language developed by IBM [9]. Megaffic reduces the computation time by precomputing several of the simulation elements, such as route selection, lane selection and vehicle speed.

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 than Megaffic, some large-scale traffic simulators have been proposed in recent years. For instance Bragard et al. recently proposed dSUMO [8], which is a distributed version of SUMO, an open source microscopic traffic simulation system including the simulation application and supporting tools for network imports and demand modeling. 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 [18].

2.2 Pay-as-you-go Cost Model for Computing Resources

Pay-as-you-go is a new model for renting computational resources, where users can decide how much and for how long they want to use specific resources. This model is fine-grained: e.g., renting can be done in the order of seconds [7] or minutes [2,5,6], and it is also flexible, allowing users to define exactly what infrastructure they need, in terms of memory, network or CPU [1]. While pay-as-you-go is often associated with the Cloud computing environment, it is also popular in the area of supercomputers: see for instance the examples of Tsubame [7] and K computer [3].

3 Framework for Adaptive Resource Provisioning

This section describes in details the simulation model used in Megaffic and our system. In particular, we give a presentation of the three main modules of our framework: workload predictor, resource provisioning optimizer, resource controller.

3.1 Simulation Model of Megaffic Simulation

As mentioned in the previous section, we assume the same computational model and the same agent behavior as Megaffic. The simulator gets data such as road definition and vehicle trip description (origin, destination, exact path and departing time). After giving this data to the system, roads and vehicles are distributed to each worker according to the precomputed road map partitions. The roads have as many queues as they have lanes, and each vehicle is inserted in the queues. The system starts iterating then, each iteration representing 1 second of real world time. During each iteration the vehicles are processed to determine where they are at the next iteration. After finishing to process all its vehicles, each worker start communicating with the roads of its neighboring workers. And so on until the end of the simulation.

3.2 Overview of the System

To achieve an efficient resource utilization on a pay-as-you-go system that is fine-grained (in terms of renting time) and flexible, we have mainly three problems to address:

- How do we predict the workload in the next steps to provision exactly the required resource?
- How do we optimize cost given some user's objectives, such as, "as fast as possible simulation (whatever cost)", "as fast as possible but for a cost less than 20 dollars", "as cheap as possible but not lasting more than 1 hour"?
- How do we increase and decrease the number of machines during the simulation according to the objectives we want to optimize?

To solve these problems, our system consists of mainly three components in addition to the traffic simulator itself: the workload predictor, the resource arrangement optimizer and the resource controller. Figure 2 shows an overview our system. The system is built on top of a master-worker architecture where the master controls utilization of the resources and manages the synchronization of the simulation and the meta-data such as the cross points and roads arrangement given to the workers. The workers process the simulation scenario itself.

Workload predictor The workload predictor predicts the workload for next iterations based on its analysis of the input data and some feedback from all workers. It returns the predicted next the workload information for next stepts such as CPU usage, memory usage, network I/O usage, number of outgoing vehicles and incoming vehicles.

Resource provisioning optimizer The resource provisioning optimizer optimizes the cost of running the simulation according to some user-defined requirements. The optimizer gets the result of predicted workload and returns the most efficient arrangement of the simulation state to the workers.

Resource controller The resource controller controls the physical and/or virtual machines environment of the simulation. According to the optimized arrangement, the controller launches new machines via the proper IaaS API if it requires extra machines. After finishing an iteration, the controller also releases the possible unused machine.

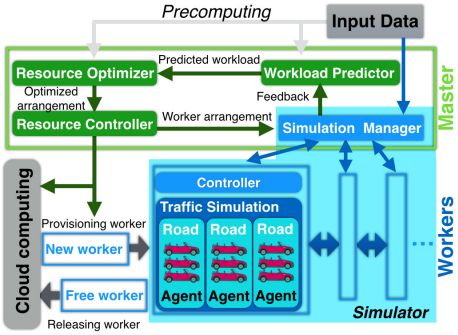

Fig. 2. System overview

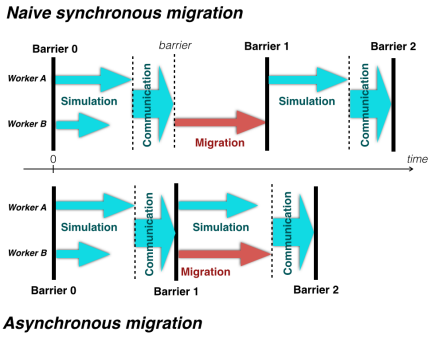

Fig. 3. Asynchronous migration

4 Efficient Migration for Agent-Based Simulation

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 cost to migrate 1 agent, C_r is 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:

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

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:

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

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

The key idea of our proposed solution is to include the migration in the simulation execution itself: the load imbalance between nodes leaving time when workers can exchange workload. To do so, we make the migration *asynchronous*. The problem that needs to be considered carefully is how to keep consistency of simulating results. If migration occurs at arbitrary points during the simulation, the meta data of the simulation state (roads and cross points) may change incorrectly, which could result in inconsistent simulation results. In our system, for appropriate asynchronous migration, the migration occurs only at the beginning of the iteration.

In Megaffic, the whole simulation flow consists of a series of iterations of parallel processing of individual roads and communications alternatively. Between the processing of roads and the communication, the synchronization barrier occurs in all roads. Thus for a consistent asynchronous simulation it is sufficient to synchronise the simulation state metadata at the start of the communication. Figure 3 shows the comparison between a naive way of synchronising and our proposed solution. In the naive synchronous way to migrate, workers with no migration are idle until all migrations are finished. In contrast, in our asynchronous solution, the migrations overlap with the simulation execution and we reduce the total execution time.

5 Implementation

As we already mentioned, we use a master-worker architecture, meaning that a master process controls the resource management and the synchronization of the traffic simulation while workers execute each a part of the traffic simulation. We use ZooKeeper, an open source software for distributed systems configuration, to manage the coordination of the system. ZooKeeper is in charge of naming registry, synchronization mechanism, addressing of machines, role of each machine, and to maintain the status of each machine.

In our traffic simulation, which is mainly implemented in Java, the heaviest process is communication between workers, especially the sending and reception of Java objects. Regarding the communication between objects, we use Messagepack serialization format to serialize simulation object and Netty, nonblocking networking framework, for sending and receiving such serialized simulation objects. As for the predictor, the evolution in the number of departing vehicles at each iteration step is analyzed statically before executing the simulation. Then the predictor returns the sum of the departing vehicles and vehicles to process by each worker. In the optimizer, the worker resource arrangement is optimized based on precomputed partitioned road map data and the number of vehicles at the next step as predicted by the predictor. The road map data is partitioned by a k-ways graph partitioning algorithm [12] using METIS [13] before execution. Other solutions could be used among the various space/graph/road network partitioning algorithms, such as SParTSim [21]. The resource controller uses physical machines instead of the flexible resource provider, just for evaluation purposes. This does not include the shutdown or start up of machines, and the provisioning occurs instantly.

6 Evaluation

In this section, we evaluate our proposed cost reduction method for traffic simulation. We use eight worker machines in total and one master machine, all running Linux 3.8.0. Each machine has two 2.40 GHz Xeon E5620 CPUs and 32GB RAM. We execute the simulation on Java SE 7 update 4 with option *-Xmx16g*. We use the road network of the bay area in Tokyo, which includes 161,364 crosspoints (junctions) and 20,2976 roads. There are a total of 250,000 vehicle trips over a 24 hour period (82,800 steps). The evolution of the number of vehicles during 24 hours are based on the ratio of some traffic data in All of Tokyo collected by the MLIT (Ministry of Land, Infrastructure, Transport and Tourism) in 2011 (see Figure 1 in section 1).

We first conduct evaluations of the scalability and the roads migration time. Figure 4 shows the execution time of simulating 24 hours in Tokyo according to the number of machines. The execution time decreases to as low as 52 % of a single machine's execution time, but plateaus in 6 or 7 machines. This is because the communications cost increases with the number of workers. Figure 5 shows the migration time of roads between two machines. The migration cost increases according to the number of roads and the number of vehicles on the roads.

We then evaluate our proposed asynchronous migration of roads and vehicles for distributed traffic simulation. We adapt the number of workers to the number of new departing vehicles (given before the simulation by the predictor). The optimizer optimizes the roads arrangement every hour (3,600 steps) to keep the number of new vehicles to 2,000. Figure 6 shows the average time for one iteration in synchronous or asynchronous migrations. Note that the migration can be included in the simulation itself (not the communication though) and the time for the asynchronous migration could decrease even more. However, in our traffic simulation scenario, the simulation time is very short compared to the migration time, and the effect of the asynchronous migration does not appear as important as it should be. The asynchronous migration technique could be more effective if the simulation time is large compared to the migration time. Figure

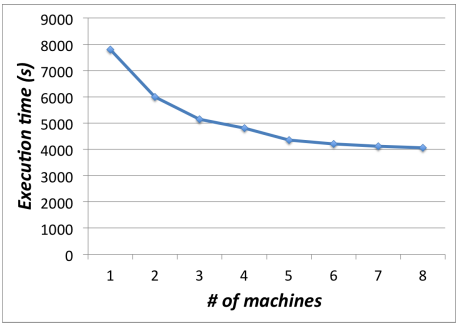

Fig. 4. Scalability of the traffic simulation

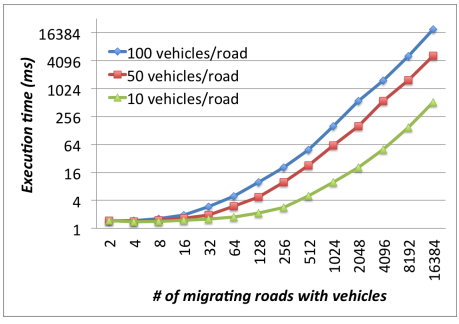

Fig. 5. Migration time for roads

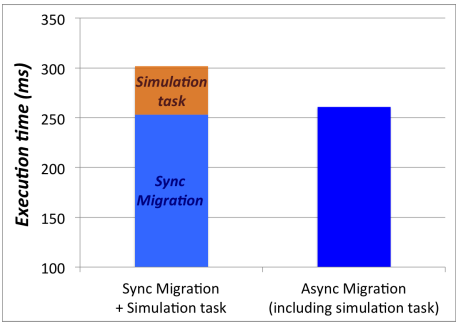

Fig. 6. Overall execution time

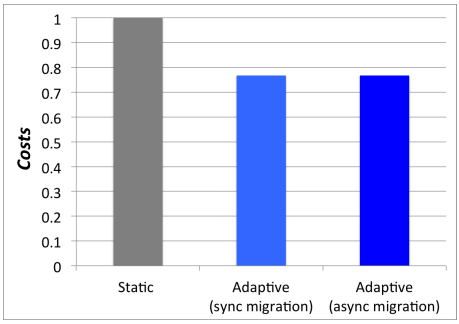

Fig. 7. Cost of migration

7 shows the comparison of costs (i.e., sum of the CPU times including idle time in all machines) between static and adaptive migrations. Adaptive techniques reduce the cost of migration by 26% compared to the static solution.

7 Related Work

Shengming Li, et al. [14] proposed a workload prediction-based multi-VM provisioning mechanism, which contains a multi-VM provisioning technique based on time-based billing aware multi-VM provisioning algorithm (TBAMP) and a workload prediction technique based on ARIMA (autoregressive integrated moving average). This technique achieves effective rental cost saving in the Cloud and consists of a prediction module and an optimizing module, like our work. However, unlike our approach, this technique does not take machine communication into consideration. This is ok for applications without any machine communication, like web servers or cache servers, which can be linearly faster with more machines and do not require to keep consistency. However this is not the case in the agent-based simulation domain. We then need to take network topology (e.g., communication) into consideration to keep or even improve the performance, and we need a migration solution of the simulation states for consistency.

Efficient resource allocation of computer resources is a challenging research topic. There are a lot of work on efficient resource allocation using resource provider. For example, Gandhi et al. [11] proposed a method to reduce the energy cost while meeting SLAs by using a workload prediction and a reactive allocations modules. This approach is similar to ours but from a different perspective: we are interested in users of computational resources while they focus on resource providers. Thus, we do not share the same assumptions. First, we can use some application specific and some semantic information for prediction. For instance, we can get some input about the number of vehicles and the road map data, and we can predict based on traffic semantics such as "it is noon" or

"it is early in the morning". This enables a more detailed and accurate prediction than just using general profiling of machines. Second, we do not take any resource limit into consideration and assume we can provision as many machines as we need (as long as we pay for them).

8 Conclusion

In this paper, we presented a framework for adaptive resource provisioning of traffic simulations, providing a method to reduce the utilization cost of computing resources. We also proposed a technique to migrate the simulation objects efficiently by overlapping the simulating processes. We simulated the traffic of Tokyo with 8 commodity servers and we confirmed that our method can save up to 26 % of the simulation costs without impacting the simulation results.

We would like to improve some of the components of our framework as future work. The prediction could be more accurate by using some mathematical or machine learning techniques. For example, we can predict traffic flows more precisely using ARIMA. The optimizer could benefit from a dynamic graph partitioning and would certainly optimize the resource assignment for next steps. For example, by using incremental graph partitioning technique, we can adapt the resource assignment more effectively to the information given by moving vehicles. Finally we need to implement and evaluate fully the Cloud computing environment in the resource controller module.

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 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. Tsubame 2.5, `http://www.gsic.titech.ac.jp/en/tsubame/`
8. Bragard, Q., Ventresque, A., Murphy, L.: dSUMO: towards a distributed SUMO. In: SUMO Conference (2013)
9. Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., Von Praun, C., Sarkar, V.: X10: an object-oriented approach to non-uniform cluster computing. ACM SIGPLAN Notices 40(10), 519–538 (2005)
10. Collier, N., North, M.: Repast HPC: A platform for large-scale agent-based modeling. Wiley (2011)

11. Gandhi, A., Chen, Y., Gmach, D., Arlitt, M., Marwah, M.: Minimizing data center sla violations and power consumption via hybrid resource provisioning. In: Green Computing Conference and Workshops, pp. 1–8. IEEE (2011)
12. Karypis, G., Kumar, V.: Multilevel k-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed Computing 48(1), 96–129 (1998)
13. Karypis, G., Kumar, V.: METIS - a software package for partitioning unstructured graphs, meshes, and computing fill-reducing orderings of sparse matrices-version 5.0. University of Minnesota (2011)
14. Li, S., Wang, Y., Qiu, X., Wang, D., Wang, L.: A workload prediction-based multi-vm provisioning mechanism in cloud computing. In: Asia-Pacific Network Operations and Management Symposium, pp. 1–6. IEEE (2013)
15. Osogami, T., Imamichi, T., Mizuta, H., Morimura, T., Raymond, R., Suzumura, T., Takahashi, R., Ide, T.: IBM Mega Traffic Simulator. Technical report, Technical Report RT0896, IBM Research–Tokyo (2012)
16. Osogami, T., Imamichi, T., Mizuta, H., Suzumura, T., Ide, T.: Toward simulating entire cities with behavioral models of traffic. IBM Journal of Research and Development 57(5), 1–6 (2013)
17. Paolucci, M., et al.: Towards a living earth simulator. The European Physical Journal Special Topics 214(1), 77–108 (2012)
18. Raney, B., Cetin, N., Völlmy, A., Vrtic, M., Axhausen, K., Nagel, K.: An agent-based microsimulation model of swiss travel: First results. Networks and Spatial Economics 3(1), 23–41 (2003)
19. Suzumura, T., Kanezashi, H.: Accelerating large-scale distributed traffic simulation with adaptive synchronization method. In: ITS World Congress (2013)
20. Suzumura, T., Kato, S., Imamichi, T., Takeuchi, M., Kanezashi, H., Ide, T., Onodera, T.: X10-based massive parallel large-scale traffic flow simulation. In: ACM SIGPLAN X10 Workshop, p. 3. ACM (2012)
21. Ventresque, A., Bragard, Q., Liu, E.S., Nowak, D., Murphy, L., Theodoropoulos, G., Liu, J.Q.: SParTSim: A space partitioning guided by road network for distributed traffic simulations. In: DS-RT, pp. 202–209. IEEE (2012)