論文 / 著書情報
Article / Book Information

題目(和文)	大規模システムの並列分散離散イベントシミュレーションにおける性能最適化
Title(English)	Performance Optimization for Parallel and Distributed Discrete Event Simulation of Large-Scale Systems
著者(和文)	華井雅俊
Author(English)	Masatoshi Hanai
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第10104号, 授与年月日:2016年3月26日, 学位の種別:課程博士, 審査員:首藤 一幸,増原 英彦,渡辺 治,遠藤 敏夫,脇田 建
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第10104号, Conferred date:2016/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Performance Optimization for Parallel and Distributed Discrete Event Simulation of Large-Scale Systems

A DISSERTATION PRESENTED BY

MASATOSHI HANAI

SUPERVISED BY

KAZUYUKI SHUDO

FOR THE DEGREE OF

DOCTOR OF SCIENCE

DEPARTMENT OF MATHEMATICAL AND COMPUTING SCIENCES

GRADUATE SCHOOL OF INFORMATION SCIENCES AND ENGINEERING

TOKYO INSTITUTE OF TECHNOLOGY

TOKYO, JAPAN

MARCH 2016

Performance Optimization for Parallel and Distributed Discrete Event Simulation of Large-Scale Systems

Abstract

In the era when new large-scale systems have emerged because of the improvement in information technologies, it is necessary to clarify the behavior of large-scale systems, because society has become more dependent on these systems. Computational simulation is widely recognized as an important method to examine or study such large-scale systems because their behavior is too complex to analyze in static or mathematical ways. In addition, parallel and distributed discrete event simulation (PDES) techniques are used for fast and scalable simulation execution. This thesis discusses studies on the simulation of such large-scale systems with PDES, and I propose three new techniques.

First, I present a new modeling technique of emerging P2P systems. The proposal includes a data partitioning method for distributed computers and performance optimization for P2P system simulation on PDES, as well as a representation of P2P systems with the PDES scheme. The effective partitioning decreases the cost of machine-to-machine communication, which is one of the main performance factors in PDES. In addition, the proposal shows the effectiveness of some optimization techniques used to accelerate the simulation performance.

The second technique is a new redundancy reduction technique for a repeating simulation. Large-scale system analysis by simulation requires repeating the execution of simulation many times with various patterns of scenarios or parameters in order to validate or calibrate the results, which results in a large amount of redundancy, even when the change from a prior scenario to a later scenario is very minor. The new technique, *exact-differential simulation*, is illustrated. The technique enables to simulate only the changing scenarios in later executions while retaining exactly the same results as in the case of a complete simulation. The proposal also shows

the implementation of the exact-differential simulation on the top of optimistic PDES, and it illustrates the experimental results with large-scale traffic simulation.

The final technique provides efficient simulation execution optimized for a cloud computational environment. When using the cloud, where computing resources are rented in a ”pay as you go” charge model, cost performance is important. I proposed an adaptive virtual machine provisioning method used during the simulation, which enables the effective use of computational resources. The experiment, conducted with a case study of an agent-based traffic simulation on PDES, shows the improvement of its cost performance.

Contents

List of Figures

Acknowledgments

I would like to acknowledge the supports and advices by Prof. Kazuyuki Shudo, his group members, and dissertation committee members: Prof. Osamu Watanabe, Prof. Hidehiko Masuhara, Prof. Ken Wakita, and Prof. Toshio Endo. I also would like to thank Prof. Toyotaro Suzumura, Prof. Kalyan S. Perumalla, Prof. Georgios Theodoropoulos, and Dr. Anthony Ventresque for their deep discussion on my research.

It is definitely impossible to complete my Ph.D. course without support of my wife. We had a great time in Japan, UK and Ireland. Finally, I would like to thank my families for everything.

1

Introduction

Simulation has been one of the main computational applications for a long time ever since Stanislaw Ulam and John von Neumann used computational simulation in order to solve problems of neutron diffusion in 1946 [36]. It is used for various purposes such as solving scientific problems, testing physical systems, optimizing performance of computational systems, trainings drivers and pilot, and playing computer games. Especially in order to study or examine very complicated or large-scale systems using mathematical or static analysis methods, the computational simulation has a great advantage because it performs more detailed calculations of the systems' behavior than the mathematical or static methods. In this thesis, the focus is on such large-scale systems comprising over millions of entities, such as traffic systems and P2P systems on the Internet.

1.1 Parallel and Distributed Discrete Event Simulation of Large-Scale Systems

For large-scale system simulation, parallel and distributed discrete event simulation (PDES) is a beneficial way for fast and scalable execution. PDES is concerned with the problem of efficient using multiple processors simultaneously in order to execute single discrete event simulation, where the system's state is discrete and changes at discrete points of simulation time (on the other hand, in continuous simulation, the system's state changes continuously over time). The technologies and studies of PDES mainly contribute to large-scale system simulation in two ways as follows.

- **High performance**: With PDES, a lot of events can be processed very fast with multiple processors. In addition, a large number of entities in large-scale system simulation can be dealt with.
- **Accuracy**: With synchronization mechanisms of PDES, simulation events have to be processed in accurate time order even though computing environments (e.g., the number of cores, threads, processes, machines and other components) are changed.

1.1.1 Software for Parallel and Distributed Discrete Event Simulation

Figure 1.1.1 shows an overview of a typical PDES simulator, which usually has three layers: an application, simulation engine, and a computing environment.

Application

The application includes the core logic of a target system's behavior. A target system's actions are modeled by a sequence of discrete events and the system itself is modeled as a set of states which are spatially divided into some partitions, and which are changed by the events as a reaction of receiving events.

There are several applications adapted to PDES. For example, network systems have been traditionally one of the main applications. Pelkey and Riley proposed and implemented the distributed version of ns-3 [70], which is a well-known discrete event network simulator. Its performance evaluation and some optimization techniques have been demonstrated in recent

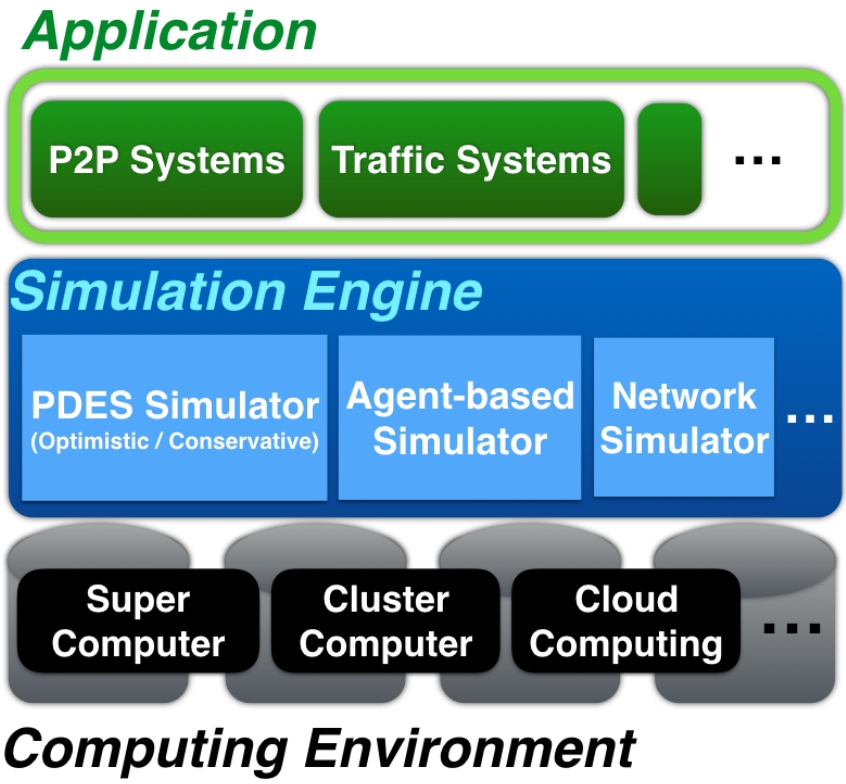

Figure 1.1.1: System Overview.

studies [43, 48, 62, 63, 80, 84, 90]. OMNeT++, which is a discrete event network simulator, also supports PDES [93]. Perumalla and Park illustrate the simulation result of MPI programs with over 0.22 billion virtual MPI processes on top of 216K physical processors (Cray XT5 supercomputer) [74], using $\mu\pi$ simulator [73], which is an MPI process simulator with PDES. Engelman also proposes *xSim*, which is a extreme-scale MPI program simulator, showing it can scale to 134M simulated MPI ranks on a 960-core Linux cluster [31, 32]. For hardware level simulation of supercomputers, Liu et al. proposed a new simulating technique to predict and analyze exa-scale computer systems by PDES. They show that their solution can simulate a computer system with torus network at a scale of millions to billions using up to 16,384 cores (IBM Blue Gene/L) [64]. They also showed a simulation result of supercomputers and data centers with a fat tree network [58]. In addition to these studies, many simulation techniques for supercomputers and data centers have been proposed during these decades. Pllana's good

survey [77] provides details for older work on simulation.

In addition, because of the improvement of mobile sensor devices, large-scale traffic systems have been simulated in recent research [15, 44, 51, 56, 60, 65–67, 69, 83, 88, 99]. Suzumura et al. examined Dublin city traffic with a Megaffic simulator and Dublin's real traffic data, which includes 92,465 junctions and 191,210 roads [87]. Meister et al. worked on simulating all the traffic in Switzerland using a MATSim simulator and Switzerland's real data, which includes 24,000 junctions and 60,000 roads [60]. In SUMO simulator community, Bieker et al. simulated a part of Bologna's traffic with real data, and Codeca et al. also simulated Luxembourg's traffic with a SUMO simulator and generated data, which includes 2376 junctions and 5969 roads [15, 28]. Zhao et al. analyzed the Buffalo-Niagara metropolitan area with TRANSIMS (TRansportation ANalysis SIMulation System), using 2185 junctions and 3037 roads [99].

Simulation Engine

A simulation engine (or simply a simulator) is middleware used to process events. It ensures that events are processed in the correct order even when the parallelism in computing environments is changed, such as the number of threads, the number of CPU cores, or the number of machines. This is called the *synchronization* mechanism. The synchronization mechanism is one of the main performance factors in PDES.

There are mainly two synchronization approaches: *optimistic* and *conservative*.

- **Optimistic synchronization**: In optimistic synchronization, events are processed ensuring only final events' ordering. *LP* (Logical Process: unit of parallel processing) processes events speculatively, and when causality destruction happens because of reception of events from other LPs, a LP rollbacks its local time and then reprocesses events again from that time.

- **Conservative synchronization**: In conservative synchronization, events are processed completely keeping the correct causal order. The conservative synchronization retains not only the final correct order but also maintain the order during event processing, differentiating it from the optimistic synchronization, which allows causality destruction during event processing and which only ensures final causal correctness.

There have been several studies and simulation engine middlewares , as shown in table 1.1.1.

Simulator	Modeling Type	Synchronization Method
ScaleSim	**PDES**	**Optimistic**
μsik	PDES	Optimistic / Conservative
ROSS	PDES	Optimistic
ROOTSIM	PDES	Optimistic
WARPED	PDES	Optimistic
XAXIS	Agent-based	Conservative
Repast HPC	Agent-based	Conservative
NS-3	Network	Conservative
OMNet++	Network	Conservative

Table 1.1.1: General Purpose PDES Engines.

ScaleSim [38] is a general optimistic PDES simulator using MPI and C++, which supports *exact-differential* simulation that is presented in Chapter 3. μsik is a general purpose parallel and distributed discrete event simulator, using MPI and C++, which supports both optimistic and conservative synchronization. The simulator is constructed by Perumalla [71], and it includes a simple standardized API switchable optimistic and conservative synchronization, base on Jha and Bagrodia's research [50]. In their most recent work, the state-of-the-art GVT mechanism was implemented and it performed great scalability to 216,000 cores of Cray XT5 systems [75, 76]. Rensselaer's Optimistic Simulation System (ROSS) is also a general purpose parallel and distributed discrete event simulator implemented using MPI and C, which supports optimistic PDES [23]. It is the fastest implementation of PDES in recent studies, as shown in [11]. Moreover, there are several optimistic PDES implementations such as Rome optimistic simulator (ROOT-Sim) [92] by the University of Roma and WARPED [59, 96], developed by the University of Cincinnati. XAXIS [85] is a general purpose parallel and distributed agent-based simulator by IBM Research. It is implemented in X10, which is a partitioned global address space (PGAS) model language for parallel computing. XAXIS uses a one of the time-window synchronization mechanisms, where barrier synchronization occurs for every time-window and agents are processed in parallel within each time-window. On the top of XAXIS, a large-scale agent-based traffic simulation, *Megaffic*, is implemented and it scales well, using 1,536 cores of a total of 128 machines on TSUBAME supercomputer [89]. Repast HPC [29] is also a parallel and distributed agent-based simulator implemented by Argonne National Laboratory. It is a high performance version of Repast Simphony, supporting ReLogo, which is

an agent-based simulation-specific language. Other than the listed simulators, network simulators, such as NS-3 [7] and OMNet++ [93], have similar architectures and mechanisms, where a middleware simulation engine controls a synchronization mechanism among distributed computers and has a simple application programming interface encapsulating such mechanisms.

Computing Environment

A simulation engine and application codes are executed on actual computing environments such as laptop computers, cluster computers and supercomputers. The typical computing environments and their features are listed as follows. As well as the computing power, such as CPU clock speed, there are various characteristics to be considered in each environment.

- **Supercomputer** A supercomputer is the main computing environment used for large-scale systems simulation. There are some differences between today's supercomputers and commodity clusters. First, today's supercomputers have much faster interconnections than commodity clusters, such as InfiniBand. Second, they have CPU/GPU accelerator such as Intel Xeon Phi and NVIDIA Tesla. For example, both the TSUBAME supercomputer [10] at the Tokyo Institute of Technology and the Titan supercomputer [55] at the Oak Ridge National Laboratory have GPU accelerators and the machines are connected with InfiniBand. Moreover, the next-generation supercomputers, such as Summit [54] at the Oak Ridge National Laboratory, have a bigger memory capacity, including volatile memory for big data processing, which has a big impact to large-scale systems simulation because memory size is the main factor limiting scalability in large-scale systems simulation, especially with optimistic PDES.

- **Cloud computing platform** A cloud computing platform is a relatively new computing environment for distributed systems. There are some differences from supercomputers. First, a cloud computing environment adopts a pay-as-you-go cost model for users to access its machines. Therefore, cost performance is an important factor when using this environment. Second, the machines are basically hosted as virtual machines. Thus, we can easily and flexibly control the number of machines being used, though the environment has lower computing power than supercomputers. For example, in web-server applications, an auto-scaling technique is well-known, where the system adds virtual machines when CPU utilization is full because of the bursting of page viewing.

- **Laptop computer** A commodity laptop computer is still used for large-scale systems simulation, especially for large-scale traffic simulation such as MATSim and SUMO [1, 14]. However, stand-alone execution limits its performance and in order to solve performance problems, it sometimes needs to remodel simpler application code, which destroys the accuracy of the results. For fine and accurate simulation, a distributed computing method is required.

For more details on PDES, Fujimoto's book, *Parallel and Distributed Simulation Systems*, provides a excellent guidance [34].

1.2 Performance Optimization in Parallel and Distributed Discrete Event Simulation of Large-Scale Systems

This section provides problem statements and my proposals from the viewpoint of the three layers of PDES simulators.

In the application layer, where a target large-scale system prototype is constructed (simplified enough to calculate using computing methods), there exists a trade-off between modeling complexity and performance. Therefore, how to simplify the original system and which abstraction level the simulation model adopts are important concerns for ensuring performance. In addition, the method used to partition processors or machines highly impacts performance.

In the simulation engine, which directly impacts performance, efficient repetition of the simulation is one of the challenging points, along with parallelism, the caching mechanism, and the choice of synchronization methods (optimistic or conservative). Simulation analysis of large-scale systems needs to repeat execution many times with various patterns of scenarios or parameters. The number of repetitions increases according to the number of entities a target system has. For example, traffic accidents in Tokyo, which has 770K junctions and 2M roads, needs 770K scenario patterns (equal to the number of junctions) to determine what happens if an accident occurs at each junction. It needs to repeat the simulation execution 770K times. For more complex situations, such as two accidents happening simultaneously, the number of possible scenarios increases exponentially.

To use computing environments efficiently, a specific optimization of each computing environment is needed, because different computing environments have different performance characteristics, such as the number of cores, the volume of memory, network bandwidth, and

other factors. Not only the elapsed time of the simulation execution but also other performance measures, such as energy consumption and payment cost, become important in today's computational environments.

In the rest of this section, a brief overview of the proposals is presented from the viewpoint of each layer in the PDES simulators.

1.2.1 Application: High Performance P2P System Simulation

P2P systems with over a million nodes have emerged on the Internet. For example, BitTorrent, which is a P2P file-delivering system, has grown to 150 million nodes [16], using 9 million nodes running simultaneously to construct a single overlay network [91].

However, existing simulators and techniques cannot simulate at such a large scale. Even parallelized simulators have not solved the scale problem; they provide large amounts of memory and hold a large number of nodes, but the speed of simulation degrades more significantly than sequential simulation because of much overhead needed for synchronization among simulation processes. Actually, using dPeerSim [30] with 16 LPs, simulating Chord protocol in a scenario with 320,000 nodes issuing 320,000 requests takes 1 h and 6 min, which is 86 times of sequential simulation.

In this proposal, a new method to model P2P simulation for fast execution is presented. This modeling method basically uses optimistic PDES, differentiating the modeling method from other P2P simulation techniques which use conservative PDES for synchronization. This technique also includes some optimization for low-cost synchronization based on the workload characterization of P2P system simulation. The experience using this method shows good strong scalability, which is accomplished using a combination of the optimistic synchronization and its optimization techniques.

1.2.2 Simulation Engine: Exact-Differential Simulation of Large-Scale Systems

To deal with the repetition problem, a new redundancy reduction technique, called *exact-differential simulation*, is presented, which enables the simulator to simulate only changing scenarios during later execution while keeping exactly the same results as in the case of the complete simulation. The proposal provides two main contributions: (1) a key idea and algorithm for the exact-differential simulation, and (2) a method to build large-scale traffic simulation on the top of the exact-differential simulation. In experiments simulating a single traffic accident in Tokyo, the

exact-differential simulation shows 7.26 times as much elapsed time improvement on average and 2.26 times improvement over the complete simulation, even in the worst case where the single traffic accident mostly spreads over Tokyo.

This thesis also provides an extended technique of exact-differential simulation for more efficient simulation analysis, including two ideas, (1) *what-if scenario filtering*, which is method for ignoring meaningless scenarios to decrease repeating simulation, and (2) *exact-differential cloning*, which is a method for executing tasks in parallel with exact-differential simulation.

1.2.3 Computing Environment: Cost Optimization For the Cloud

In order to execute PDES of large-scale systems, a typical approach is to use multiple computers such as supercomputers, cluster computers, or cloud computing. Different computing platforms have different attributes and simulators need to optimize for each platform.

The focus in this proposal is on efficient usage of cloud computing environment, which can provide and release VMs easily and is rented using a pay-as-you-go cost model.

The proposal provides two main contributions.

- **Framework for VM provisioning**: A method for efficient utilization of VM resources for PDES, providing a mechanism that adapts the resource provisioning to users' objectives and workload evolution
- **Staged asynchronous migration technique**: A technique for limiting the migration overhead when the number of workers change by overlapping the migration process with the simulation workload

The experimental results using a 24 h scenario of traffic in the city of Tokyo show that our system outperforms a static provisioning by 12% on average and 23% during periods when workload changes a lot.

1.3 Outline of this Thesis

In Chapter 2, the new modeling of P2P systems on optimistic PDES for high performance execution is illustrated. Chapter 3 and Chapter 4 provide a new technique of PDES engines, discussing the optimization of execution repetition. In Chapter 5, the cost optimization of large-scale traffic simulation on a cloud computing platform is discussed as a new technique for the computing environment layer. Finally, Chapter 6 provides the conclusions of this thesis.

2

High Performance P2P System Simulation

Research on large-scale distributed systems such as P2P systems requires support of simulation. It is difficult and often impossible to provide realistic experimental systems because they consist of a large number and various kinds of computers and devices on practical networks. Static analysis is also difficult. A lot of simulators have been developed to meet various kinds of requirements that include physical layer analysis [6, 7, 93], application layer analysis [13, 24, 35, 61, 82], and researching security problems.

In terms of system scale, there is a big gap between the simulators and the largest systems running on Internet. Existing simulators cannot simulate such a large scale of those running systems. Table 2.0.1 shows how many nodes the simulators can simulate and the scale of one of the largest systems, BitTorrent DHT. BitTorrent, which is a P2P file delivering system on Internet, has grown up to 150 millions scale up to January 2012 [16]. As of May 2014, 9 millions nodes run simultaneously and compose an overlay network [91]. Existing simulators cannot simulate the actual BitTorrent scale. This is a serious problem because we do not have a means to evaluate the following issues.

- How much a load on Internet does such an existing large-scale system have?

System	Maximum number of nodes
BitTorrent DHT (as of May 2014)	*9,000,000*[91]
SimGrid	2,000,000
OverSim	100,000
PeerSim	100,000
ns-2, ns-3	10,000

Table 2.0.1: Maximum Number of Simulated Nodes.

- Does such a system interfere stability of Internet, e.g. by unexpected heavy traffic?

Today's spreading smart phones and other mobile devices make the scale of Internet larger and larger. Moreover, not only such mobile devices but also vehicles and consumer electronics have Internet connections. This should bring about numbers of distributed systems with over hundreds of millions of devices.

On the other hand, to overcome the limit on the scale of single computer simulation, numbers of techniques have been proposed in the field of parallel discrete event simulation (PDES) [12][23][29]. But such simulation techniques have been hardly adopted to P2P simulation and there are few work on a combination of PDES and P2P simulation [30][78], which mentioned that adapting PDES to P2P simulation is challenging and hard to accomplish better performance than single core or single machine simulation because P2P simulation workload does not match traditional PDES algorithms. But it appears that such difficulty is not because of essential problem of P2P simulation workload but because of selection of a PDES algorithm including optimization and modeling of P2P systems, which is not for the scale or performance but for the other demand when the simulator was designed.

In this chapter, a new simulation technique for large-scale P2P systems is presented. The technique is based on optimistic parallel discrete event simulation (PDES) and employs low cost synchronization techniques, which are effective for P2P simulation. Thus the contributions are the followings.

- Demonstration of optimistic PDES of P2P systems and its performance
- Optimization techniques for optimistic PDES, which are effective for P2P simulation

Simulator	Abstraction level of modeling	Modeling of time	Maximum scale	Parallelization
SimGrid	Overlay	Discrete event	2,000,000	Multi threaded
Overlay Weaver	Overlay	Real time and discrete event	1,000,000	Multi threaded for real time and distributed
dPeerSim, PeerSim	Overlay	Discrete event and query cycle	320,000	Distributed
OverSim	Underlay with OMNet++	Discrete event	100,000	Single thread
PlanetSim	Overlay	Discrete event and query cycle	100,000	Multi threaded

Table 2.1.1: Comparison of P2P Simulators.

2.1 Previous P2P Simulators

Numbers of P2P simulators and simulation techniques have been proposed. They were designed with a variety of modeling to match their own purposes. These simulators are classified according to abstraction level of modeling, scalability and a way to parallelize. Table 2.1.1 shows the comparison of such P2P simulators.

dPeerSim [30] is an extension of PeerSim [61] that allows distributed simulation of very large scenarios using a conservative PDES technique. However, the overhead of distributing the simulation is very high. With 16 logical processes, simulating Chord protocol in a scenario with 320,000 nodes issuing 320,000 requests takes 1 hour and 6 minutes, which is 86 times of sequential simulation. The approach of this research is similar to our research, but the difference from my research is that they use conservative algorithm, which is a natural extension of an existing sequential simulator but not suited for scaling P2P simulation, where network topology changes dynamically and have little lookahead.

SimGrid [24, 78] is a general open-source simulation framework including a parallel P2P simulator based on a PDES technique. In this research, instead of using traditional PDES techniques, they proposed an original method to parallelize P2P simulation inspired by the concept of operating system. The system is divided into user space and OS kernel. The former layer is sequential simulation engine and the latter layer manages parallelism of the system. P2P simulation is well parallelized and shows better performance than any other existing simulator. The

difference between this research and mine is that we achieve not only parallelization of simulation but also execution on distributed environment, which is necessary to overcome the possible scale of a single machine.

Overlay Weaver [82] is an overlay construction toolkit, which supports development P2P algorithms. It provides multiple messaging layers for simulation and a real network and they enable simulated P2P algorithms to be simulated and also run on a real network. It can simulate about one hundred millions nodes on a single computer with enough memory not slower than real time.

OverSim [13] is one of widely used P2P simulators. It was reported to simulate up to 100,000 nodes in an event driven approach. When the network needs to be precisely simulated, its scalable network models can be replaced with OMNet++ packet-level simulator [93].

PlanetSim [35] permits parallel simulation. Its query-cycle approach makes its parallelization rather straightforward such that at each simulation step, each process has to be processed separately. The authors report a speedup by 1.3 times on two processors. The simulator achieved a good speedup by adopting query cycle approach, in which there are some limitations in modeling of P2P systems behavior. The approach requires excessively simple modeling and different from discrete event simulation.

2.2 Optimistic Parallel Discrete Event Simulation of P2P Systems

In this section, I describe a way to model P2P systems by optimistic PDES. As I discussed above, my goal is simulation of very large-scale P2P systems, which does not require excessive details of underlay networks such as packet queueing on a network switch. To simulate such very large-scale P2P systems, there are three point to be considered carefully.

- **Accuracy at application layer**

  The simulation must be accurate at application layer. But it does not have to be excessively detailed in network layers. Over detailed models waste processing power and memory and do not scale.

- **Memory efficiency**

  To simulate as many nodes as possible, it is necessary to reduce a memory footprint as long as it can simulate the P2P systems accurately. Simpler modeling yields a less memory footprint.

```
1 interface Node {
2   /**
3   * Processes a received message
4   * and generates a message to be sent.
5   * @param Received message
6   * @return Message to be sent
7   */
8   Event eventHandler(Event receivedMessage);
9 }
```

Figure 2.2.1: Interface of Node Behavior Description.

- **Partition-ability**

  In PDES, simulated objects are arranged to distributed memory space. Thus it is important to partition the simulated objects in a way to reduce as much communications between machines as possible to run efficiently on multiple computers.

2.2.1 Modeling of P2P Systems

My model of a P2P system is the same as PeerSim and Overlay Weaver, in which a system consists of a set of nodes on an overlay network. An underlay network including physical layer, layer 2 and layer 3 is abstracted and the nodes communicate each other directly by specifying an IP address of the receiver.

Figure 2.2.1 shows my interface of node behavior description. Node behavior is described in the *eventHandler*() method, which takes a received message as a parameter and returns messages to be sent. Figure 2.2.2 shows an example of an event in which Node A sends a message to Node B. An event has a source address, a destination address, processing time and an event value. The events are stored in event queue by the processing time order. First, an event with the earliest processing time is dequeued from the event queue. A node with the destination address specified by the dequeued event processes the event by invoking the *eventHandler*() method. The node generates new events and they are inserted to the event queue. These steps are iterated as long as the event queue has an event or until specified finishing time. Figure 2.2.3

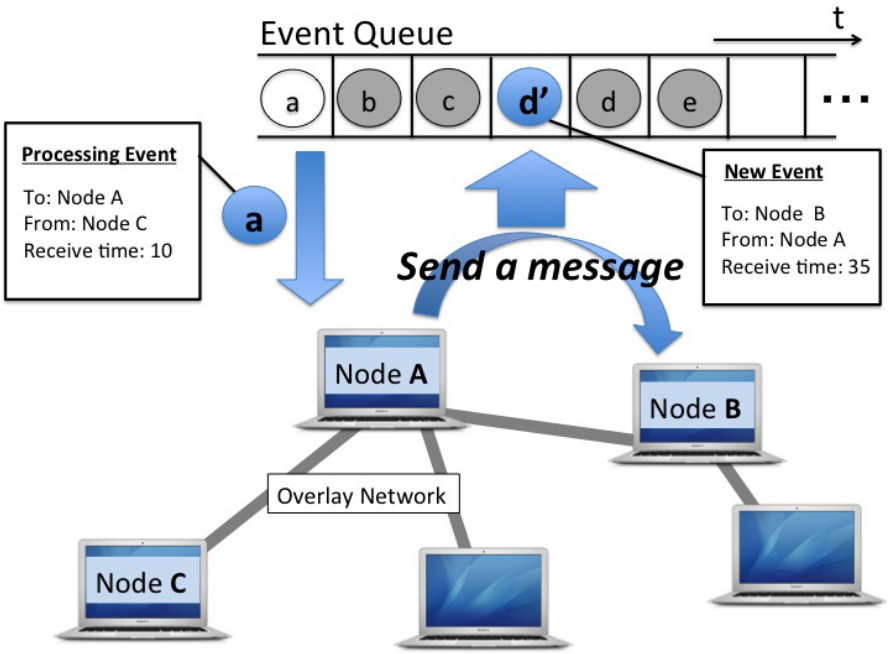

Figure 2.2.2: Communication in Discrete Event Simulation.

shows a pseudo code of event processing.

2.2.2 Optimistic Parallel P2P Simulation

As discussed above, I use an optimistic PDES technique to simulate P2P systems on multiple computers. The set of all nodes in a P2P system is divided into subsets and the subsets are allocated to each logical process. Logical processes perform optimistic PDES. Each logical process has its local event queue and the queue stores events to be processed. When a logical process receives an event with processing time less than its local time, a rollback occurs and the process sends anti-messages to other logical processes.

Figure 2.2.4 shows an example where Node B in logical process 1 sends a message to Node C in logical process 2. Node B processes message *a* by invoking *eventHandler*() and generate message *b* to be sent to Node C which belongs to logical process 2. Thus an inter-machine communication occurs between memory space A and memory space B. If the processing time of the message *b* is less than the local time in logical process 2, a rollback occurs.

```
1  // Provides an event queue
2  EventQueue eventQueue = new EventQueue();
3
4  // Processes events
5  while (eventQueue.size() > 0 &&
6     eventQueue.time() < TIME_TO_FINISH) {
7     Event receivedMessage
8        = eventQueue.getEariestEvent();
9     Node node
10       = getNode(receivedMessage.getDistination());
11    Event generatedMessage
12       = node.messageHandler(receivedMessage);
13    eventQueue.insertEvent(generatedMessage);
14 }
```

Figure 2.2.3: Pseudo Code of Event Processing.

2.2.3 Block-based Partition by an Application-Level ID

How a P2P system is partitioned has much influence on performance of simulation. Partitioning has much influence on the amount of communications between machines, that dominates the performance.

Here, I estimate how much communication gets across partitions. Let us suppose that whole ID space is X and Node A belongs to partition A. And I define $p(x)$ as a probability that Node A communicates with Node X with identifier x. Hence I describe the probability in non cross over partitions p that Node A communicates with a node in a same partition as follow,

$$p = \sum_{x \in A} p(x)$$

Clearly, $1 - p$ is the probability that Node A communicates with a node in the different partition containing Node A.

In P2P algorithms there are two kinds of identifiers, IP address and application-level IDs. A node in a DHT has its IP address and node ID usually generated by a hash function. My system adapts an application-level ID for block-based partitioning. This is because mostly in P2P overlay networks *a network locality* appears in an application-level ID. Here, networks have *locality* if

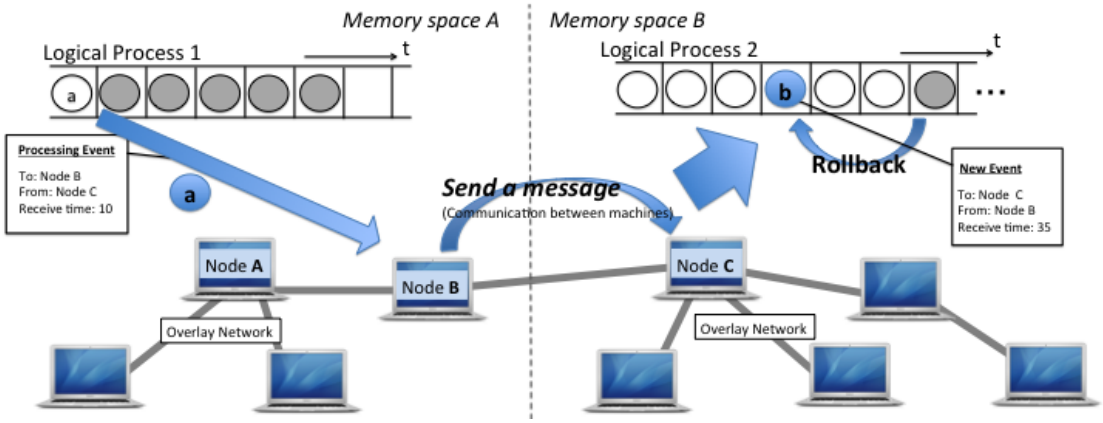

Figure 2.2.4: Communication in Parallel Discrete Event Simulation.

there is a relation between distance of IDs and the probability that nodes with such IDs are connected. Actually, in most P2P algorithms nodes with low distance is much communicated each other. Now, I assume that a distance function is $dis(x, y)$ and the number of total element of X is $|X|$. Also, I assume border point of the partitions as a_i $(\in P_i)$, where $i \in (0, 1, 2, \ldots n-1)$ and $dis(a_i, a_{i+1}) = |X|/n$. I can describe a block-based partition P_i $(i = 0, 1, 2, \ldots n-1)$ as follow,

$$P_i = \{x \in X |\ dis(x, a_i) \leq |X|/n\}$$

Since most P2P algorithms have network locality on application-level ID, by using decreasing function $f : \mathbb{R} \to [0, 1]$, I can describe $p(x)$ as,

$$p(x) = f(dis(a, x))$$

Thus, non cross over probability p is,

$$p = \sum_{dis(x,a_i) \leq |X|/n} f(dis(a, x))$$

For example of Chord with m-bits identifier, which is one of the distributed hash table algo-

rithms. If an application-level ID is used for block-based partitioning,

$$dis(a,x) = \begin{cases} x - a, & (a < x) \\ x - a + 2^m, & (x \leq a) \end{cases}$$

$$f(dis(a,x)) = 2^{-dis(a,x)} \cdot Const$$

Therefore, $p(x)$ can be described as follow,

$$p(x) = \begin{cases} 2^{a-x} \cdot Const, & (a < x) \\ 2^{a-x-2^m} \cdot Const & (x \leq a) \end{cases}$$

Thus, non cross over probability p is in the case $a < x$,

$$p = \sum_{dis(x,a_i) \leq |X|/n} 2^{a-x} \cdot Const, \quad (a < x)$$

In the case $x \leq a$,

$$p = \sum_{dis(x,a_i) \leq |X|/n} 2^{a-x-2^m} \cdot Const, \quad (x \leq a)$$

2.3 Low Cost Synchronization Effective for P2P Simulation

2.3.1 Incremental State Saving

Firstly I discuss the data structure of logical process event queue. In my implementation, I use incremental state saving [25][81][95]. Mainly, there are two kinds of implementations for event queue, copy state saving and incremental state saving. In the copy state saving, which is naive implementation of state saving, logical process makes a copy of all of the state variables. On the other hand, in incremental state saving, a log recording changes to individual state variables is stored for each event processing. The storing only changes brings about reducing memory space for saving old event. Although reducing the saving cost in the incremental state saving, rollback cost is increasing compared with the copy state saving because it needs to scan the all events from starting time to rollbacking time and then reproduce the state from the

scanning logs. In the copy state saving, all it has to do is to rollback directly to the rollbacking time and just change into the stored state.

In my P2P simulation, for example, suppose N is the average number of nodes in each logical process, n is memory cost of one node and q is the length of the state saving queue. The memory cost of incremental state saving in each logical process $MemCost_{inc}$ is represented as follow:

$$MemCost_{inc} \propto nq$$

Also the memory cost of copy state saving in each logical process $MemCost_{copy}$ is represented as follow:

$$MemCost_{copy} \propto nNq$$

Summarizing the above, in my P2P simulation, by using the incremental state saving the memory usage cost can be reduced $1/N$ in average of the cost for copy state saving.

However, such incremental state saving brings about increasing the cost for rollback because when the rollback occurs, a logical process reproduces a logical state by gathering the saved logs. Thus, in the workload that many rollbacks occur, the incremental state saving brings about serious performance degradation. In next part, I discuss about technique about reducing rollbacks by reducing anti-messages, which spread over other logical processes and cause another rollbacks.

2.3.2 Lazy Cancellation

To reduce anti-messages, I use lazy cancellation technique [57], which avoids cancelling messages, which are later recreated when the events are reprocessed. In aggressive cancellation, which is naive way of sending anti-messages, anti-messages are sent as soon as rollback occurs. This cancellation brings about excessive generating anti-messages because there are some canceled messages, which are independent from the received anti-message and process again after the rollback. On the other hand in lazy cancellation, such independent messages is ignored to cancel. When a rollback occurs from an anti-message or a straggler message, which is a message with a timestamp less than the logical process, no anti-messages are sent. Instead, the logical process only sends an anti-message if the original message was not again created when the events were reprocessed. Such situation occurs when an event does not affect to another later event.

Thus in P2P simulation, this situation occurs in the case of a message which does not change the node status, for example in the case of forwarding message, ping/pong message, get message and so on. In fact, the most of all messages is forwarding message in P2P simulation and this technique have a high effect for P2P simulation.

Suppose the frequency of non status changeable message such like forwarding message is f ($0 < f < 1$). Then, the f means anti-message reduction rate compared with the aggressive cancelling because f of the all messages is ignored in the lazy cancellation. For example in iterative static Chord, suppose frequency of a node status changeable query, for example a put query, is x, Q is a set of all queries and a query $q \in Q$ hops $h(q)$ times to the destination. Then the number of sending and receiving messages to simulate the query is

$$N_m(q) = 2h(q) + 1$$

Node status changes only at the destination by the node status changeable query. Thus the frequency of non-status changeable message f is

$$\begin{aligned} f &= x \cdot \frac{|Q|}{\sum_{q \in Q} N_m(q)} \\ &= x \cdot \frac{|Q|}{\sum_{q \in Q} \{2h(q) + 1\}} \\ &= x \cdot \frac{1}{2H + 1}, \end{aligned}$$

where H is the average of query hops. In Chord, the number of hops can be represented by $O(logN)$, where N is the number of nodes. Since f means anti-message reduction rate in lazy cancellation, in Chord simulation, I can reduce $x \cdot \frac{1}{2H+1}$ of the all anti-messages.

2.4 Performance Evaluation of Optimistic P2P Simulation

In this section, I describe the performance of the P2P simulation by simulating Chord with 10000 nodes. There are two topics to evaluate in my experiments, efficiency of the low cost synchronization and scalability of the simulation. I implement the P2P simulator with C++ and MPI, which is a standard message passing API for parallel computing. In this experiment, I use 6 commodity servers to execute the P2P simulator. Table 2.4.1 shows the configuration of

OS	Linux 3.8.0
CPU	2.40 GHz Xeon E5620 × 2
Memory	32GB RAM
MPI	Open MPI 1.6.5 (MPI 2.1 standard)

Table 2.4.1: Server Configuration.

the servers. Table 2.4.2 shows the configuration of the simulation.

Algorithm	Chord Protocol
# of Nodes	10000
Partitions	1 Partition/Core
Basic # of Queries	100,000
Basic Lookahead	100

Table 2.4.2: Simulation Configuration.

2.4.1 Efficiency of the Low Cost Synchronization

In this experiment, I evaluate the efficiency of my proposed low cost synchronization. I simulate totally 10000 chord nodes. The simulation scenario is that initially, 30 nodes make an overlay network and then the other 9970 nodes join this overlay network and send stabilize query in 10 times.

Figure 2.4.1 shows transition of state saving costs in one logical process at each global synchronization. In incremental state saving, the number of saving node state is kept to approximately 570 in average and 1181 at most but by using copy sate saving, which is a calculation result, the logical process needs to save 6200 nodes state in average and 11003 nodes state at most. The incremental state saving only use 9.1 % memory costs of the copy state saving in average and use 10 % at most.

Figure 2.4.2 shows the number of total anti-messages during the simulation, which causes the performance degradation in an optimistic parallel and distributed discrete event simulation. I change the number of machines, where there are 8 MPI processes per machine, and execute the same simulation. In the lazy cancellation, there are only 7.5 % anti-messages in average and at

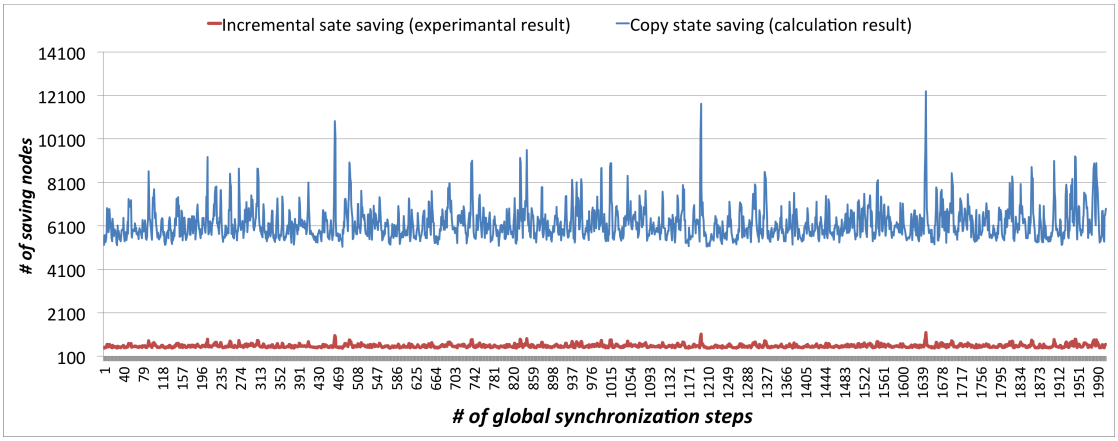

Figure 2.4.1: Temporal Transition of State Saving Cost.

Parameters of PHOLD benchmark	The P2P workload parameters
The number of logical process	The number of logical processes
Message population	The number of queries
Time stamp increment function	Network delay
Movement function	–
Computation grain	–
Initial configuration	Initial events
Lookahead	Minimal network delay and query sending interval

Table 2.4.3: Comparison between PHOLD and my P2P Workload Parameters.

least 2.5 % anti-messages compared to the case in the aggressive cancellation. Also, even though the number of machines is increasing, the lazy cancellation keep the number of anti-messages low.

These experimental results mean that actually the incremental state saving and the lazy cancellation have big impacts to the P2P simulation.

2.4.2 Scalability of Optimistic P2P Simulation

In this experiment, I show the performance of the P2P simulation with various kinds of scenarios and configurations.

Before illustrating the evaluations, I discuss a workloads parameters of the experiment to

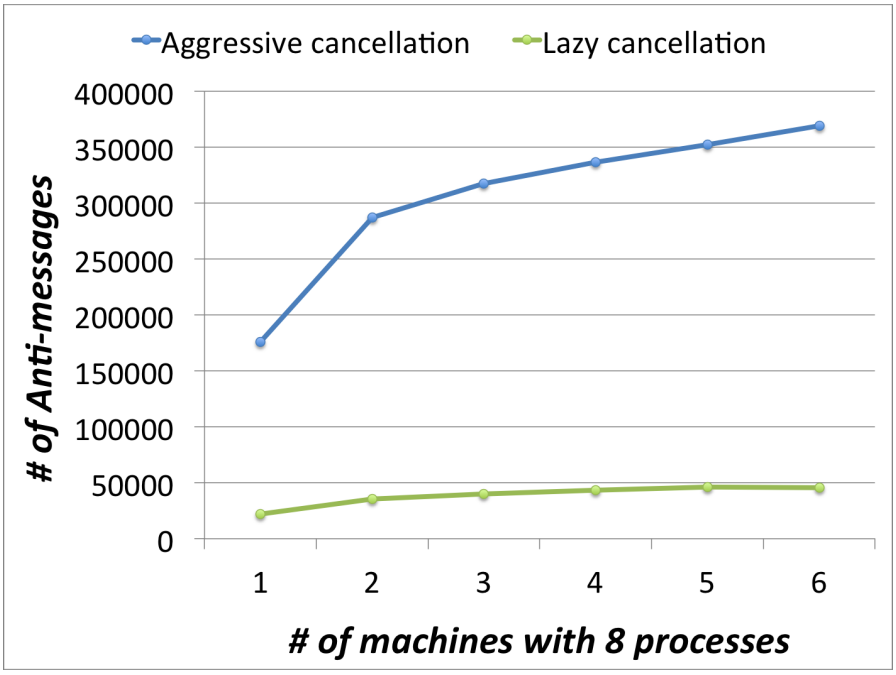

Figure 2.4.2: Number of Total Anti-Messages during Simulation.

evaluate a performance of my simulator. To evaluate performance, I prepare experimental scenarios and conditions based on PHOLD benchmark, which has been used to evaluate optimistic PDES algorithms in the field. Table 2.4.3 shows the interpretation of PHOLD in the view point of P2P simulation. In the P2P simulation, I can change 4 parameters of the PHOLD parameters – the number of logical process, message population, time stamp increment function, initial configuration and lookahead. The number of logical processes is changeable in P2P simulation such as general optimistic parallel discrete event simulator. The message population, which indicates the number of causality threads in parallel simulation, is decided by the number of queries during a simulation in the P2P simulation. The time stamp increment function, which defines the number of an incrementing time stamp of an event in each event processing, is decided by Network delay in my simulation. The initial configuration, which defines event starting time and a process where the event is started, is decided by initial events in the P2P simulation. Also lookahead is decided by a minimal network delay and query sending interval, which is the time interval to the next sending message. On the other hand, movement function and computation grain cannot be changed in the P2P simulation. Movement function, which defines which logical process an event remove to in next time, depends on a routing algorithm

of P2P system. Computation grain, which defines the amount of computation required to process a single, also depends on an algorithm of P2P system.

In this experiment, I evaluate 10000 nodes of Chord such as previous experiments. The simulation scenario is that initially, 30 nodes make an overlay network and after that the other 9970 nodes join this overlay network and send stabilize queries basically in 10 times. The network latency is calculated randomly based on uniform distribution between 100 and 1000. According to the workload parameters, I change the number of queries, the network delay, the minimal network delay and the joining interval of the nodes.

Figure 2.4.3 shows the experimental result of the simulation execution time with different lookaheads, which is represented by the minimal network delay and the query sending interval of nodes. The number 10, 50, 100 are simulation times. This result means that the lookahead have an impact to execution time. In the case that lookahead is 10 and 50, the simulation performance is getting better according to the number of machine and after 4 machines, the performance saturates. When the lookahead is 100, the execution time is totally fast but the performance saturates. In my simulation, although lookahead is small, a performance can improve by parallel execution.

Figure 2.4.4 illustrates the result of performance with various numbers of queries. As well as lookahead, the number of query impacts the simulation performance but the performance can improve by parallel execution if the there are enough queries to parallelize.

Figure 2.4.5 shows the execution time rate of each latency distribution. I use a Poisson distribution with the parameter $\lambda = 0.1, 0.5, 1.0$. The Poisson distribution with small parameter is one of the long tail distributions such as Internet latency. This result shows that the latency distribution also impacts the P2P simulation performance. When λ become smaller, the number of network links with small latency increase and this causes the performance degradation. This experimental result is little limited because the execution time of this workload is very high and cannot parallelize well in the case of Poisson distribution because in my implementation the network latency is calculated during the simulation and generating Poisson distribution needs high cost. To compare correctly with the case of uniform distribution, I add the overhead time of generating Poisson distribution to uniform distribution workload and then compare the rate of each execution time.

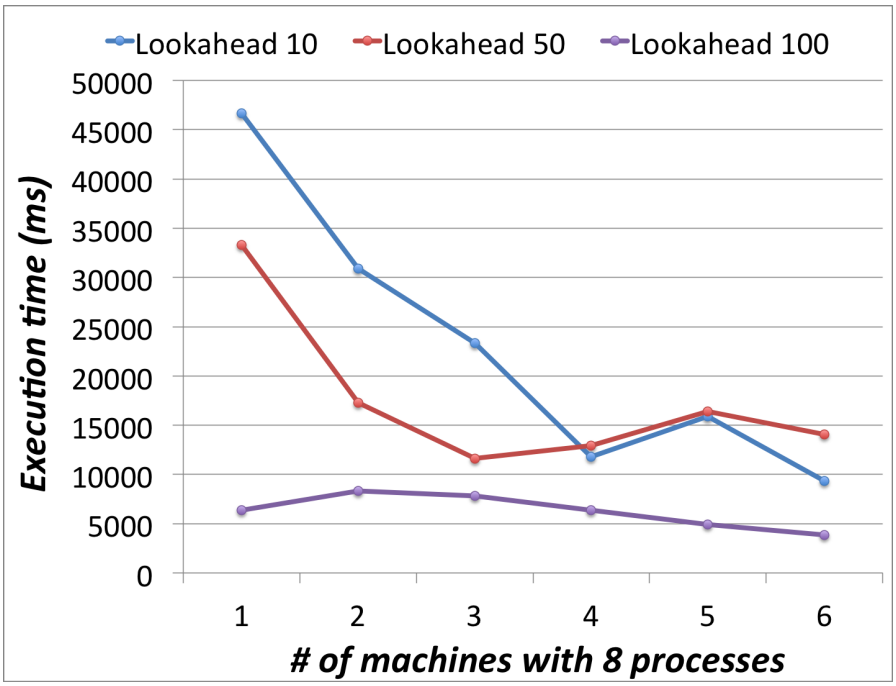

Figure 2.4.3: Execution Time in Different Lookahead.

2.5 Summary

In this chapter, I presented the P2P simulation technique based on optimistic parallel discrete event simulation. The main problem of the optimistic simulation technique is high cost for synchronization. I showed a low cost synchronization very suited to the optimistic P2P simulation. In the experiments, I illustrated an effect of the low cost synchronization techniques, incremental state saving and lazy cancellation. In the P2P simulation, the incremental state saving reduces at most 91 % of state saving cost compared to copy sate saving. Also the lazy cancellation reduces at most 97.5 % of the number of anti-messages compared to aggressive cancellation. Moreover, the scalability of the simulation is presented, which uses P2P systems scenarios based on the optimistic simulation benchmark.

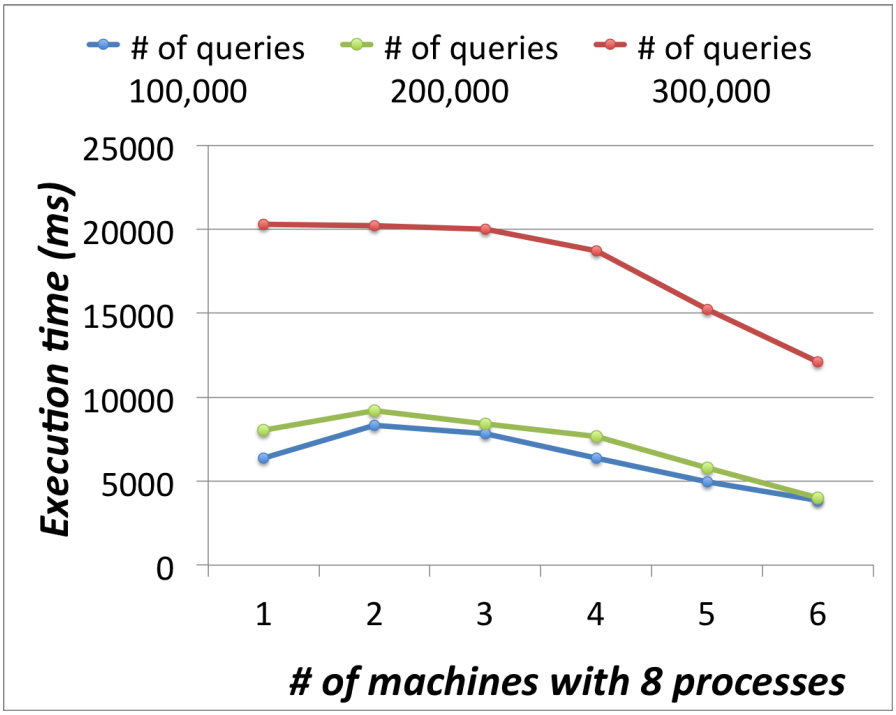

Figure 2.4.4: Execution Time in Different Number of Queries.

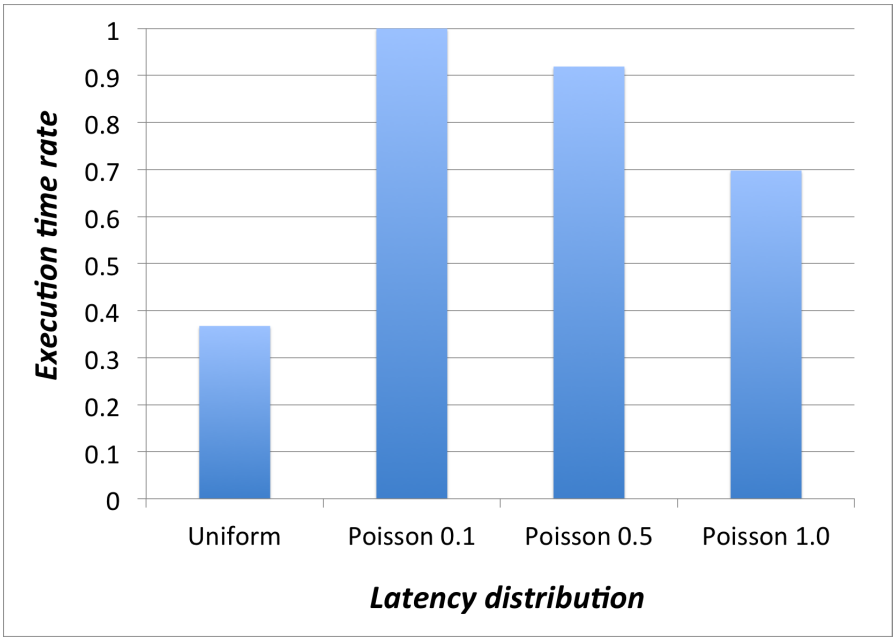

Figure 2.4.5: Execution Time Rate in Different Latency Types.

3

Exact-Differential Simulation

Large-scale microscopic traffic simulation has been a beneficial way to research on areas such as prediction of traffic congestion, planning of urban developments, citizen's behavior in emergencies. Unlike other statistical and mathematical ways, such approaches can give detail analysis results of the individual vehicles and other entities like junctions and roads because it actually emulates the vehicles' behavior and interactions with each other.

To analyse by such simulation, it needs to repeat execution many times with different scenarios and parameters. In Tokyo traffic simulation [40, 65, 66, 86], for example, we need to execute 770K times simulation when we simulate what happens if one of the roads is blocked because there are 770K junctions in Tokyo. When we simulate multiple blocks of the roads, we need to execute 2^{770K} times (the sum of combination from 770K choosing 0 to 770K). Also, it often needs to execute a lot of times for parameter tuning (e.g. road speed limit, a time interval of signals) to imitate a realistic situation.

However, previous simulating methods and simulators have a big overhead and a lot of redundancy when repeating the simulation, especially when very small part of scenarios are changed. For example, if only one of all roads is changed in later repeating execution, almost all of simu-

lating results are same as prior execution and the change affects only small part but we have to simulate whole scenarios from the beginning. The reason why the previous simulators have to simulate whole scenarios from the beginning is that a naive spatial partial way (separating simulation space in advance and only simulating the separated part) brings about inconsistency of simulating results from the whole simulating results because it cannot simulate an influence from outside of the separated part and such influence cannot be fixed in advance.

In this chapter, I propose a novel technique to simulate only a part of all scenarios and states keeping exact same simulating results as whole execution, called *exact-differential simulation*. The "exact" implies that the output result will be identical. The "differential" implies that only affected events will be reprocessed.

The main idea of the exact-differential simulation is that, in initial whole execution, the simulator stores all events and intermediate states before reprocessing only changing events in later repeating executions by using the stored events and states. There are 3 main contributions in this chapter: (i) I describe a way to store processed events and intermediate states and a way to reuse them. (ii) I illustrate implementation of the simulator, which meets requirements of the exact-differential simulation. (iii) I evaluate an efficiency and performance of the exact-differential simulation by Tokyo traffic simulation.

3.1 Exact-Differential Simulation: Concept and Algorithm

In this section, I describe the processing flow of the exact-differential simulation and its static performance analysis.

3.1.1 Processing Flow

The simulation flow basically consists of two parts: initial whole execution and repeating execution. In the initial whole execution, the simulator stores all processed events and intermediate states. After the initial whole execution, the simulator starts the repeating execution from a changing point and reprocesses only affected events using the stored events and states.

Initial Whole Execution

In the initial whole execution, events are processed in the almost same way as the optimistic PDES [49], where unlike the normal optimistic PDES, events, cancel events and stored states

are never released by the global synchronization (or *GVT, global virtual time*). Instead, such events, cancel events and states are stored to storage for reusing in later repeating execution. Thus, in full, each LP processes events in parallel with a time-sorted event queue and exchanges new generated events with other LPs. When a LP receives the new event with earlier time stamp than its own local time, the LP rollbacks its local time and sends cancel events to neighbors. In the other case that the new received event is older than its local time, the new event is just inserted to its event queue. Cancel events and states are stored whenever a new event is generated. In the global synchronization, older events, cancel events and states than the global time are stored to storage instead of releasing as usual.

Exact-Differential Simulation in Repeating Execution

In the repeating execution, the simulator, at first, inputs a what-if query, which defines changing time and place (LP) and a query's type: ADD or DELETE. Algorithm 1 shows the what-if query processing. In the case of ADD (line 3 – 7), a new event generated from the ADD query is inserted to the event queue and the local time is rollbacked to the new event's time before cancel events are sent to all affected LPs. In the case of DELETE (line 8 – 13), an old event generated from the DELETE query is removed before a cancel event related to the old event is sent. The local time is rollbacked to the old event's time. And finally, cancel events are send to affected LPs. After finishing processing what-if queries, the simulation starts from the rollbacked time in the same way as the optimistic PDES. In the repeating execution, events, cancel events and states are sometimes required to load from storage unlike the usual optimistic PDES. Algorithm 2 shows a mechanism to load the events, cancel events and states during receiving events. I extend the optimistic PDES as showed in algorithm 2 (line 3 – 24). In the exact-differential simulation, received events from other LPs are once buffered before they are inserted to event queues (line 1). If a new received event has less received time than minimum loaded time, which is initialized as infinity, then the stored events, cancel events and states are loaded from the storage (line 7 – 9) before they are inserted to the queues (line 14 – 21). After that, the minimum loaded time is updated to the new received time (line 22), and then the new received event is inserted to the event queue as usual (line 25).

Algorithm 1 Query Processing Flow.

```
1:  while hasWhatIfQuery() do
2:    query ← getWhatIfQuery()
3:    if query.type = ADD then
4:      newEvent ← query.event
5:      insert(newEvent)
6:      rollback(newEvent.time)
7:      sendCancelsToNeighbors()
8:    else if query.type = DELETE then
9:      oldEvent ← query.event
10:     delete(oldEvent)
11:     sendCancel(oldEvent)
12:     rollback(oldEvent.time)
13:     sendCancelsToNeighbors()
14:   end if
15: end while
16: while getGlobalTime() < TIME_TO_FINISH do
17:   reprocess unprocessed events with optimistic PDES
18: end while
```

3.1.2 Static Analysis

In this part, I illustrate efficiency of the repeating execution compered to a naive way. In the repeating execution, a main factor of its performance is how often redundancy events are skipped to process. To clarify the performance improvement, I first define speed up based on the number of processing events and redundancy reduction rate. After that, I discuss detail on the redundancy reduction rate in the repeating execution.

Speed Up

Let E be a set of events; E_{all} be a set of all events; $t_{sim}(\cdot)$ be execution time of processing events; and $t_{init}(\cdot) = t_{init\_local}(\cdot) + t_{init\_global}$ be execution time of initiation, where $t_{init\_local}(\cdot)$ is initializing events time and t_{init_global} is initializing time including state initialization and other initialization independent of events.

Algorithm 2 Receive Event Processing Flow.

```
1:  while receiveEventBuffer.isEmpty() do
2:    newEvent ← receiveEventBuffer.dequeue()
3:    /* Extended Part */
4:    if newEvent.time < store.minLoadedTime then
5:      from ← newEvent.time
6:      to ← store.minLoadedTime
7:      oldEvents ← store.getEvent(from, to)
8:      oldCancels ← store.getCancel(from, to)
9:      oldStates ← store.getState(from, to)
10:     while oldEvents.isEmpty() do
11:       loadedEvent ← oldEvens.dequeue()
12:       eventQueue.insert(loadedEvent)
13:     end while
14:     while oldCancels.isEmpty() do
15:       loadedCancel ← oldCancels.dequeue()
16:       cancelQueue.insert(loadedCancel)
17:     end while
18:     while oldStates.isEmpty() do
19:       loadedState ← oldStates.dequeue()
20:       stateQueue.insert(loadedState)
21:     end while
22:     store.minLoadedTime ← from
23:   end if
24:   /* Extended Part End */
25:   eventQueue.insert(newEvent)
26: end while
```

The execution time of whole simulation t_{whole} is represented as following.

$$\begin{aligned} t_{whole} &= t_{sim}(E_{all}) + t_{init}(E_{all}) \\ &= t_{sim}(E_{all}) + t_{init\_local}(E_{all}) + t_{init\_global} \end{aligned}$$

I represent T_{n_diff} as execution time of n times repeating exact-differential simulation.

$$T_{n\_diff} = \sum^{n} t_{sim}(E_{re}) + t'_{init}$$

, where $E_{re}(\subseteq E_{all})$ is a set of reprocessing events and t'_{init} is initialization execution time in the exact-differential simulation. Actually, the global initialization is required to execute only one time. Thus, I can represent t'_{init} as follows.

$$t'_{init} = \sum^{n} t_{init\_local}(E_{re}) + t_{init\_global}$$

As the result, T_{n_diff} is represented as following.

$$\begin{aligned} T_{n\_diff} &= \sum^{n} t_{sim}(E_{re}) + t'_{init} \\ &= \sum^{n} t_{sim}(E_{re}) + \sum^{n} t_{init\_local}(E_{re}) + t_{init\_global} \\ &= \sum^{n} \{t_{sim}(E_{re}) + t_{init\_local}(E_{re})\} + t_{init\_global} \end{aligned}$$

I assume processing and initializing time per event is constant. Then, I can represent the execution time by one event processing time t_{ev} as follows.

$$\begin{aligned} t_{sim}(E) &= |E| \cdot t_{ev} \\ t_{init\_local}(E) &= |E| \cdot t_{init\_ev} \end{aligned}$$

, where $|E|$ is the number of elements in E and $t_{ev} = t_{sim}(\{e\})$, $t_{init\_ev} = t_{init\_local}(\{e\})$ $(e \in E)$.

To simplify the discussion, I also assume the number of whole events (E_{all}) and reprocessing events (E_{re}) is constant even if scenarios are changed. Then, the sum of events processing time

is represented as following.

$$\begin{aligned}\sum^{n} t_{sim}(E_{all,re}) &= n \cdot |E_{all,re}| \cdot t_{ev} \\ \sum^{n} t_{init\_local}(E_{all,re}) &= n \cdot |E_{all,re}| \cdot t_{init\_ev}\end{aligned}$$

I define speed up as the division of naive way's execution time ($T_{n\_whole} := \sum^{n} t_{whole}$) by my proposal one ($T_{n\_diff}$). The speed up is represented as follows.

$$\begin{aligned}(Speed\ Up) &= \frac{T_{n\_whole}}{T_{n\_diff}} \\ &= \frac{\sum^{n}\{t_{sim}(E_{all}) + t_{init\_local}(E_{all}) + t_{init\_global}\}}{\sum^{n}\{t_{sim}(E_{re}) + t_{init\_local}(E_{re})\} + t_{init\_global}} \\ &= \frac{n\{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}\}}{n\{|E_{re}|t_{ev} + |E_{re}|t_{init\_ev}\} + t_{init\_global}} \\ &= \frac{n\{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}\}}{n\{r|E_{all}|t_{ev} + r|E_{all}|t_{init\_ev}\} + t_{init\_global}} \\ &= \frac{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}}{r|E_{all}|t_{ev} + r|E_{all}|t_{init\_ev} + t_{init\_global}/n}\end{aligned}$$

, where r is *redundancy reduction rate* defined as $r := |E_{re}|/|E_{all}|$. In the case that the execution is repeated enough much times, t_{init_global}/n can be ignored.

$$\begin{aligned}(Speed\ Up) &\sim \frac{|E_{all}|t_{ev} + |E_{all}|t_{init\_ev} + t_{init\_global}}{r|E_{all}|t_{ev} + r|E_{all}|t_{init\_ev}} \\ &= \frac{1}{r} \cdot \left(1 + \frac{t_{init\_global}}{|E_{all}|(t_{ev} + t_{init\_ev})}\right) \qquad (*)\end{aligned}$$

This result means that in the case that the global initialization time is much shorter than the event processing and initialization time, or in the case that there are many events to be processed, the speed up is nearly proportional to $1/r$. The more redundancy is reduced, the more speed up it is. On the other hand, in the case that the global initialization cannot be ignored compared to the event processing and initializing time, the speed up depends on the ratio of the global initialization time by the event processing and initializing time, and on the number

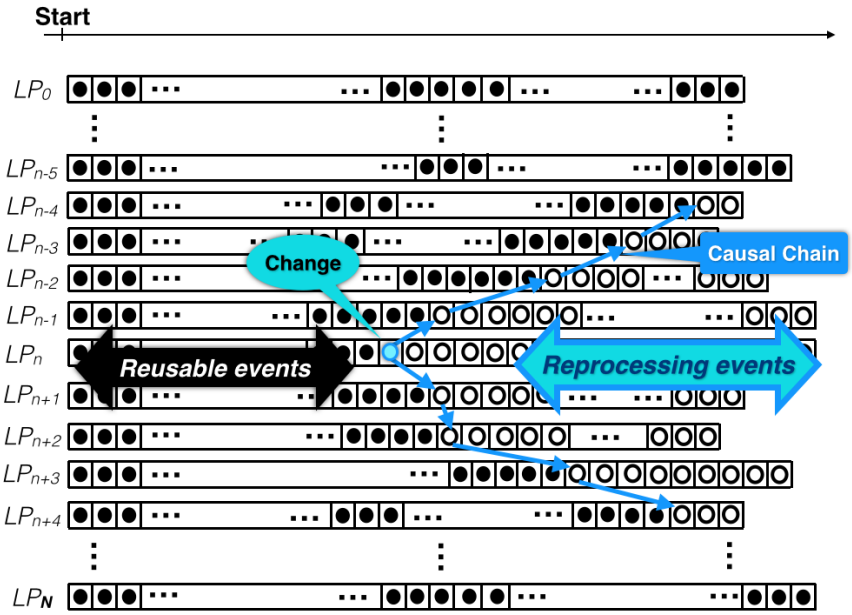

Figure 3.1.1: Reusable Events and Reprocessing Events.

of all events.

Redundancy Reduction Rate

The rest of the section focuses on the redundancy reduction rate, namely parameter r as stated above. Let lp_i be a logical process and LP be a set of logical processes. I assume there are l logical processes in simulation. The LP is represented as following.

$$LP := \{lp_i | i = 0, 1, 2, ..., l-1\}$$

I define a function that makes a causal chain from one event as $f : E \to E^l$.

$$f(e) := \{f_0(e), f_1(e), f_2(e), ..., f_{l-1}(e)\}$$

, where $f_i(e)$ is a function from an event to the earliest affecting event in lp_i. Also, let Ei_{re} be an event after $f_i(e)$.

$$Ei_{re} := \{e | e \geq f_i(e)\}$$

The parameter r is represented like that (Figure3.1.1).

$$r = \frac{|E_{re}|}{|E_{all}|} = \frac{\sum_{i=0}^{l-1} Ei_{re}}{|E_{all}|}$$

3.2 Implementation of Exact-Differential Simulation

In this section, I show system implementation. For implementation, there are two requirements to be satisfied. First, my proposal is for actual city-scale or country-scale traffic simulation. Its implementation needs to be scalable to large-scale and to be run in parallel. Second, as I showed in Section 3.1, my proposal needs to store all processed events and intermediate states in the initial whole simulation. To meet these requirements, my system is designed as the extension of an optimistic parallel traffic simulator.

3.2.1 Overview

Figure 3.2.1 shows the system overview. My simulator is executed on distributed environment and includes three modules for traffic simulation and one module for storing events and states: application, Time Warp layer, communicator and local storage.

In application, actual simulation logic and algorithms are constructed. The exact-differential simulation mechanism is fully independent of the application code such as modeling of traffic simulation or logic of a vehicles' behavior. The application gets an event and simulation state (namely the state of junction and roads) from Time Warp layer before processing the event based on the vehicles' behavior algorithm, and then returns a new event and a changed state because of the processing event.

The Time Warp layer, which is a core module of my system, consists of a LP manager, LPs and a local storage. Each LP has an event queue, a cancel queue and a state queue just the same way as optimistic PDES. The LPs are managed by a thread pool in LP manager because

there are some load imbalances between LPs in the traffic simulation. For example, such load imbalance is happened when in some roads there are a lot of vehicles to be process while in other roads there are few vehicles to be processed. The LP manager also manage the partition of the simulation, that is, the meta-data of each LP. Also, the LP manager controls the access to the local storage, where all processing events and intermediate states are stored in the initial whole execution.

The communicator controls node to node communication using MPI as well as inner process communication between LPs in the same node.

3.2.2 Initial Whole Execution

Figure 3.2.1 also shows the flow of initial execution per node. In the initial execution, the simulator at first inputs states and scenarios before simulation. The states data are deployed to each node based on the defined partitioning. After inputting, the simulation starts and processes events (Figure 3.2.1). In the initial execution, the LP manager allots a free thread to a LP to process events. The LP passes the earliest unprocessed event to the application before the event is processed in the application. After that, the LP gets a new generated event and a changed state from the application. The new event is sent to a new destination LP via communicator according to a partition discussed later. Also a new cancel event is stored in the cancel queue. If the event has to be sent to other node, the event is communicated via MPI. On the other hand, if the new destination LP is in the same node, the event is sent as inner process communication. Such new sending event is received in the communicator and passed to the destination LP.

Unlike usual optimistic PDES system, events with smaller time stamp than global time are not released after global synchronization. Instead, such events are stored in the local storage for later exact-differential execution.

3.2.3 Exact-Differential Simulation in Repeating Execution

In repeating exact-differential execution (showed in Figure 3.2.2), the system first inputs a changing query and then distributes to a destination LP. The query is received in the LP via communication layer before accessing the local storage to load the changing event and all events affected by the changing event. These events (the changing event and the affected events) are inserted to an event queue to process again. After that, the LP passes the earliest unprocessed event to

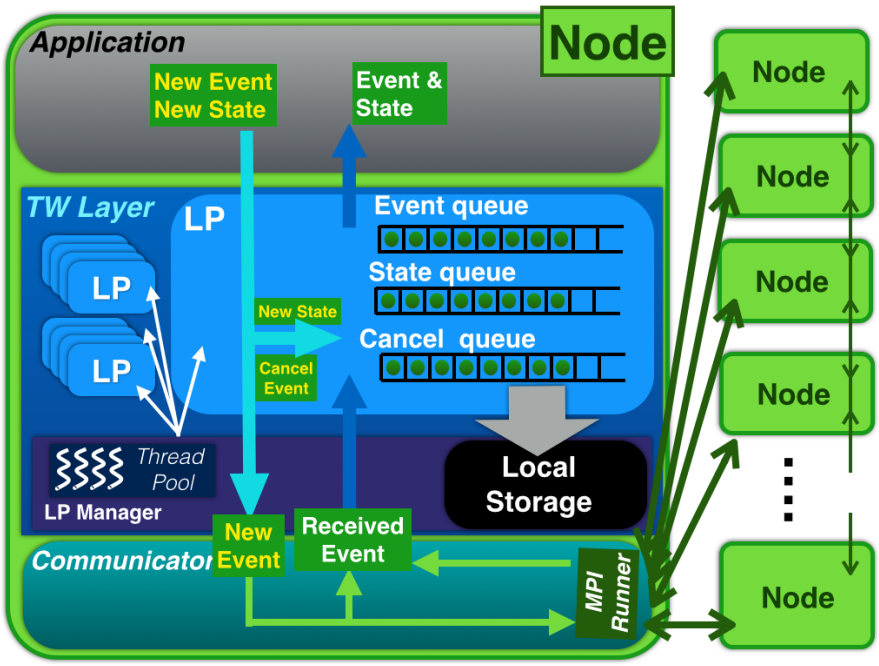

Figure 3.2.1: System Overview and Initial Whole Simulation.

the application before gets the new generated event and changed state by the application. The new generated event is sent to the destination LP via communicator and the new state is stored to a state queue. After communicator receives the new event, the new event's received time is checked and the affected events are loaded if the time is less than minimum loaded time as discussed in Section 3.1.

3.3 Large-Scale Traffic Simulation with PDES

In this section, I describe the modeling of traffic system. In a first half of this section, I illustrate how to simulate local vehicles' behavior, that is, how to decide their route at junctions and vehicles' speed on a road. In the other half, I show the model of the global interaction with each other on the road map.

There are two requirement in the modeling way like the system requirement discussed in Section 3.2. First, the traffic simulation needs to be scalable to city-scale or country-scale. Second, the traffic simulation is required to be modeled on the top of optimistic PDES since the exact-differential simulation is totally based on the optimistic PDES as I discussed in Section

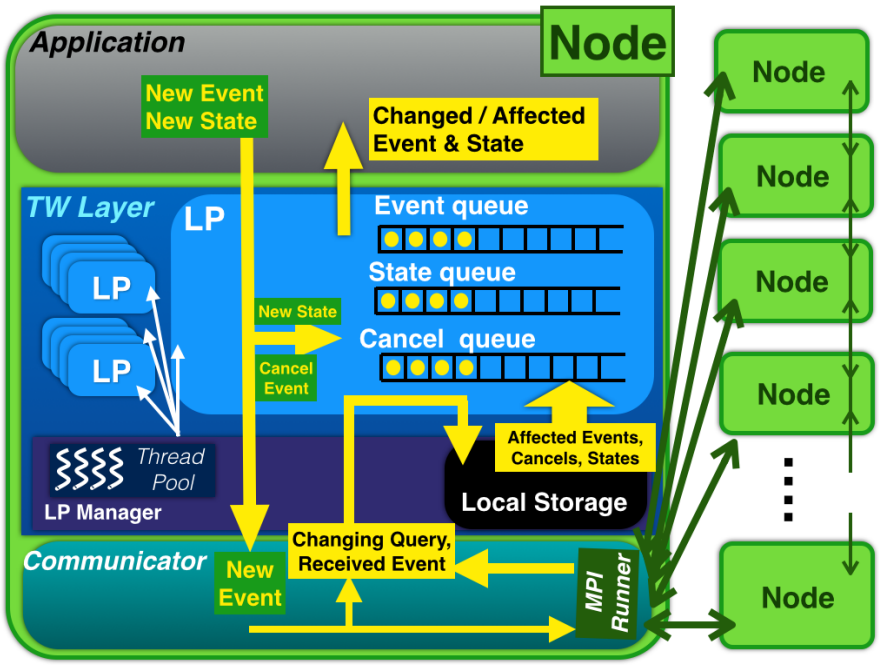

Figure 3.2.2: Exact-Differential Simulation.

3.1. To meet such requirement, the model of my traffic simulation is based on IBM Megaffic [65, 66, 88] and SCATTER [72, 97].

3.3.1 Individual Vehicle's Behavior

The individual vehicle's behavior is based on Megaffic, where it optimises drivers' decisions by estimating some of the parameters of the model from probe-car data before actual simulation execution, differentiating Megaffic from many other traffic simulators 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. This is because in large-scale traffic simulation, the processing time of individual vehicles becomes the main bottleneck of the simulation and has to be reduced as much as possible.

In the same way as Megaffic, a vehicle's track of junctions from origin to destination is all fixed before execution by estimation with some defined behavior model, for example, shortest path or minimum hops of junctions. After that, in the execution, the vehicle's speed, traveling time

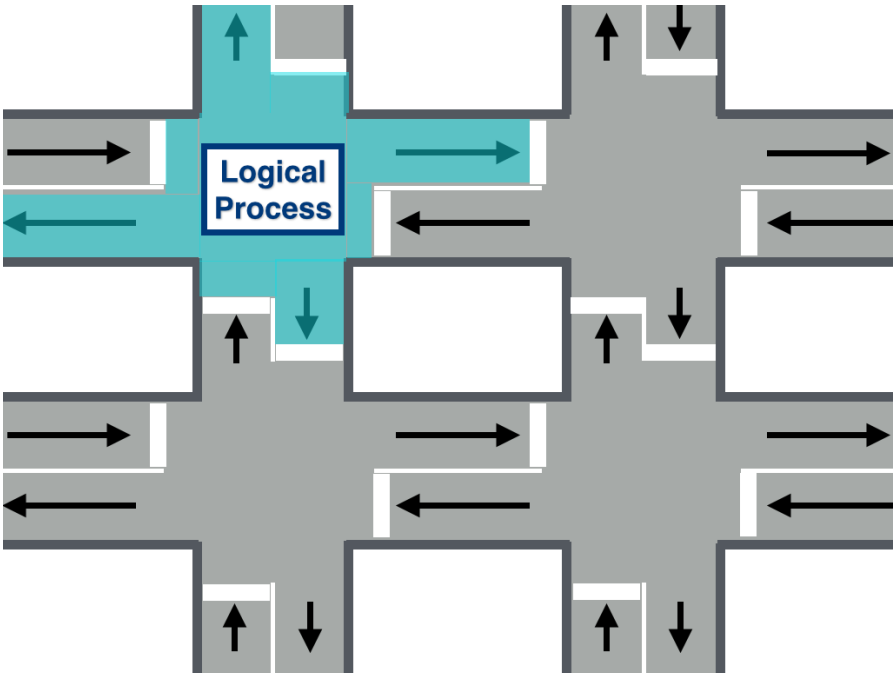

Figure 3.3.1: Road Map and a Logical Process Unit.

to next junction and selection of a lane are calculated based on some defined behavior models. Finally when the vehicle reaches its destination, it is removed from the simulation.

3.3.2 Interaction of Vehicles on the Road Map

The global interaction of vehicles around the road map is based on SCATTER. Thus, I mainly use the optimistic PDES technique for parallelization including synchronization. As SCATTER, I represent a vehicle's track as a sequence of events. One event is represented as vehicle's moving from one junction to a next junction. A unit of a logical process in the optimistic PDES is a set of a junction and its outgoing roads (Figure 3.3.1). The arrival timing to junctions and synchronization with other junctions are fully controlled by the optimistic PDES.

Also, the road map is partitioned in advance by the k-ways graph partitioning algorithm [53] with METIS [52], which is the software including the k-ways algorithm implementation.

3.4 Evaluation of Exact-Differential Large-Scale Traffic Simulation

In this section, I describe evaluation of the exact-differential simulation with Tokyo traffic scenarios.

There are two topics to evaluate: efficiency and performance. In the efficiency evaluation, I evaluate how the exact-differential simulation can reduce the number of redundant events in repeating execution. I experiment with two types of changing scenarios. The first one is the case to change vehicle's scenarios. I change a vehicle's track in repeating execution. The second one is the case to change the road map. I change the parameter of one of LPs (one junction and its out-going roads) in repeating execution. The efficiency evaluation shows, in high level, how the exact-differential simulation "potentially" can improve the performance. On the other hand, in performance evaluation, I show the "actual" performance improvement in my implementation, where I evaluate the elapsed time compared to the whole simulation as well as the scalability and parallelization of the simulator.

3.4.1 Efficiency Evaluation

In this part, I illustrate the efficiency of the exact-differential simulation with Tokyo traffics in 3 hours. Table 3.4.1 shows the simulation scenario. I simulate the traffic in Tokyo bay area with 161,364 junctions and 203,363 roads (Figure 3.4.1). Based on the Tokyo's statistical data collected by the MLIT (Ministry of Land, Infrastructure, Transport and Tourism) in 2011, totally 5000 vehicles depart from their origin in 3 hours. Each vehicle has a trip pattern which has randomly generated origin/destination. In total, this scenario generates 798,177 events to be outputted, where I use the term "outputted event" as a processed event which is fixed and never canceled in Time Warp layer. Thus, actually in Time Warp layer, over 798,177 events are handled and canceled.

In this experiment, I use Hamilton4 supercomputer (10 MPI processes) in Durham University but the result of efficiency here is independent of the cluster environment. A result influenced by the environment (thus, elapsed time) is discussed in the performance evaluation.

Road Map	Tokyo Bay Area (Figure 3.4.1)
– # of Junctions	161,364
– # of Roads	203,363
Scenario of Tokyo's Traffic	
– Sum of Departing Vehicles	5,000 (3 hours)
– Trips Origin/Destination	Random
Result of Whole Simulation	
– Total Outputted Events	798,177

Table 3.4.1: Traffic Scenario.

Evaluation with Changing a Vehicle's Track

Here, I study the impact of changing vehicles on repeating execution. In this evaluation, I change one of the 5000 vehicles' track paths and then execute the exact-differential simulation. The new track path is generated randomly so that the number of hops is unchanged. I totally change all 5000 vehicles' track path respectively and count the number of events affected by the change, which need to be outputted in the exact-differential simulation.

Figure 3.4.8 shows the number of outputted events in the changes. I plot the results in departing time order, but as you can see, I cannot find the impact of departing time in the exact-differential simulation. In Figure 3.4.6, I compare the worst case and average case to the whole simulation. In average, the number of outputted events in the exact-differential simulation is 44,261, which is only 5.5 % of the whole simulation. Even in the worst case, the number of outputted events is 233,749, which is 29.2 % of the outputted events in the whole simulation.

Evaluation with Changing a Road Parameter

In this evaluation, I show the efficiency of the exact-differential simulation with changing a speed limit parameter of the road map. I change the speed limit of one LP and execute the exact-differential simulation from the beginning, where I randomly pick up about 1 % of all 161,364 LPs (1,600 LPs).

Figure 3.4.9 shows the number of outputted events in each LP ordered by junction ID, where close numbers are roughly located near points in the actual road map. As you can see, the number of outputted events in the differential simulation has a big gap, that is, some junctions affect the large number of events while the others bring about few events. This is because in Tokyo

Figure 3.4.1: Road Map of Tokyo Bay Area.

there are two types of junctions; the first one is the hub, where a lot of vehicles cross over and the other one is a junction, where only few vehicles enter. In Figure 3.4.7, I compare the worst case and average case to the whole simulation. In average, the number of outputted event is 61,206. Thus, the events to be outputted are only 7.6 % of whole simulation. Even in the worst case, the number of outputted events in the differential simulation is 297,181, which are 37.2 % of the outputted events in whole simulation.

To sum up the efficiency evaluation, the exact-differential can reduce over 90% of whole events in average (94.5% in vehicle changes, 92.4% in road changes), and even in the worst case it can reduce over 60% of whole events (70.8 % in a vehicle change, 62.8% in a road change).

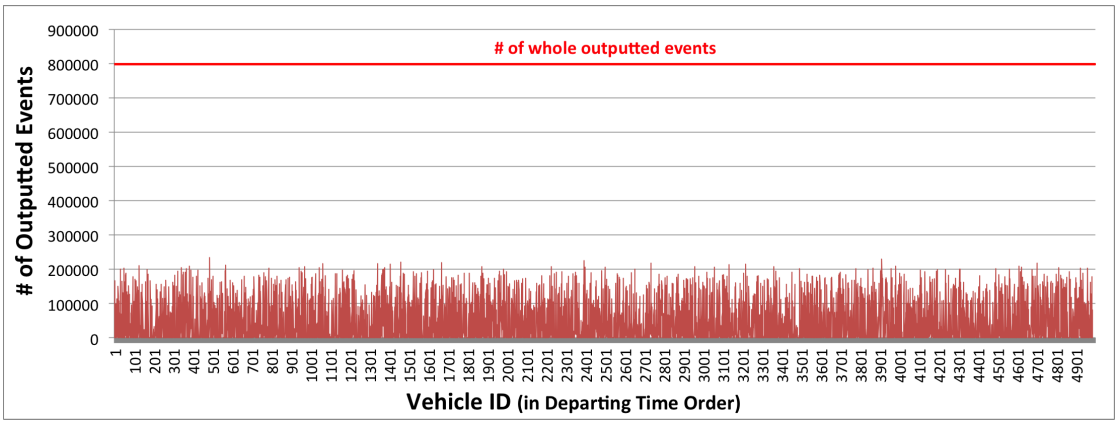

Figure 3.4.8: Number of Differential Outputted Events in a Vehicle Change.

3.4.2 Performance Evaluation

In this part, I show the actual performance of the exact-differential simulation with my simulator. I use the worst and average cases of road changing scenarios as I evaluated above.

Table 3.4.2 shows a summary of the evaluation environment. My simulator is implemented by C++ and MPI with a hybrid parallel architecture, where each MPI process includes multiple threads. I run the simulator with TSUBAME 2.5 Supercomputer in Tokyo Institute of Technology, where there are 12 processors per node with 54GB memory and connected by QDR InfiniBand.

Service	TSUBAME 2.5 in Tokyo Tech.
CPU	Intel Xeon X5670/2.93GHz × 2
Memory	54GB per Node
Network	QDR InfiniBand Interconnect
OS	SLES 11 SP3
MPI	Open MPI 1.6.5

Table 3.4.2: Cluster's Configurations.

I change the number of processors according to the table 3.4.3. In the simulator, it needs 2 processors per node at minimum because MPI thread and event processing thread are separated in my simulator. Also, it needs 2 MPI process at minimum because of the implementation. Thus, I start from 4 processors and increase the number of cores to 192 (including 16 nodes with 12

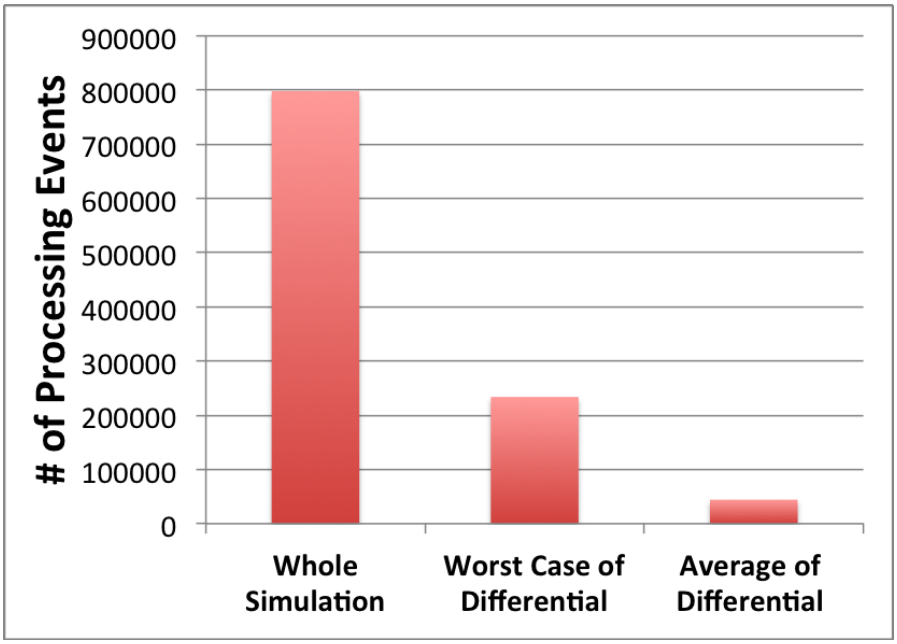

Figure 3.4.6: Number of Differential Outputted Events in a Vehicle Change.

threads per node).

Nodes	Threads per Node	Processors
1	4 (2 MPI threads)	4
1	6 (2 MPI threads)	12
2	12 (2 MPI threads)	24
4	12 (4 MPI threads)	48
8	12 (8 MPI threads)	96
16	12 (16 MPI threads)	192

Table 3.4.3: Number of Nodes, Threads and Processors.

According to the efficiency result, I pick up the worst case of a road changing scenario needed to be outputted 297,181 events and the average case needed to be outputted 61,530 events, and then evaluate the elapsed time respectively.

Figure 4.3.3 shows a strong scaling of the simulator. From 12 processors to 24 processors, the performance becomes once worse because from 24 processors, node to node communication occurs, which is higher cost than a inner node communication. From 24 processors, the

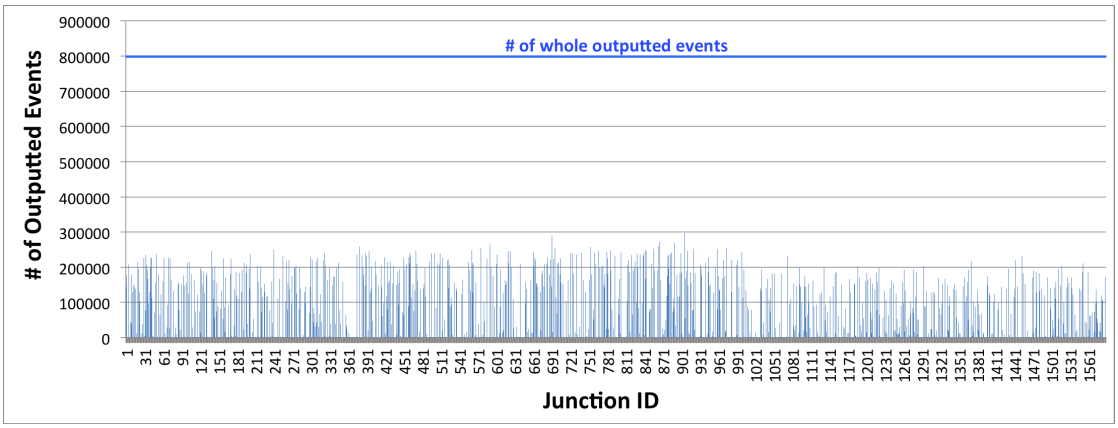

Figure 3.4.9: Number of Differential Outputted Events in a Road Change.

performance becomes better according to the number of processors. In the whole simulation, the elapsed time is improved to be 21.7 % of 4 processors elapsed time. In the exact-differential simulation of the worst case scenario (297,181 outputted events), the elapsed time is improved to be 33.6 % of 4 processors elapsed time. Also, in the exact-differential simulation of the average case scenario (61,530 outputted events), the elapsed time is improved to be 49.7 % of 4 processors elapsed time.

Figure 3.4.11 shows a speed up from the whole simulation, defined as follows.

$$\frac{(\textit{Elapsed Time in Whole Sim.})}{(\textit{Elapsed Time in Exact-Differential Sim.})}$$

The exact-differential simulation achieves at most 7.26 times as much speed up in average case and 2.26 times speed up in the worst case as the whole simulation (4 processors). The speed up decreases according to the number of processors. There are still gaps from ideal cases (showed dotted line in Figure 3.4.11), calculated by (*) in Section 3.1, where I assume $(\textit{Speed Up}) = 1/r$ because $|E_{all}|$ is enough bigger than t_{init_global}. This is because the overhead of the processing events increases according to the number of processors as discussed later.

Figure 3.4.12 shows elapsed time per outputted event. The elapsed time becomes worse if the number of outputted events decreases in the same number of processors, because the effect of parallel processing decreases in smaller input size.

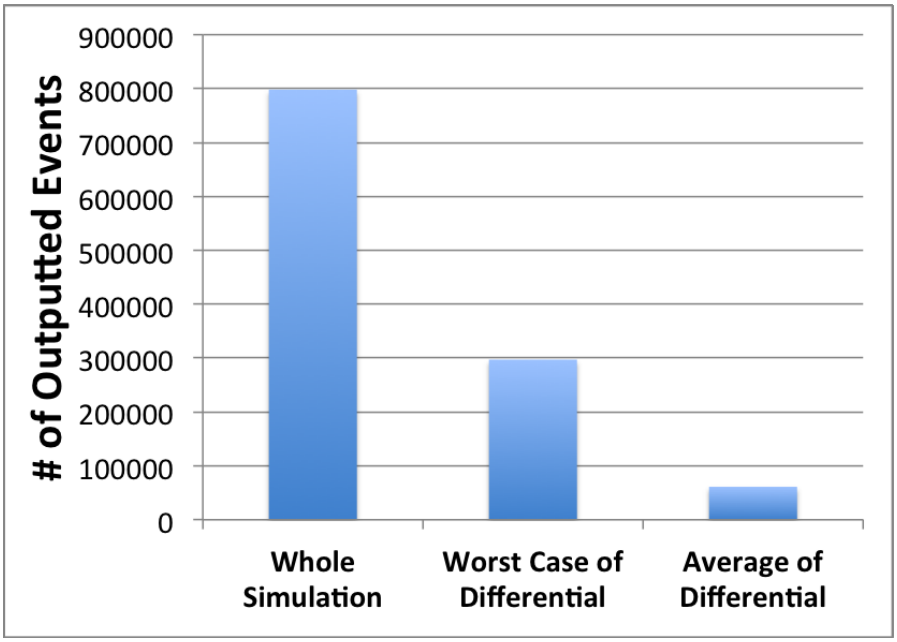

Figure 3.4.7: Number of Differential Outputted Events in a Road Change.

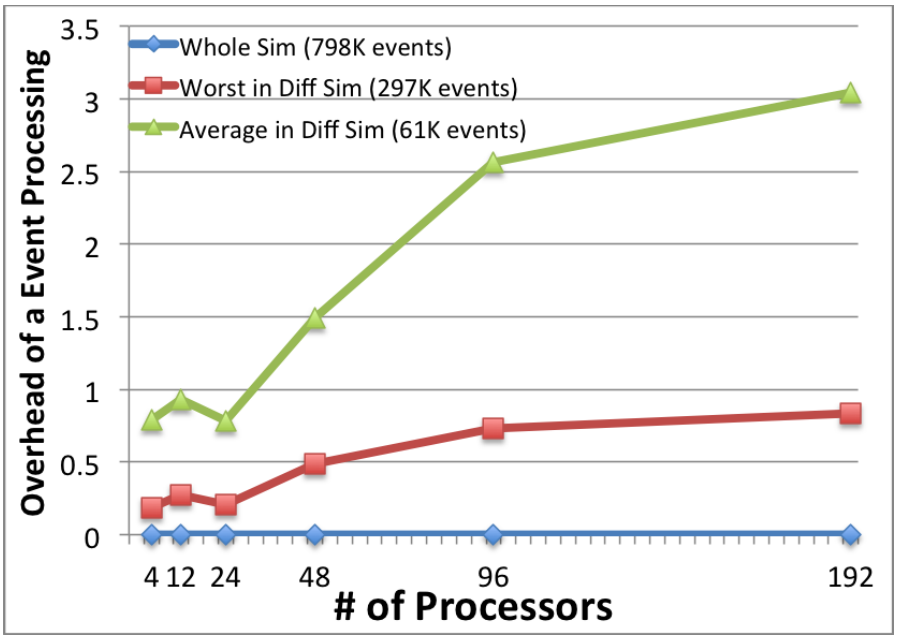

Figure 3.4.13: Overhead of Differential Simulation Compared to Whole Simulation.

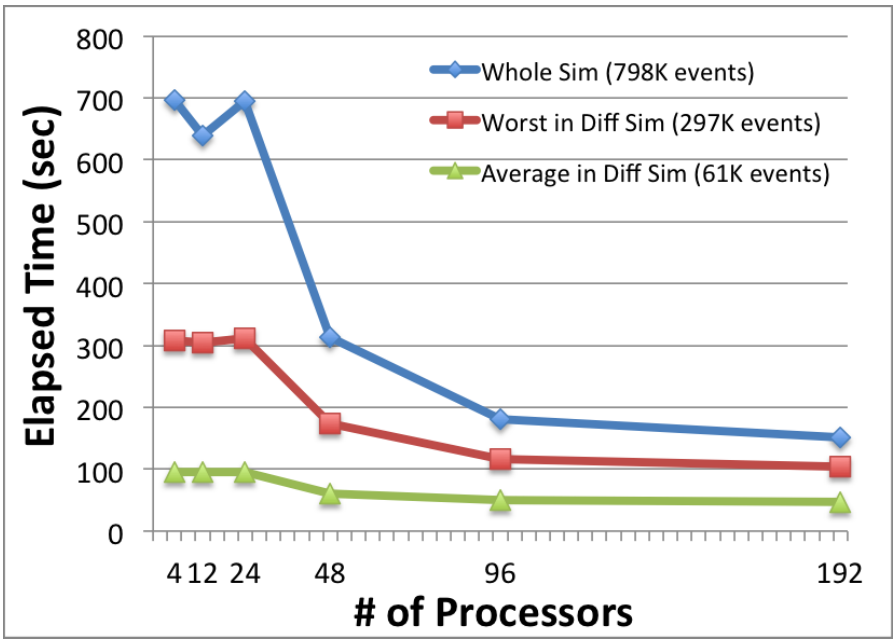

Figure 3.4.10: Strong Scaling of Simulation.

Figure 3.4.13 shows the overhead of repeating execution compared to the whole simulation. The overhead of a differential simulation becomes worse according to the number of processors, where the overhead is defined as follows.

$$\frac{(\textit{Time per Event in Whole Sim}) - (\textit{Time per Event})}{(\textit{Time per Event in Whole Simulation})}$$

This is because if the more processors there are, the more events are needed to be processed for synchronization, namely, the number of cancel events and the frequency of rollback increase because of the increasing of processors. Also, the overhead increases in the fewer events because the ratio of overhead increases because the elapsed time becomes faster itself in more processors.

To sum up, my simulator with the exact-differential simulation achieves a good performance improvement from the whole simulation. It achieves 7.26 times as much performance improvement in average and 2.26 times improvement even in the worst case as the whole simulation with a road changing scenario. Also, it achieves a good strong scaling till 48 processors in the worst case. Although the strong scaling in the average case is still limited and it has much overhead compared to the whole simulation, the exact-differential simulation can achieve better

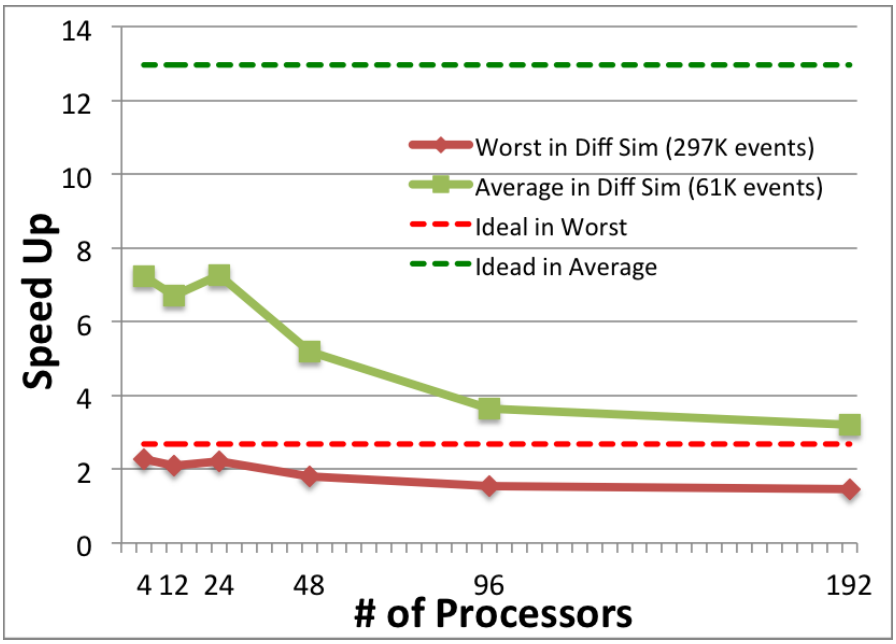

Figure 3.4.11: Speed Up from Whole Simulation.

elapsed time even with fewer processors than the whole simulation.

3.5 Related work

Updateable simulation [33] is highly related in our work. In this research, they propose the technique to simulate a part of events and states in repeating execution by canceling and reprocessing events in the similar way as optimistic PDES technique. In the proposal, they define the *reuse function*, which estimate the number of reprocessing events in repeating simulation and it enables to reprocess part of simulation efficiently. Not only the target domain of application, there are also mainly two differences between the proposal and our approach. First, our approach always achieves the exactly same results as the original whole simulation. This is possible by assuming the separated state (LP) and storing all of intermediate states instead of using the reuse function, which is too general to always ensure the exactly same results. Second, in our proposal, we evaluate the simulator on much larger scale than their proposal. This is one of the main contributions in our research.

Cloning techniques [27, 45–47, 98] are also ways to reuse the intermediate simulation states

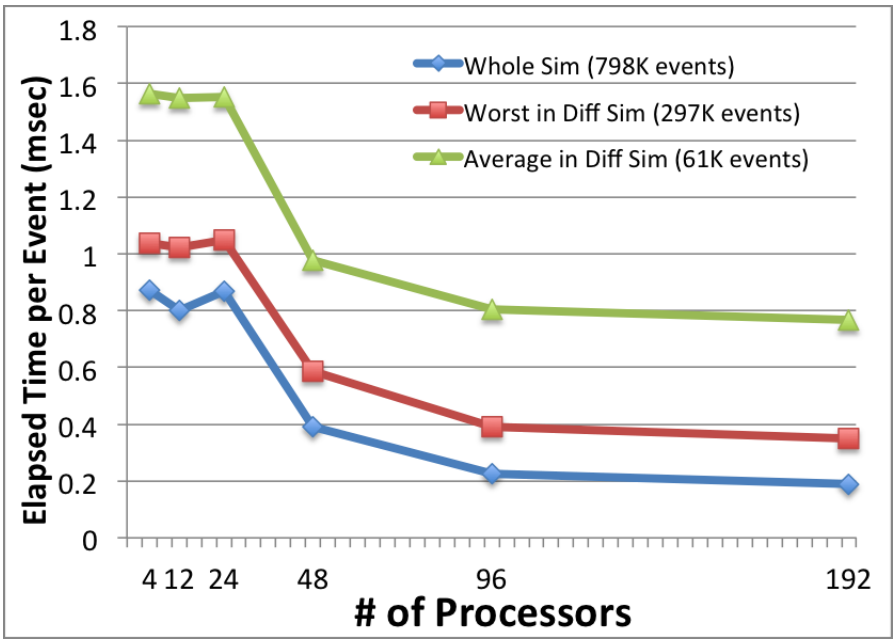

Figure 3.4.12: Elapsed Time per Outputted Event.

and events for efficient repeating simulation. In the cloning of simulation, the simulation states in some decided time is copied and from the decided time the simulation is branched with different parameters or scenarios. The difference from our proposal is that such technique does not have "differential" feature, namely, the cloning technique cannot simulate a part of whole state but they can only simulate wholly from intermediate time.

Reducing the scenario patterns or parameter spaces is the direct and general way to speed up the repeating execution. There have been previous researches with such techniques, illustrated in [20–22]. For example in [21], the all patterns of scenarios are filtered by random sampling and clustering algorithm before simulation for getting the optimum scenario pattern. In [22], they use GA (genetic algorithm) to pick up the appropriate scenario patterns to be simulated in agent-based simulation.

3.6 Summary

In this chapter, I proposed the exact-differential simulation for large scale agent-based traffic simulation, which enables to reduce the redundant events in repeating the simulation. In my

evaluation, I illustrated how many events are required to be processed in the exact-differential simulation and show a big improvement in reducing the number of processing events. Actually, in the case of vehicles' path changes, I can reduce 94.5 % of processing events in average and even in the worst case, I can reduce 70.8 % of processing events compared to a naive way. In the case to change the parameter of one LP, I can reduce 92.4 % in average and 62.8 % in the worst case. Also with my traffic simulator, I show 7.26 times as much performance improvement in average and 2.26 times improvement even in the worst case as the whole simulation with changing a road's speed limit.

4

Large-Scale What-If Analysis

In my work *exact-differential simulation* [41], I achieved to improve the execution time of the repeating simulation by reducing redundancy of events processing. However, there is still a big problem in the number of repeating times. The number of repeating times for what-if simulation increases as the simulation scale becomes large. For example, when we simulate bigger city than Tokyo (the number of junctions is 770K as mentioned above), the number of repeating times becomes over 770K when we simulate what happens if one of the junctions is blocked.

In this chapter, I propose new techniques to efficiently repeat such a large number of simulation tasks for reducing total analyzing time. There are two main ideas in my proposal: *what-if scenario filtering* and *exact-differential cloning*.

- *What-if Scenario Filtering*: This is a technique to reduce the number of scenarios itself by filtering meaningless scenarios which do not impact results of simulation.

- *Exact-Differential Cloning*: This is a technique to run exact-differential simulating tasks in parallel. I show my new exact-differential simulator which stores events and states logs in a distributed file system instead of a local storage and which enables to execute exact-

differential simulating tasks in parallel.

4.1 Efficient Repeating of What-if Simulation

In this section, I describe an overview of our proposal, showed in Figure 4.1.1. The key ideas of my proposal are to reduce as much what-if scenarios as possible in advance and then to execute what-if simulation tasks in parallel. There are 3 steps to efficiently repeat what-if simulation: what-if scenarios filtering, exact-differential cloning and actual simulation executing with exact-differential simulation. In the what-if scenarios filtering, the system pre-analyzes what-if scenarios and picks up some "meaningful" what-if scenarios filtered by some defined condition (e.g. the number of affected events, departing time of the vehicles) with some mechanisms (e.g. machine learning, statistical way) after initial execution, where all processed event and state logs are stored in a database. I discuss more details on the filtering in Section 4.2. In the exact-differential cloning, a cloning controller schedules what-if simulation tasks based on the already filtered what-if scenarios and on the number of available computer resources. After that, each task clones events and states in parallel from a distributed file system which can be accessed by every machines in clusters, in the same way as loading events and states from a local storage in ScaleSim. I discuss more detail in Section **??**. After that, the exact-differential simulation tasks start to execute.

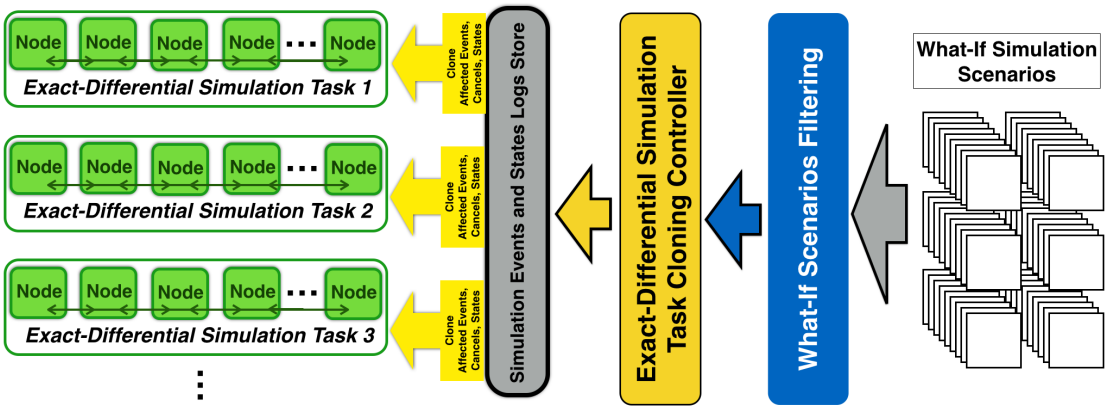

Figure 4.1.1: Overview of Large-Scale What-If Simulation with Exact-Differential Simulation.

4.2 What-If Scenario Filtering

In this section, I clarify objectives and potential on the what-if scenarios filtering and show its overview. What-if scenario filtering is a technique to reduce the number of what-if scenarios in advance of what-if simulation.

4.2.1 Objectives

My aim of the what-if scenarios filtering is to pick up only ”meaningful” scenarios for what-if simulation. For example in changing roads' speed limit, if the changing is very little and this changing hardly influences the simulation result, then this what-if scenario is ”meaningless”. On the other hand, if the changing is very large and this changing strongly influences the simulation result, then this scenario is ”meaningful”. As I discussed, the number of what-if patterns becomes large as the size of simulation. For example in Tokyo traffic simulation, there are 770K junctions and when I simulate what happens if a accident happens in each road, I need to execute simulation 770K times. Thus, it is necessary to reduce the number of scenarios as much as possible, which directly reduces total execution time of repeating simulation tasks.

Figure 4.2.1 shows a flow of what-if scenarios filtering. With input data (vehicles' origin/destination data and road map data) and historical simulation results, the system filters meaningless scenarios and picks up the meaningful scenario by some filtering technique (e.g. machine learning, statistical way). Such mechanism is highly depends on simulation models and is generally difficult to achieve but it should be feasible in traffic simulation because traffic input data (vehicle data and road map data) has relatively rich information in advance, including departing time and a track of all paths (origin/destination and all junctions to reach), and it enables to pre-analyze what-if scenarios closely. Also, I can use the simulation result in initial execution for filtering.

My main goal is to develop the traffic specific filtering, which has never been proposed in large-scale traffic simulation domain. Actually, such filtering mechanisms have been already proposed and achieved good results in other domain of simulation researches [20–22]. For example in agent-based simulation of emergency departments in hospitals [21], the all patterns of scenarios are filtered by random sampling and clustering algorithm before executing what-if simulation tasks for getting the optimum scenario. In [22], they use a machine learning method based on GA (genetic algorithm) to pick up the appropriate scenarios to be simulated in general

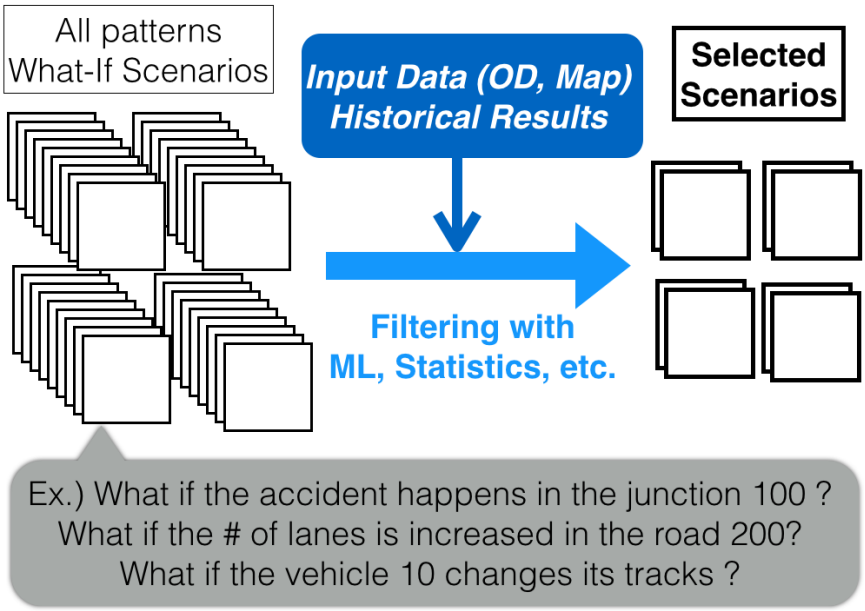

Figure 4.2.1: What-If Scenario Filtering.

agent-based simulation.

4.2.2 Preliminary Evaluation

In this part, I describe potential of the filtering by estimating evaluation with my previous work's result I discussed in Chapter 3. I evaluate how many what-if scenarios are estimated to be filtered in the previous work's situation showed in Figure 4.2.3, which represents relation between LP ID (junction ID) and the number of output events by changing the LP's parameter (junction and road parameter).

Figure 4.2.2 shows the relation between the percentage of filtered scenarios and a filter size, which implies how strict the filtering condition is, and the *filter size* is defined as follows.

> What-if scenarios predicted to generate smaller number of output events than the *filter size* are filtered and ignored to be processed.

In the evaluation, I assume that I can filter what-if scenarios by the filter size with some prediction technique. For example, I should be able to roughly predict the number of events affected

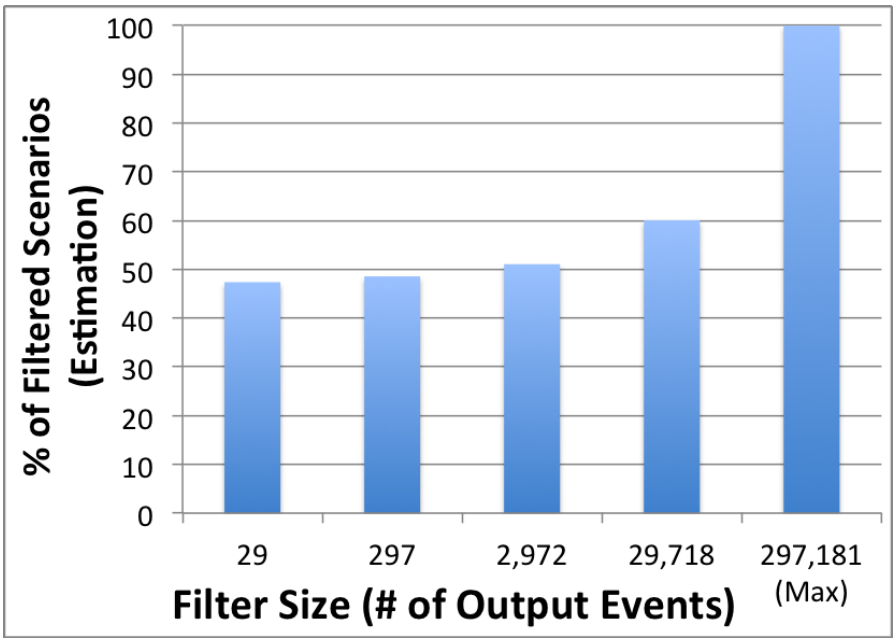

Figure 4.2.2: Estimated Ratio of Filtered Scenarios.

by the what-if scenarios with pre-analyzing the result of initial execution, which includes logs of all vehicles tracks, and I can get causal chains of all events (sequences of affected events).

As you can see in Figure 4.2.2, the percentage of filtered scenarios keeps high (over 45%) even though the filter size is very small (1/10000 of the maximum). The reason why such filtering is highly effective in the traffic simulation, especially in the scenarios (picking up one junction and changing its parameter), is that in most junctions their changes hardly affect the results because very few vehicles go through the junction. Such what-if scenarios (picking up such uncrowded junction) is filtered even though the filtering size is very small. On the other hand, there are very few "hub" junctions, which a lot of vehicles go through and whose changing strongly affects the simulation result.

4.3 Exact-Differential Cloning for Parallel Task Execution

In this section, I describe a exact-differential cloning technique to execute exact-differential simulation tasks in parallel. In the exact-differential simulation, repeating execution is independent of each other and it is basically possible to run these tasks in parallel. Thus in this section, I de-

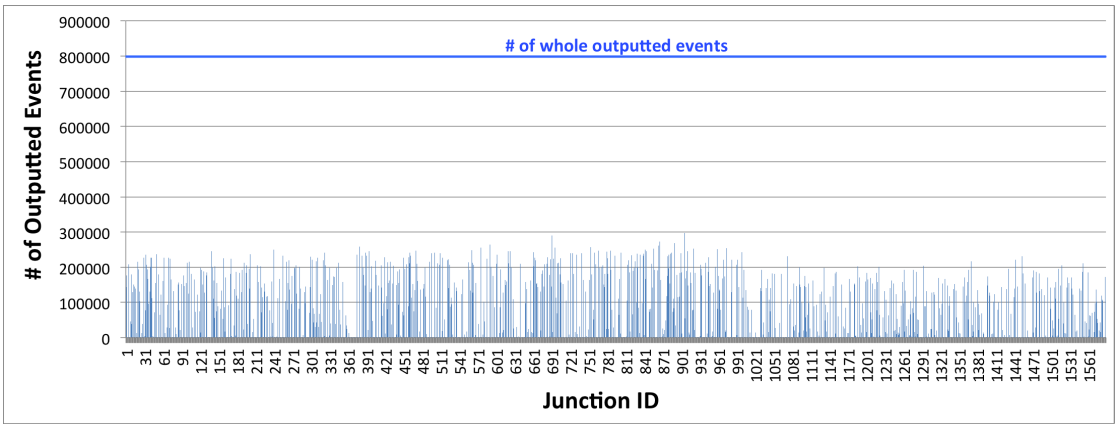

Figure 4.2.3: Number of Differential Outputted Events with Junction & Road Change.

scribe especially on how I implement a system extended from my previous exact-differential simulator.

4.3.1 Implementation

I implement the system by reconstruct the local storage in previous version (see Figure 3.2.1 and 3.2.2 in Chapter 3). In order to be accessible from multiple machines, I will use a distributed file system (e.g. Lustre, GPFS, nfs), which can be access from every machine in a cluster. Figure 4.3.1 and Figure 4.3.2 show a mechanism of the exact-differential cloning. In initial execution, the simulator stores all historical events and states logs to the database on a distributed file system instead of local storage, which can be globally accessed from every machine in a cluster (Figure 4.3.1). In repeating execution, the simulator clones affected events and states in parallel from the database on a distributed file system (Figure 4.3.2).

4.3.2 Preliminary Evaluation

In this part, I show preliminary and estimated evaluation. According to my previous work's results in Chapter 3, I estimate how the task parallel execution with the exact-differential cloning can improve the performance. First, I briefly show my previous work's result in Figure 4.3.3. Second, I state assumptions for estimation and finally, I illustrate estimated elapsed time of 161K times repeating execution, comparing 3 cases (whole simulation, previous sequential re-

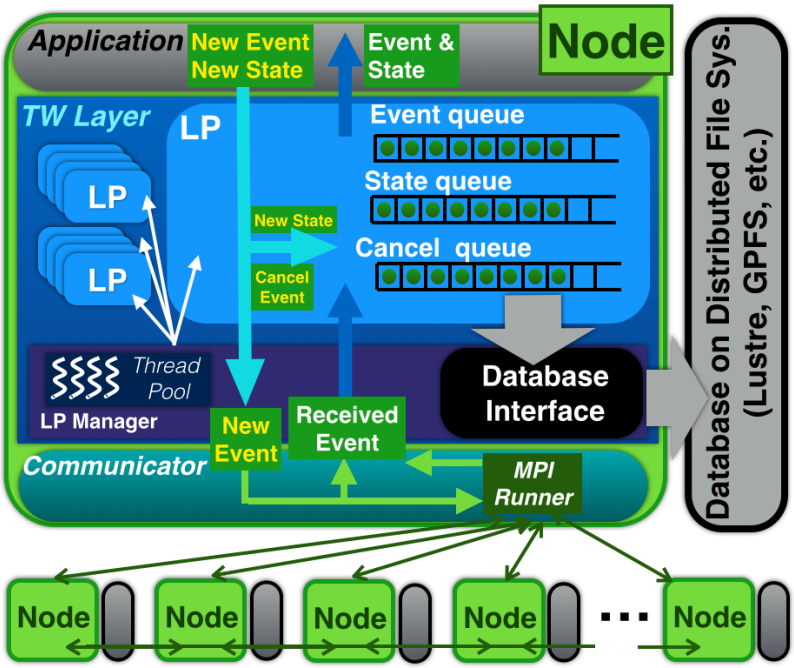

Figure 4.3.1: Storing Logs in Initial Execution.

peating, my proposing parallel repeating) in Figure 4.3.4.

Figure 4.3.3 shows strong scaling of Tokyo bay area's traffic simulation (161,364 junctions and 203,363 roads for 3 hours) with what-if scenarios of junctions' condition changing, comparing whole simulation (798K events), exact-differential simulation in the worst case (297K events), and exact-differential simulation in the average case (61K). I evaluated my simulator with 192 cores (12 cores $\times$ 16 nodes) in parallel on TSUBAME 2.5 supercomputer in Tokyo Institute of Technology.

There are 3 assumptions for this estimation to simplify the following discussion.

1. Event processing time is enough bigger than event cloning time from distributed file systems and an overhead to clone event can be ignored even though there is approximately over 10 times overhead compared to in-memory reading.

2. I can use the same number of cores and machines (192 cores, 16 nodes) in the 3 cases.

3. Elapsed time of each exact-differential execution is always same as the average case (thus the total elapsed time is equal to actual total time).

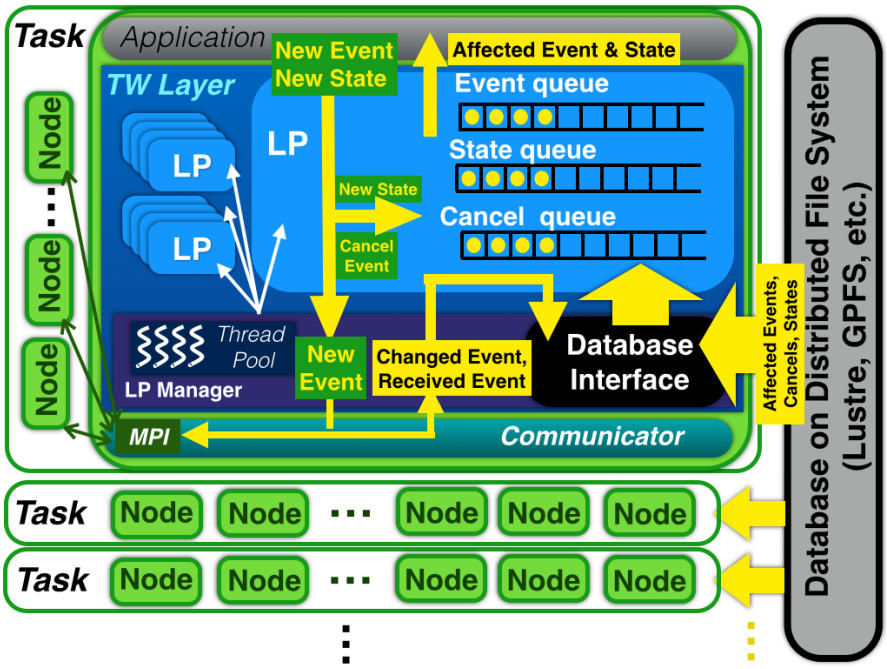

Figure 4.3.2: Cloning Affected Events and States in Repeating Execution.

Figure 4.3.4 shows the estimated elapsed time of 161,364 times repeating execution, where the 161,364 means the number of junctions in the Tokyo road map. Thus, I consider what-if scenarios where all junctions' conditions are changed respectively. I evaluate in 3 ways to run the 161K times simulation tasks.

First in whole simulation, which is the naivest way, I estimate the elapsed time to simply repeat whole simulation 161K times with 192 cores. It takes 6788 hours (= 151 seconds × 161,364 times / 3600) according to the Figure 4.3.3. Seconds in sequential repeating with exact-differential simulation, which is my previous work, I estimate the elapsed time by calculating,

$$(\textit{Elapsed time in average case with } 192 \textit{ cores}) \times 161,364 \textit{ times},$$

on the assumption that elapsed time of one time repeating is always same as the average case. It takes 2117 hours in this case according to my previous works. Finally in task parallel repeating with exact-differential cloning, which is my proposal, I estimate the elapsed time by calculating

$$(\textit{Avg. elapsed time with most efficient \# cores}) \times 161K \textit{ times} \div \frac{192 \textit{ cores}}{\textit{most efficient \#}},$$

where I fix the "most efficient" number of cores as "48" based on the strong scaling, because the performance on average case is saturated at 48 cores and it does not improve from 96 cores as Figure 4.3.3. It takes 675 hours in my proposed task parallel way, which is 31.9 % of the previous sequential way and is only 9.94 % of the naive repeating way.

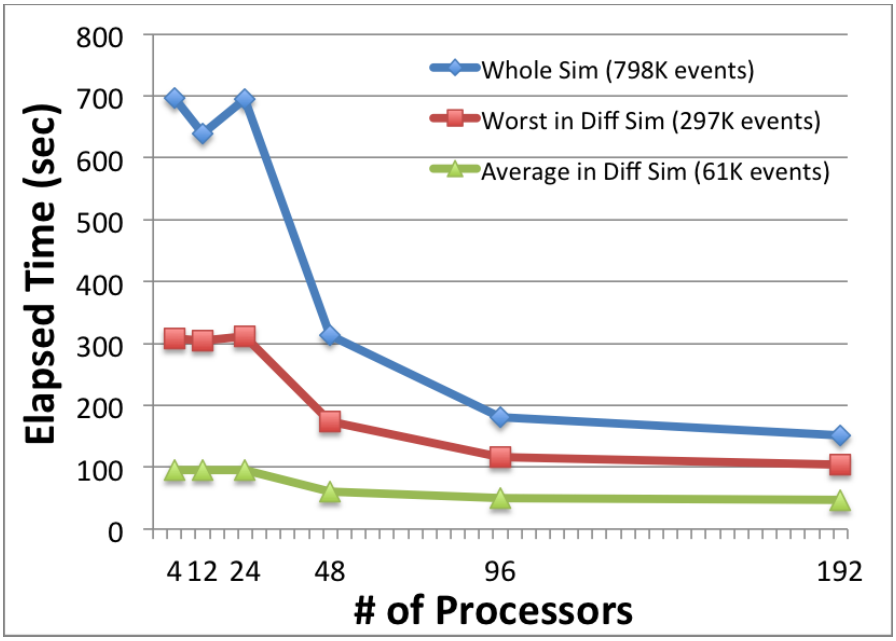

Figure 4.3.3: Strong Scaling.

The main reason of such improvement is that in the exact-differential simulation, the number of processing events is sometimes (e.g. average case) too small to be scaled well. This task parallel technique solves such imbalance between data size and the number of cores and it achieves efficient usage of the computer resources.

4.4 Summary

In this chapter, I proposed the technique for large-scale what-if simulation of traffic systems with the exact-differential simulation, including the what-if scenarios filtering and the exact-differential cloning. In what-if scenarios filtering, I clarified the objectives to reduce the number of what-if scenarios and its preliminary estimated evaluation showed such filtering technique

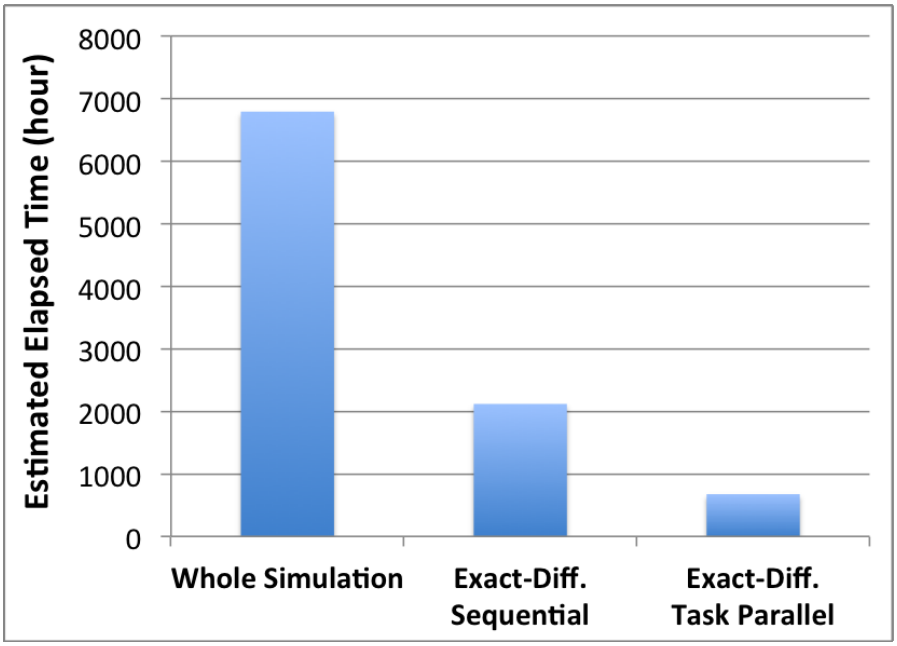

Figure 4.3.4: Estimated Elapsed Time of 161K Times Repeating Execution with 192 Cores.

should have big impacts in large-scale traffic simulation. For example of the what-if simulation changing LP (junctions & roads) in the Tokyo traffic simulation, the evaluation showed that the filtering reduces over 45 % even though the filtering condition is very week (1/10000 of maximum). In exact-differential cloning, I showed the way to extend my simulator and its preliminary estimated evaluation showed the elapsed time with my proposed task parallel way is 31.9 % of the previous sequential way and is only 9.94 % of the naive whole repeating way.

5

Cost Optimization to Cloud Computing

Research on agent-based simulation has been essential for areas such as, transportation, environmental protection, prediction of global economic evolution [68]. 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 [29, 85], 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 [86] and/or increased communication between computing nodes.

In this chapter, I 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. My 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. I also propose a novel solution for efficient migration of workload between computing nodes. My 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. My 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.

5.1 IBM Mega Traffic Simulator

In this study, IBM Mega Traffic Simulator (*Megaffic*) is used, which is a large-scale distributed agent-based traffic simulator developed by IBM [65, 66, 88]. 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 my case). XAXIS is based on *X10* [26], 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.

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. [17] have defined dSUMO, a distributed version of the free and open traffic simulation suite SUMO [14]. SUMO offers tools and algorithms for developers of transportation scenarios and helps them to import road networks (e.g., extracted from OpenStreeMap data [37]), 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 [1], 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 [79].

5.2 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 [3]) 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.

5.2.1 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 [2], Google Compute Engine [4] and Rackspace [8] 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.

5.2.2 Supercomputers with Flexible Cost Models

Recently, supercomputer have entered the pay-as-you-go model that has been popular in the Cloud.

Tsubame 2.5 [10] 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 [9]. 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 [5] 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 [9]. It is produced by Fujitsu and operated by RIKEN Advanced Institute for Computational Science (AICS). The billing interval is an hour.

5.3 Adaptive VM Provisioning System for Large-Scale Agent-based Traffic Simulation

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

5.3.1 General Description

My system shares many of its concerns with other large-scale computing systems on commodity machines (e.g., the Cloud or supercomputers) and I do not pretend I built it from scratch.

However, my solution systematises various approaches and methods and proposes to organise them in a coherent set of modules, addressing what I 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 5.3.1 and 5.3.2 give a more detailed description of how I want to achieve my objective of adaptive resource provisioning for large-scale distributed traffic simulation. Figure 5.3.1 shows the fundamental elements of my 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. My framework is constructed as a extension of the master in the existing simulator.

Figure 5.3.2 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 I give a detailed overview of the three main modules in my solution.

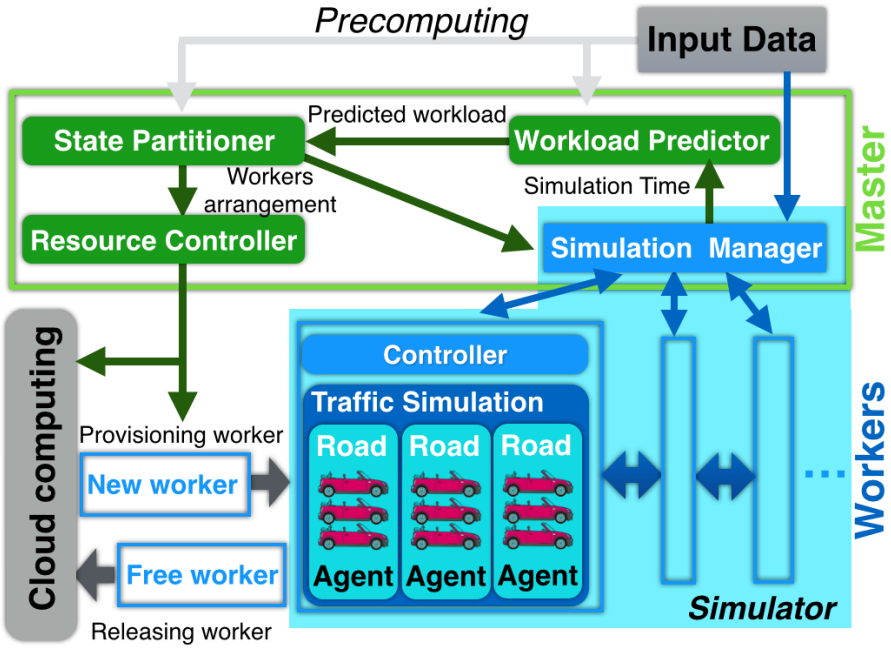

Figure 5.3.1: System Overview.

5.3.2 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 I assume followings:

- 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.

5.3.3 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

```
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 }
```

Figure 5.3.2: Computation Flow in Master.

attracted a lot of research in the area (see for instance [94]). Recently, some flexible partitioning has been proposed by Bragard et al. [18, 19] and they claim it is applicable to traffic simulation. I believe it would be possible to use such adaptive partitioning in my own system and I actually consider it for some future work.

In the current version of my system, I assume the following property about the input data (see Section 5.5 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 I use a randomization).

- I run the simulation many times with very minor changes, which does not impact the total performance. Thus, I can totally find the execution time from the second time. For example, I 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, I can define the appropriate number of workers at each iteration in following ways. In advance, I 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.

5.3.4 Resource Controller

The resource controller controls physical or virtual machines, depending on the environment. What I 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.

5.4 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. I first detail the synchronisation mechanisms in Megaffic. Then I discuss some naive, synchronous, solution which does not destroy consistency of the simulating result. After that, I 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.

5.4.1 Synchronisation and Migration in Megaffic

As I said above, my system is based on a Java implementation of Megaffic (see also Section 5.5 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 5.4.1 for an illustration of the three steps mentioned above: individual tasks (computation of vehicles' position), communication and synchronisation barrier.

Beside, in order to change the number of workers during the simulation while keeping consistency of the simulating result, I 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:

$$C_{total} = C_r \times Nr + \sum_{i=0,1,2,,..Nr} \{C_a N_a(i)\} \tag{5.1}$$

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:

$$T_{paraTotal} = \max_{i \in \{0,1,2,\cdots,N_r\}} \{Time(C_r + N_a(i) \times C_a), \} \tag{5.2}$$

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

5.4.2 Basic Ideas for a Migration Process in Megaffic

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

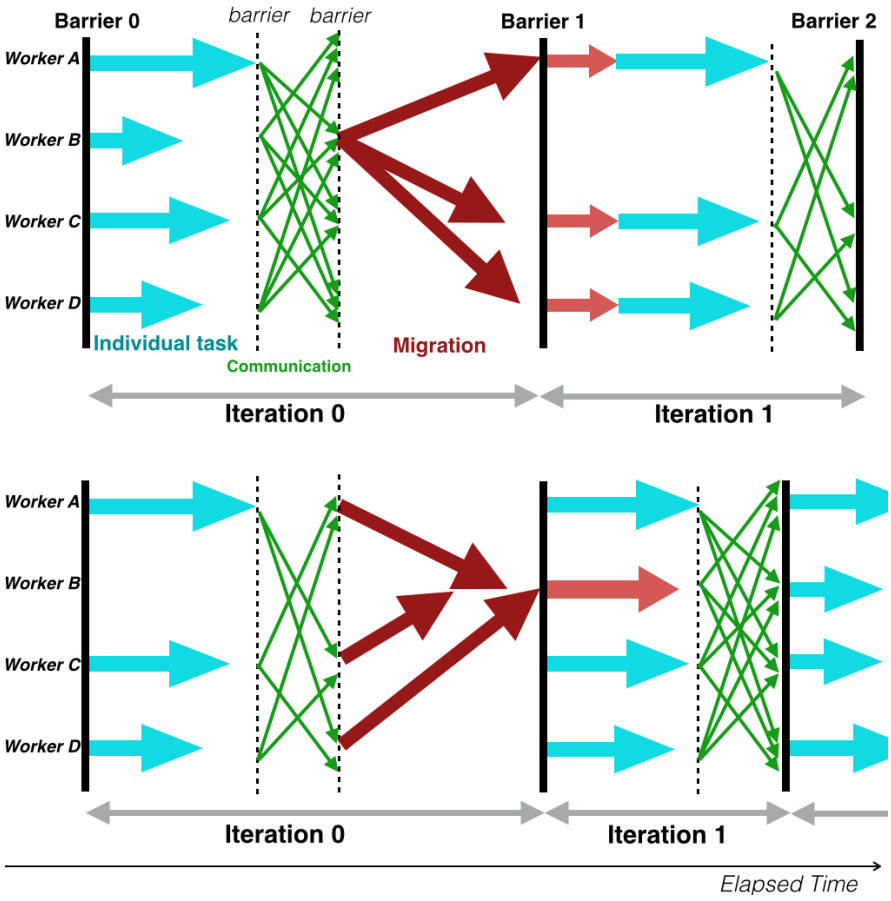

Figure 5.4.1: Naive Synchronous Migration.

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 I 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 my 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 5.4.1 shows

a basic example of such a migration process, that I 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).

5.4.3 Basic and Staged Asynchronous Migration

To reduce the migration overhead that I can observe in Figure 5.4.1 I propose two *asynchronous migration* schemes that overlap migration and individual tasks' processing, hence limiting the waiting time of the different workers.

A basic asynchronous migration (see Figure 5.4.2) (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. Figure5.4.3 shows the example of a staged asynchronous migration divided in 3 iterations.

5.5 Evaluation of Adaptive VM Provisioning

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.

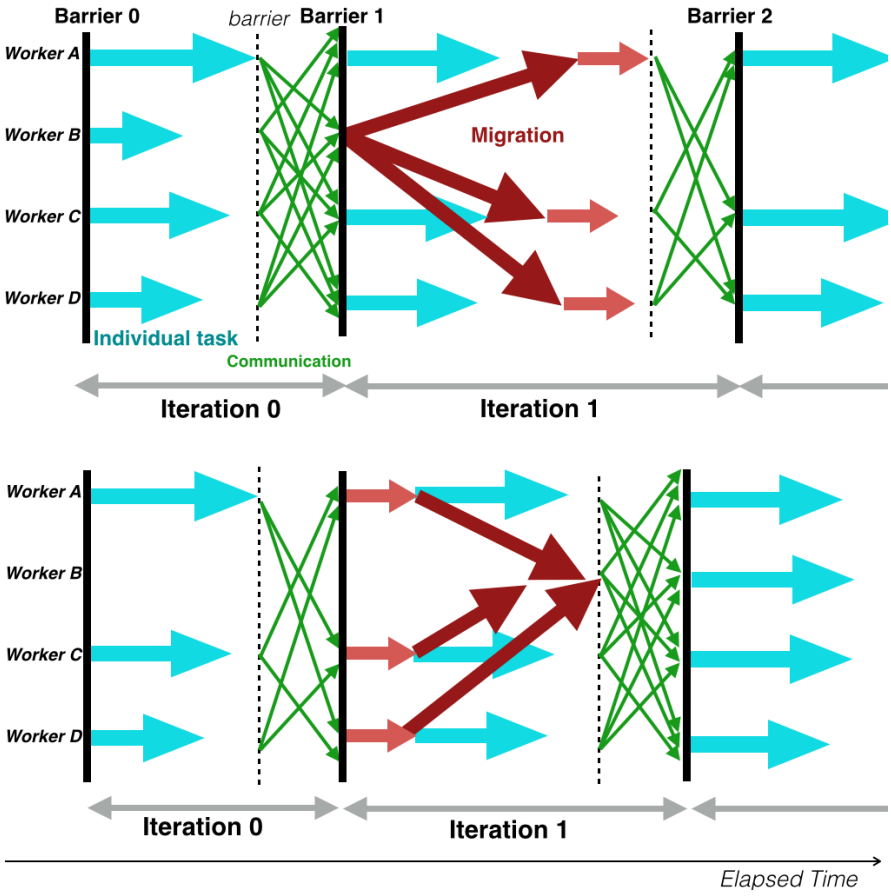

Figure 5.4.2: Asynchronous Migration.

- 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.

5.5.1 System Setups

My 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; I then decided to use Messagepack (version 0.6.6), an efficient binary serialization format. Moreover, as serialisation in my system occurs during the migration (of states between workers) and the communication (of vehicles between workers), I need an efficient communication library, to make sure the performance of the simulation is not impacted. I then use Netty (version 3.2), an NIO communication frame-

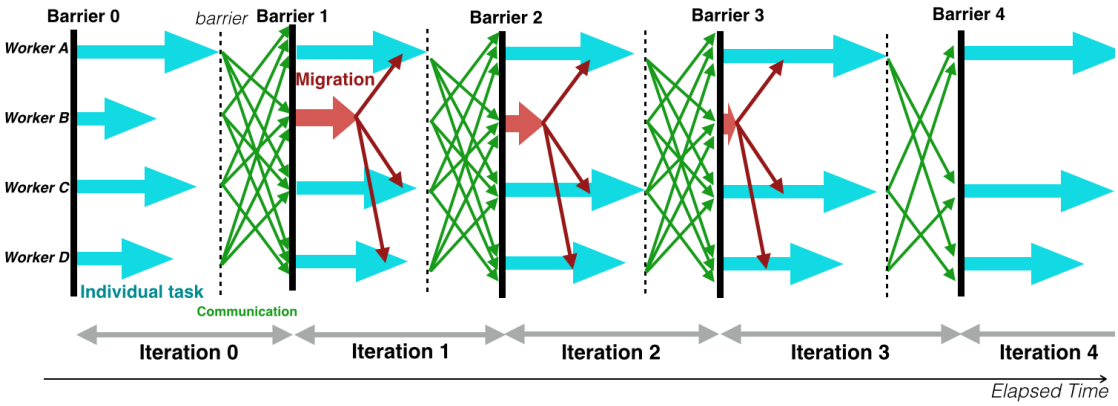

Figure 5.4.3: Staged Asynchronous Migration.

work that has good throughput and low latency.

I use Google Compute Engine as my Infrastructure as a Service solution (see Table 5.5.1), 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 5.5.1: Virtual machines' configurations.

Table 5.5.2 shows the simulation setups of my 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 5.5.1 shows the typical number of departing vehicles per 10 seconds of real time in Tokyo (it corresponds to the workload for the simulation). I use the full road network of Tokyo,

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 5.5.2: Scenario's input data.

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). I generate it randomly, as well as the exact route of every 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.

5.5.2 Evaluation of the Optimal Number of Workers

Here I 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. I 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 5.5.1). The results of this evaluation are given in Figure 5.5.2, 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 I have so I stick to this value of 16. From that study I extract the optimal values of number of workers per load as given in Table 5.5.3.

5.5.3 Performance Evaluation of the System

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

Figure 5.5.3 shows the average elapsed time of one iteration (including staged asynchronous

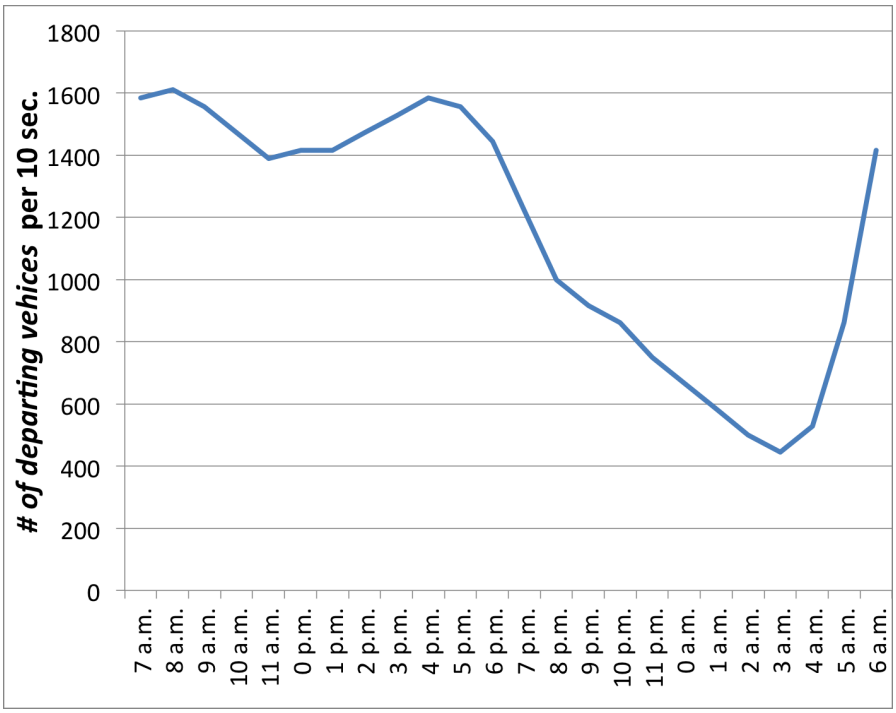

Figure 5.5.1: Number of Departing Vehicles in a Typical 24 hour Scenario of Tokyo.

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 5.4. 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".

Figure 5.5.4 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 I focus to the 8 p.m. to 6 a.m scenario, where the number of departing vehicles changes a lot, the

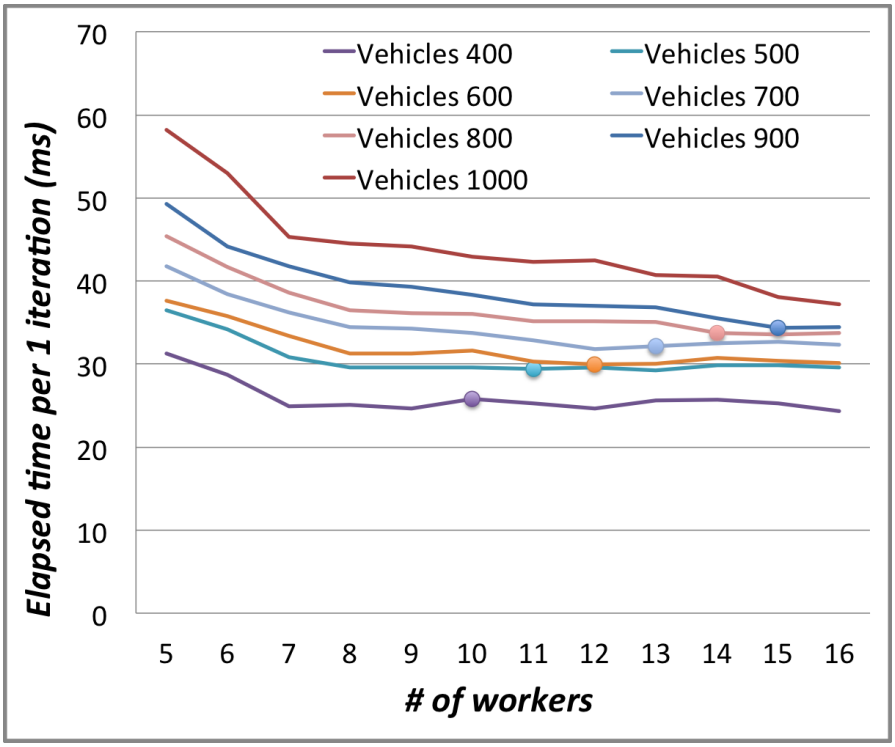

Figure 5.5.2: Strong Scaling Evaluation and Saturation Points for Each Rate of Departing Vehicles.

reduction is even bigger (23%) compared to the non elastic execution with 16 workers.

5.6 Summary

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

# 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 5.5.3: # of Workers Suggested Depending on the Load.

Figure 5.5.3: Time of One Iteration Including (Staged) Migration.

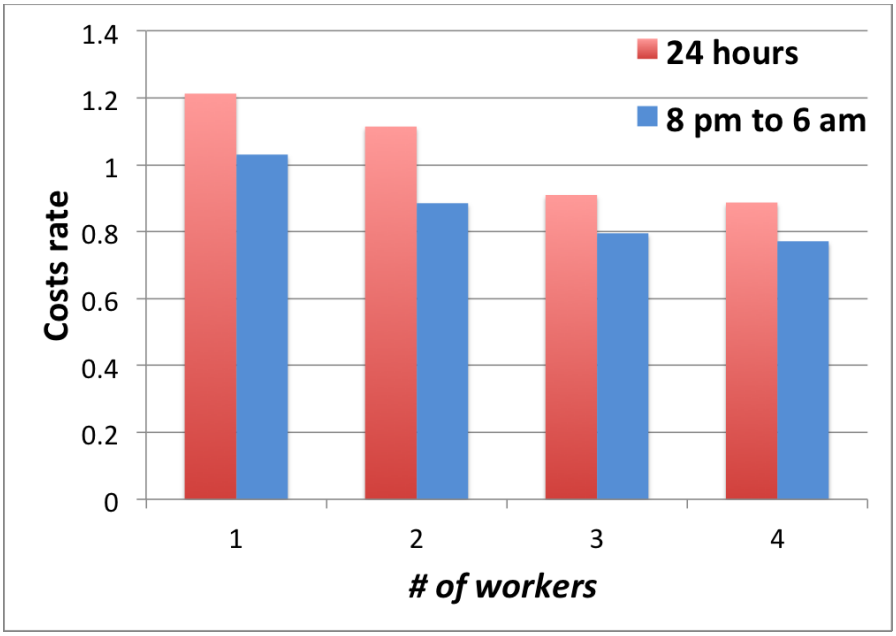

Figure 5.5.4: Cost Reduction.

6

Conclusion

In this thesis, I presented new techniques for large-scale systems' simulation with parallel and distributed discrete event simulation. I showed mainly 3 contributions in application layer, simulation engine layer and computing environment layer, respectively.

In Chapter 2, I discussed the modeling way of emerging large-scale P2P systems. My proposal includes a partition technique and performance optimizations as well as how I represent P2P systems on PDES. For the partition, I showed statically that application level partition with application ID (e.g. hash value in Chord) is better than underlay address (e.g. IP address). For the optimization, I showed that the incremental state saving and the lazy cancellation reduces over 90 % overhead, including memory usage and # of anti-messages. Finally, I illustrate the strong scalability of the parallel execution using up to 48 cores with various P2P system situations.

In Chapter 3 and Chapter 4, I illustrated the new techniques for analysis in simulation flow. I proposed *exact-differential simulation*, which is the redundancy reduction techniques in repeating simulation with minor change from initial execution. First, I showed the concept and algorithm of exact-differential simulation in Chapter 3, where I also showed the performance

impact in traffic simulation. In the experiment of Tokyo traffic, the exact-differential simulation shows 7.26 times as much elapsed time improvement in average and 2.26 times improvement even in the worst case as the whole simulation. After that, I discussed more sophisticated solutions for repeating execution with the exact-differential simulation in Chapter 4. One solution is the what-if scenario filtering, enabling to pick up meaningful what-if scenarios and reducing the number of what-if scenarios, which directly decreases total execution time of repeating. The other solution is the exact-differential cloning, enabling to execute exact-differential simulation tasks in parallel to improve its total execution time. In preliminary evaluation in Tokyo traffic simulation, I showed potential of the solutions by estimating how the what-if scenarios filtering reduces the number of meaningless scenarios and also by estimating a performance improvement from my previous works with the exact-differential cloning.

In Chapter 5, I discussed the efficient usage of the cloud computing environment in PDES, including mainly two contributions: (i) 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. The experimental results on PDES of traffic simulation in the city of Tokyo during 24 hours show that my dynamic provisioning method outperforms a naive static provisioning by 12% in average and 23% during periods when workload changes a lot.

Bibliography

[1] MATSim: multi-agent transport simulation. http://www.matsim.org/ (Last access: 30 Oct. 2015).

[2] Microsft Asure. http://azure.microsoft.com/ (Last access: 30 Oct. 2015).

[3] Amazon EC2. https://aws.amazon.com/ec2/ (Last access: 30 Oct. 2015).

[4] Google Compute Engine. https://cloud.google.com/compute/ (Last access: 30 Oct. 2015).

[5] K computer. http://www.kcomputer.jp/en/ (Last access: 30 Oct. 2015).

[6] The network simulator – ns–2, . http://www.isi.edu/nsnam/ns/ (Last access: 30 Oct. 2015).

[7] The network simulator – ns–3, . https://www.nsnam.org (Last access: 30 Oct. 2015).

[8] Rackspace Cloud. http://www.rackspace.com/ (Last access: 30 Oct. 2015).

[9] TOP500. http://www.top500.org/ (Last access: 30 Oct. 2015).

[10] Tsubame 2.5. http://www.gsic.titech.ac.jp/en/tsubame/ (Last access: 30 Oct. 2015).

[11] P.D. Barnes, Jr., C.D. Carothers, D.R. Jefferson, and J.M. LaPre. Warp Speed: Executing time warp on 1,966,080 cores. In *Proceedings of the 1st ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS'13, pages 327–336. ACM, 2013.

[12] D.W. Bauer Jr., C.D. Carothers, and A. Holder. Scalable time warp on Blue Gene supercomputers. In *Proceedings of the 23rd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, PADS'09, pages 35–44. IEEE, 2009.

[13] I. Baumgart, B. Heep, and S. Krause. OverSim: A flexible overlay network simulation framework. In *Proceedings of the IEEE Global Internet Symposium 2007*, GI'07, pages 79–84. IEEE, 2007.

[14] M. Behrisch, L. Bieker, J. Erdmann, and D. Krajzewicz. SUMO - Simulation of Urban MObility: an overview. In *Proceedings of the 3rd International Conference on Advances in System Simulation*, SIMUL'11, pages 63–68. IARIA, 2011.

[15] L. Bieker, D. Krajzewicz, A. Morra, C. Michelacci, and F. Cartolano. Traffic simulation for all: a real world traffic scenario from the city of bologna. In *Modeling Mobility with Open Data*, pages 47–60. Springer, 2015.

[16] BitTorrent, Inc. BitTorrent and μTorrent software surpass 150 million user milestone; announce new consumer electronics partnerships, 2012. http://www.bittorrent.com/intl/es/company/about/ces_2012_150m_users/ (Last access: 30 Oct. 2015).

[17] Q. Bragard, A. Ventresque, and L. Murphy. dSUMO: towards a distributed SUMO. In *Proceedings of the 1st SUMO User Conference*, 2013.

[18] Q. Bragard, A. Ventresque, and L. Murphy. Synchronisation for Dynamic Load Balancing of Decentralised Conservative Distributed Simulation. In *Proceedings of the 2nd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS'14, pages 117 – 126. ACM, 2014.

[19] Q. Bragard, A. Ventresque, and L. Murphy. Global Dynamic Load-Balancing for Decentralised Distributed Simulation. In *Proceeding of the 2006 Winter Simulation Conference*, WSC'14, pages 3797 – 3808. IEEE, 2014.

[20] S.A. Brueckner and H. Van Dyke Parunak. Resource-aware exploration of the emergent dynamics of simulated systems. In *Proceedings of the 2nd international joint conference on Autonomous agents and multiagent systems*, AAMAS'03, pages 781–788. ACM, 2003.

[21] E. Cabrera, E. Luque, M. Taboada, F. Epelde, and L. Ma Iglesias. Abms optimization for emergency departments. In *Proceedings of the 2012 Winter Simulation Conference*, WSC'12, pages 1–12. IEEE, 2012.

[22] B. Calvez and G. Hutzler. Automatic tuning of agent-based models using genetic algorithms. In *Proceedings of the 6th international conference on Multi-Agent-Based Simulation*, MABS'05, pages 41–57. Springer, 2005.

[23] C.D. Carothers, D. Bauer, and S. Pearce. ROSS: A high-performance, low-memory, modular Time Warp system. *Elsevier Journal of Parallel and Distributed Computing*, 62(11): 1648–1669, 2002.

[24] H. Casanova, A. Legrand, and M. Quinson. SimGrid: A generic framework for large-scale distributed experiments. In *Proceedings of the 10th International Conference on Computer Modeling and Simulation*, UKSIM'08, pages 126–131. IEEE, 2008.

[25] S. Chandrasekaran and M.D. Hill. Optimistic simulation of parallel architectures using program executables. In *Proceedings of the 10th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation*, PADS'96, pages 143–150. IEEE, 1996.

[26] 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.

[27] D. Chen, S.J. Turner, W. Cai, B.P. Gan, and M.Y.H. Low. Algorithms for HLA-based distributed simulation cloning. *ACM Transactions on Modeling and Computer Simulation*, 15 (4):316–345, 2005.

[28] L. Codeca, R. Frank, and T. Engel. LuST: a 24-hour scenario of luxembourg city for SUMO traffic simulations. In *Proceedings of the 3rd SUMO User Conference*, 2015.

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

[30] T. T. A. Dinh, M. Lees, G. Theodoropoulos, and R. Minson. Large scale distributed simulation of P2P networks. In *Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, PDP'08, pages 499–507. IEEE, 2008.

[31] C. Engelmann and F. Lauer. Facilitating co-design for extreme-scale systems through lightweight simulation. In *Proceeding of the 2010 IEEE International Conference on Cluster Computing: 1st Workshop on Application/Architecture Co-design for Extreme-scale Computing*, AACEC'10, pages 1–8. IEEE, 2010.

[32] C. Engelmann and T. Naughton. Improving the performance of the extreme-scale simulator. In *Proceedings of the IEEE/ACM 18th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT'14, pages 198–207. IEEE, 2014.

[33] S.L. Ferenci, R.M. Fujimoto, M.H. Ammar, K.S. Perumalla, and G.F. Riley. Updateable simulation of communication networks. In *Proceedings of the 16th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation*, PADS'02, pages 107–114. IEEE, 2002.

[34] R.M. Fujimoto. *Parallel and distributed simulation systems*. Wiley New York, 2000.

[35] P. García, C. Pairot, R. Mondéjar, J. Pujol, H. Tejedor, and R. Rallo. PlanetSim: A new overlay network simulation framework. In *LNCS 3437 for SEM 2004*, pages 123–136. Springer, 2005.

[36] D. Goldsman, R.E. Nance, and J.R. Wilson. A brief history of simulation. In *Proceedings of the 2009 Winter Simulation Conference*, WSC '09, pages 310–313. Winter Simulation Conference, 2009.

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

[38] M. Hanai. ScaleSim: Large-scale parallel/distributed discrete event simulator. http://masahanai.jp/software/#scalesim (Last access: 31 Oct. 2015).

[39] M. Hanai and K. Shudo. Optimistic parallel simulation of very large-scale peer-to-peer systems. In *Proceedings of the IEEE/ACM 18th International Symposium on the Distributed Simulation and Real Time Applications*, DS-RT'14, pages 35–42. IEEE, 2014.

[40] M. Hanai, T. Suzumura, A. Ventresque, and K. Shudo. An adaptive vm provisioning method for large-scale agent-based traffic simulations on the cloud. In *Proceedings of the IEEE 6th International Conference on Cloud Computing Technology and Science*, CloudCom'14, pages 130–137. IEEE, 2014.

[41] M. Hanai, T. Suzumura, G. Theodoropoulos, and K.S. Perumalla. Exact-differential large-scale traffic simulation. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS'15, pages 271–280. ACM, 2015.

[42] M. Hanai, T. Suzumura, G. Theodoropoulos, and K.S. Perumalla. Towards large-scale what-if traffic simulation with exact-differential simulation. In *Proceedings of the 2015 Winter Simulation Conference*, WSC'15. IEEE, 2015.

[43] K. Harrigan and G. Riley. Simulation speedup of ns-3 using checkpoint and restore. In *Proceedings of the 2014 Workshop on Ns-3*, WNS3 '14, pages 7:1–7:7. ACM, 2014.

[44] Y. Hou, Y. Zhao, K.F. Hulme, S. Huang, Y. Yang, W.A. Sadek, and C. Qiao. An integrated traffic-driving simulation framework: Design, implementation, and validation. *Elsevier Journal of Transportation Research Part C: Emerging Technologies*, 45:138 – 153, 2014.

[45] M. Hybinette. Just-in-time cloning. In *Proceedings of the 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, PADS'04, pages 45–51. IEEE, 2004.

[46] M. Hybinette and R.M. Fujimoto. Cloning parallel simulations. *ACM Transactions on Modeling and Computer Simulation*, 11(4):378–407, 2001.

[47] M. Hybinette and R.M. Fujimoto. Scalability of parallel simulation cloning. In *Proceedings of the 35th Annual Simulation Symposium*, SS'02, pages 275–282. IEEE, 2002.

[48] J.S. Ivey, B.P. Swenson, and G.F. Riley. PHOLD performance of conservative synchronization methods for distributed simulation in ns-3. In *Proceedings of the 2015 Workshop on Ns-3*, WNS3 '15, pages 47–53. ACM, 2015.

[49] D.R. Jefferson. Virtual time. *ACM Transaction Programming Languages and Systems*, 7(3): 404–425, 1985.

[50] V. Jha and R.L. Bagrodia. A unified framework for conservative and optimistic distributed simulation. In *Proceedings of the ACM/IEEE/SCS 8th Workshop on Parallel and Distributed Simulation*, PADS'94, pages 12–19. ACM, 1994.

[51] L. Jie, V.H. Zuylen, Y. Chen, F. Viti, and I. Wilmink. Calibration of a microscopic simulation model for emission calculation. *Elsevier Journal of Transportation Research Part C: Emerging Technologies*, 31:172 – 184, 2013.

[52] G. Karypis and V. Kumar. METIS - a software package for partitioning unstructured graphs, meshes, and computing fill-reducing orderings of sparse matrices-version 5.1.0. http://glaros.dtc.umn.edu/gkhome/metis/metis/overview (Last access: 30 Oct. 2015).

[53] G. Karypis and V. Kumar. Multilevel *k*-way partitioning scheme for irregular graph. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[54] Oak Ridge National Laboratory. Summit, . https://www.olcf.ornl.gov/summit/ (Last access: 30 Oct. 2015).

[55] Oak Ridge National Laboratory. Titan, . https://www.olcf.ornl.gov/titan/ (Last access: 30 Oct. 2015).

[56] S. Lawe, J. Lobb, A. Sadek, S. Huang, and C. Xie. TRANSIMS implementation in chittenden county, vermont. *Transportation Research Record: Journal of the Transportation Research Board*, 2132:113–121, 2009.

[57] Y-B Lin and Edward D. Lazowska. Exploiting lookahead in parallel simulation. *IEEE Transaction on Parallel and Distributed Systems*, 12(4):457–469, 1990.

[58] N. Liu, A. Haider, X. Sun, and D. Jin. FatTreeSim: Modeling large-scale fat-tree networks for HPC systems and data centers using parallel and discrete event simulation. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, PADS'15, pages 199–210. ACM, 2015.

[59] D.E. Martin, T.J. McBrayer, and P.A. Wilsey. WARPED: a time warp simulation kernel for analysis and application development. In *Proceedings of the 29th Hawaii International Conference on System Sciences*, volume 1, pages 383–386. IEEE, 1996.

[60] K. Meister, Balmer M., Ciari F., Horni A., Rieser M., Waraich R.A., and Axhausen K.W. Large-scale agent-based travel demand optimization applied to switzerland, including mode choice. In *Working paper in Institute for Transport Planning and Systems*. ETH, 2010.

[61] A. Montresor and M. Jelasity. PeerSim: A scalable P2P simulator. In *Proceedings of the IEEE 9th International Conference on Peer-to-Peer Computing*, P2P'09, pages 99–100. IEEE, 2009.

[62] S. Nikolaev, P.D. Barnes, Jr., J.M. Brase, T.W. Canales, D.R. Jefferson, S. Smith, R.A. Soltz, and P.J. Scheibel. Performance of distributed ns-3 network simulator. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, SimuTools '13, pages 17–23. ICST, 2013.

[63] S. Nikolaev, E. Banks, P.D. Barnes, Jr., D.R. Jefferson, and S. Smith. Pushing the envelope in distributed ns-3 simulations: One billion nodes. In *Proceedings of the 2015 Workshop on Ns-3*, WNS3 '15, pages 67–74. ACM, 2015.

[64] L. Ning and C.D. Carothers. Modeling billion-node torus networks using massively parallel discrete-event simulation. In *Proceedings of 2011 IEEE Workshop on Principles of Advanced and Distributed Simulation*, PADS'11, pages 1–8. IEEE, 2011.

[65] 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.

[66] 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, 2013.

[67] K. Ozbay, H. Yang, E.F. Morgul, S. Mudigonda, and B. Bartin. Big data and the calibration and validation of traffic simulation models. *Transportation Research Record: Journal of the Transportation Research Board*, page 92, 2014.

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

[69] B. Park and J. Kwak. Calibration and validation of TRANSIMS microsimulator for an urban arterial network. *KSCE Journal of Civil Engineering*, 15(6):1091–1100, 2011.

[70] J. Pelkey and G. Riley. Distributed simulation with MPI in ns-3. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques*, SIMUTools '11, pages 410–414. ICST, 2011.

[71] K.S. Perumalla. μsik - a micro-kernel for parallel/distributed simulation systems. PADS'05, pages 59–68. IEEE, 2005.

[72] K.S. Perumalla. A systems approach to scalable transportation network modeling. In *Proceedings of the 2006 Winter Simulation Conference*, WSC'06, pages 1500–1507. IEEE, 2006.

[73] K.S. Perumalla. $\mu\pi$: A scalable and transparent system for simulating MPI programs. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools'10, pages 62:1–62:6. ICST, 2010.

[74] K.S. Perumalla and A.J. Park. Simulating billion-task parallel programs. In *Proceedings of the 2014 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, SPECTS'14, 2014.

[75] K.S. Perumalla, A.J. Park, and V. Tipparaju. GVT algorithms and discrete event dynamics on 129k+ processor cores. In *Proceedings of the 2011 18th International Conference on High Performance Computing*, HIPC'11, pages 1–11. IEEE, 2011.

[76] K.S. Perumalla, A.J. Park, and V. Tipparaju. Discrete event execution with one-sided and two-sided GVT algorithms on 216,000 processor cores. *ACM Transactions on Modeling and Computer Simulation*, 24(3):16:1–16:25, 2014.

[77] S. Pllana, I. Brandic, and S. Benkner. Performance modeling and prediction of parallel and distributed computing systems: A survey of the state of the art. In *Proceeding of the 1st International Conference on Complex, Intelligent and Software Intensive Systems*, CISIS'07, pages 279–284. IEEE, 2007.

[78] M. Quinson, C. Rosa, and C. Thiéry. Parallel simulation of peer-to-peer systems. In *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGrid'12, pages 668–675. IEEE, 2012.

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

[80] C. Renard, K.and Peri and J. Clarke. A performance and scalability evaluation of the ns-3 distributed scheduler. In *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*, SIMUTOOLS '12, pages 378–382. ICST, 2012.

[81] R. Rönngren, M. Liljenstam, R. Ayani, and J. Montagnat. Transparent incremental state saving in Time Warp parallel discrete event simulation. In *Proceedings of the 10th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation*, PADS'96, pages 70–77. IEEE, 1996.

[82] K. Shudo, Y. Tanaka, and S. Sekiguchi. Overlay Weaver: An overlay construction toolkit. *Elsevier Journal of Computer Communications*, 31(2):402–412, 2008.

[83] M. Smith, A. Sadek, and S. Huang. Large-scale microscopic simulation: Toward an increased resolution of transportation models. *ASCE Journal of Transportation Engineering*, 134(7):273 – 281, 2008.

[84] S. Smith, D.R. Jefferson, P.D. Barnes, Jr., and S. Nikolaev. Improving per processor memory use of ns-3 to enable large scale simulations. In *Proceedings of the 2015 Workshop on Ns-3*, WNS3 '15, pages 60–66. ACM, 2015.

[85] T. Suzumura and H. Kanezashi. Highly scalable x10-based agent simulation platform and its application to large-scale traffic simulation. In *Proceedings of the 2012 IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT'12, pages 243–250. IEEE, 2012.

[86] T. Suzumura and H. Kanezashi. Accelerating large-scale distributed traffic simulation with adaptive synchronization method. In *Proceedings of the 20th ITS World Congress*. ITS Japan, 2013. Paper No.4083.

[87] T. Suzumura and H. Kanezashi. Multi-modal traffic simulation platform on parallel and distributed systems. In *Proceedings of the 2014 Winter Simulation Conference*, WSC'14, pages 769–780. IEEE, 2014.

[88] 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 *Proceedings of the 2012 ACM SIGPLAN X10 Workshop*, X10'12, pages 3:1–3:4. ACM, 2012.

[89] T. Suzumura, C. Houngkaew, and H. Kanezashi. Towards billion-scale social simulations. In *Proceedings of the 2014 Winter Simulation Conference*, WSC'14, pages 781–792. IEEE, 2014.

[90] B.P. Swenson, J.S. Ivey, and G.F. Riley. Performance of conservative synchronization methods for complex interconnected campus networks in ns-3. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, pages 3096–3106. IEEE, 2014.

[91] Decentralized Systems and Network Services Research Group in Karlsruhe Institute of Technology. Live monitoring of the BitTorrent DHT. http://dsn.tm.uni-karlsruhe.de/english/2936.php (Last access: 30 Oct. 2015).

[92] High Performance University of Rome and Dependable Computing Systems Research Group. ROOT-SIM (The ROme OpTimistic Simulator). https://github.com/HPDCS/ROOT-Sim (Last access: 30 Oct. 2015).

[93] A. Varga. The OMNeT++ discrete event simulation system. In *Proceedings of the European simulation multiconference*, ESM'01, page 185, 2001.

[94] 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 *Proceedings of the IEEE/ACM 16th International Symposium on Distributed Simulation and Real Time Applications*, DS-RT'12, pages 202 – 209. IEEE, 2012.

[95] D. West and K. Panesar. Automatic incremental state saving. In *Proceedings of the 10th ACM/IEEE/SCS Workshop on Parallel and Distributed Simulation*, PADS'96, pages 78–85. IEEE, 1996.

[96] P.A. Wilsey. WARPED. http://eecs.ceas.uc.edu/~paw/research/warped/ (Last access: 30 Oct. 2015).

[97] S.B. Yoginath and K.S. Perumalla. Parallel vehicular traffic simulation using reverse computation-based optimistic execution. In *Proceedings of the 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation*, PADS'08, pages 33–42. IEEE, 2008.

[98] G. Zhang, M. Fang, M. Qian, and S. Xu. Parallel cloning simulation of flood mitigation operations in the upper-middle reach of huaihe river. In *Proceedings of the 2012 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, CyberC'12, pages 73–80. IEEE, 2012.

[99] Y. Zhao and W.A. Sadek. Large-scale agent-based traffic micro-simulation: Experiences with model refinement, calibration, validation and application. *Elsevier Journal of Procedia Computer Science*, 10:815 – 820, 2012.