Exact-Differential Simulation: Differential Processing of Large-Scale Discrete Event Simulations

MASATOSHI HANAI, Nanyang Technological University, Singapore
TOYOTARO SUZUMURA, IBM T.J. Watson Research Center, USA
ELVIS S. LIU, Southern University of Science and Technology, China
GEORGIOS THEODOROPOULOS, Southern University of Science and Technology, China
KALYAN S. PERUMALLA, Oak Ridge National Laboratory, USA

Using computer simulation to analyze large-scale discrete event systems requires repeated executions with various scenarios or parameters. Such repeated executions can induce significant redundancy in event processing when the modification from a prior scenario to a new scenario is relatively minor, and when the altered scenario influences only a small part of the simulation. For example, in a city-scale traffic simulation, an altered scenario of blocking one junction may only affect a small part of the city for considerable length of time. However, traditional simulation approaches would still repeat the simulation for the whole city even when the changes are minor. In this paper, we propose a new redundancy reduction technique for large-scale discrete event simulations, called exact-differential simulation, which simulates only the altered portions of scenarios and their influences in repeated executions while still achieving the same results as the re-execution of entire simulations. The paper presents the main concepts of the exact-differential simulation, the design of its algorithm, and an approach to build an exact-differential simulation middleware that supports multiple applications of discrete event simulation. We also evaluate our approach by using two case studies, PHOLD benchmark and a traffic simulation of Tokyo.

CCS Concepts: • **Computing methodologies** → **Modeling and simulation**; **Discrete-event simulation**; *Massively parallel and high-performance simulations.*

Additional Key Words and Phrases: Differential processing; incremental processing; what-if simulation; redundancy reduction

ACM Reference Format:
Masatoshi Hanai, Toyotaro Suzumura, Elvis S. Liu, Georgios Theodoropoulos, and Kalyan S. Perumalla. 2010. Exact-Differential Simulation: Differential Processing of Large-Scale Discrete Event Simulations. *ACM Trans. Model. Comput. Simul.* 0, 0, Article 0 (2010), 25 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Parallel Discrete Event Simulation (PDES) is a beneficial method for fast and scalable execution of large-scale discrete event simulations such as city-/country-scale traffic simulation or Internet-scale network simulation. PDES is able to deal with large numbers of simulated entities executed in a synchronized manner across multiple processors and multiple computers. Recent studies have

Authors' addresses: Masatoshi Hanai, Nanyang Technological University, Singapore, mhanai@acm.org; Toyotaro Suzumura, IBM T.J. Watson Research Center, USA, tsuzumura@acm.org; Elvis S. Liu, Southern University of Science and Technology, China, esyliu@sustc.edu.cn; Georgios Theodoropoulos, Southern University of Science and Technology, China, georgios@sustc.edu.cn; Kalyan S. Perumalla, Oak Ridge National Laboratory, USA, perumallaks@ornl.gov.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2010 Copyright held by the owner/author(s). Publication rights licensed to ACM.
XXXX-XXXX/2010/0-ART0 $15.00
https://doi.org/0000001.0000001

shown the outstanding scalability of PDES, where over one hundred million states and over 30 trillion events can be handled on systems with up to 2 million CPU cores [1, 2, 5].

Since large-scale systems consist of numerous events and states, the use of PDES to analyze them requires a large number of repeated simulations based on various what-if scenarios and parameter patterns. For example, the Tokyo traffic simulation [15, 25, 26, 29] made 770,000 (770K) repeated simulations to analyze what happens if one of the roads was blocked, since there are 770K junctions and thus generated the same number of blocking scenarios. If pairs of blocks of the junctions are simulated, the total number of simulations exceeds 296 billion $\binom{770K}{2}$. If scenarios become more complex, the number of simulations would increase exponentially and reach $2^{770K} = \sum_{i=1}^{770K} \binom{770K}{i}$.

However, repeating the entire simulation leads to a lot of redundancy in event processing especially when the altered scenarios affect only a small part of the simulation and, as a result, most of the events and simulation results would be the same as the prior simulations. Furthermore, since it would be difficult to determine how much the altered scenarios affect the final results, therefore it is generally difficult to identify the altered parts before repeating the simulation. In fact, this problem can also be seen in the Tokyo traffic simulation—some road blocks affect a very limited part of Tokyo but others affect wide areas, and the level of influence cannot be determined before repeating the simulation.

In this paper, we propose *exact-differential simulation*, a differential processing technique for large-scale discrete event simulations. This technique only simulates the parts changed by the altered scenarios while keeping other parts of the simulation intact. The word "*exact*" implies that the final results are identical to the results obtained from traditional simulation of the altered scenario. The word "*differential*" implies that only events which differ from the base simulation, and subsequent causal effects are reprocessed. The main steps of the exact-differential simulation are as follows: (1) in the initial baseline simulation, the simulator processes the entire scenario and stores the logs of all states and events of the simulation, and (2) in a subsequent repeating simulation, the simulator only reprocesses altered events and their influences on other events by using the stored logs of the baseline simulation.

1.1 Challenges and Contributions

The challenges of the exact-differential simulation fall in two different areas: (1) algorithmic challenges and (2) software challenges.

(1) How can the impact of altered scenarios be determined and reprocessed? Which types of logs are required for these processes?
(2) How can the system be built to separate the application logic from the implementation? In other words, is it possible to construct the system as a middleware, where it does *not* need additional application code by the simulation user, in order to exploit the exact-differential mechanisms?

Addressing these challenges, the main contributions of the paper are as follows:

(1) It presents a mechanism of exact-differential simulation by using optimistic PDES.
(2) The potential performance improvement of the proposed mechanism as well as the impact of altered scenarios to the number of reprocessing events are analyzed statically.
(3) The architecture of a middleware of exact-differential simulation is presented, which facilitates exact-differential simulation without requiring any additional application-specific code. The middleware only reuses original application codes.
(4) The performance improvement of the exact-differential simulation is evaluated by using two different types of applications: a synthetic PDES benchmark (PHOLD) and a large-scale microscopic traffic simulation of Tokyo city.

Based on our previous work on differential processing for traffic simulations [13], we extend its applicability to general PDES. In addition, we present a new static analysis based on states' data structure of the simulation model, and an empirical analysis of the synthetic benchmark (PHOLD) using our simulator.

The rest of the paper is organized as follows. Section 2 briefly describes the background of PDES. Section 3 presents the principles in the design of the exact-differential simulation as well as a discussion on static analysis. Section 4 presents and discusses the implementation of the proposed system. Section 5 evaluates the proposed system by using two different applications. In Section 6, a review of the related work is given, before concluding the paper in Section 7.

2 PARALLEL DISCRETE EVENT SIMULATION

PDES is concerned with parallelizing a sequential timestamp-ordered discrete event simulation on multiple processors or multiple computers. In a PDES, a simulation is divided into multiple units of sequential event processing, called *logical process (LP)*; each LP includes a local clock, a local state, and a timestamp-ordered priority queue for future events. An LP locally processes its own events in timestamp order while advancing its clock time. Since the events may be communicated among LPs and their local clocks influence each other, performing time synchronization to obtain the same simulation results as sequential simulation is a key challenge of PDES, which has been studied extensively over the years. In particular, *optimistic* synchronization is a well-known approach, featuring rollback and restart mechanisms. PDES with optimistic synchronization can be referred to as optimistic PDES, which is described in the following subsection.

Optimistic Parallel Discrete Event Simulation

In optimistic PDES, isolated LPs execute their events speculatively and concurrently in timestamp order. When a newly generated event is assigned to an LP, it will be immediately sent to this destination LP and added to a timestamp-ordered priority queue. Such speculative process may lead to causality violations of the events' timestamp order. For example, an LP may sometimes receive new events with a timestamp earlier than its local clock time. To recover from the causality violation, the optimistic PDES employs a *rollback* mechanism, which requires every LP to keep a history of three types of intermediate data, including: (1) future events in the priority queue, (2) negative copies of the generated event messages after the process (referred to as *anti-messages*), and (3) data required to rollback state updates (referred to as *state data*), such as an entire state in each update (referred to as *copy state saving*) or the modified parts of the state (referred to as *incremental state saving*). By using these intermediate data, the causality violation can be recovered by using the following approach.

- If an LP receives a new event that has a timestamp earlier than its local clock time, then the LP corrects its local clock time and local state to the received event's timestamp (i.e. *rollback*). After that, the new event is inserted to the LP's priority queue. Since the new event may cause additional causality violations (i.e. different results in the subsequent event processing), stored anti-messages later than the corrected time are sent to their destinations to recover the causality violations. Finally, the LP restarts local processing.
- If an LP receives a new anti-message, it carries out a process based on the following two cases.
  - Case 1: The timestamp of the received anti-message is earlier than the LP's local clock time. In this case, the LP sets its local clock time and local state to the anti-message's timestamp (i.e., *rollback*). After that, stored anti-messages with a later timestamp are resent recursively.

– Case 2: The timestamp of the anti-message is later than the LP's local clock time. In this case, a stored future event, which is identical to the anti-messages, is deleted from its priority queue.

After that, the receiving LP restarts local processing again.

During the speculative process, the minimum local clock time, called *Global Virtual Time (GVT)*, is periodically shared among LPs to clean up useless data of the past, such as events, anti-messages, and states. Specifically, stored data with a timestamp earlier than the GVT are ensured to be no longer used by the rollback process and are removed from the LPs. Furthermore, if GVT reaches the end time of simulation, all LPs are ensured that they can no longer rollback before the end time. Hence, the simulation can be terminated.

Optimistic synchronization has been studied extensively since the publication of its first paper [19]. Moreover, the book "Parallel and Distributed Simulation Systems" [11] provides a comprehensive summary of PDES and its synchronization techniques.

3 EXACT-DIFFERENTIAL SIMULATION

Exact-differential simulation is a differential processing method for timestamp-ordered discrete event simulation based on optimistic synchronization. It consists of two main steps: *initial baseline simulation* and one or more *repeating simulation(s)*. In the initial baseline simulation, all future events, anti-messages and state data are kept in the simulator; while in the repeating simulation, the simulator reprocesses altered scenarios and their influences, which are detected by using the mechanism of causality violation recovering in optimistic PDES. Generally, the initial baseline simulation is executed once, followed by multiple repeating simulations using various what-if scenarios and/or parameters.

3.1 Initial Baseline Simulation

In the initial baseline simulation, scenarios are processed at the beginning in the same way as a typical optimistic PDES. The future events, anti-messages, and state data are kept in the system instead of being removed during the GVT calculation; therefore, they are ensured to be no longer used for rollback. The LPs speculatively process their events in parallel. If causality violation occurs, rollback is carried out to recover it as described in Section 2. In GVT calculation, unlike the optimistic PDES where historical events, anti-messages, and state data older than GVT are released from memory space, the historical data are kept on each LP in timestamp order and will be reused in repeating simulations.

3.2 Repeating Simulation

In the repeating simulation, the simulator only processes altered events and their influences by recovering causality violation as described in Section 2. Figure 1 shows an example of the repeating simulation, which starts from the "altered scenario" at LP_n and influences $LP_{n-2} - LP_{n+3}$. For the sake of simple illustration, only the priority queues of future events are illustrated in the figure and the priority queues of anti-messages and state data are omitted.

The repeating simulation starts from altered scenarios, in which the starting LPs roll back their changes. The changes then spread to other LPs, which recursively roll back their changes, and again cascade to other LPs. Furthermore, multiple alternations are allowed, and in this case, the processes are conducted in parallel and their concurrency is controlled by the optimistic synchronization.

The altered scenarios are assigned to LPs according to their space and time, that is, starting LP's identifier (ID) and starting time. Based on the ID and time, the starting LPs correct their local clock time to the altered scenarios' time from infinity (or the finish time of the initial baseline

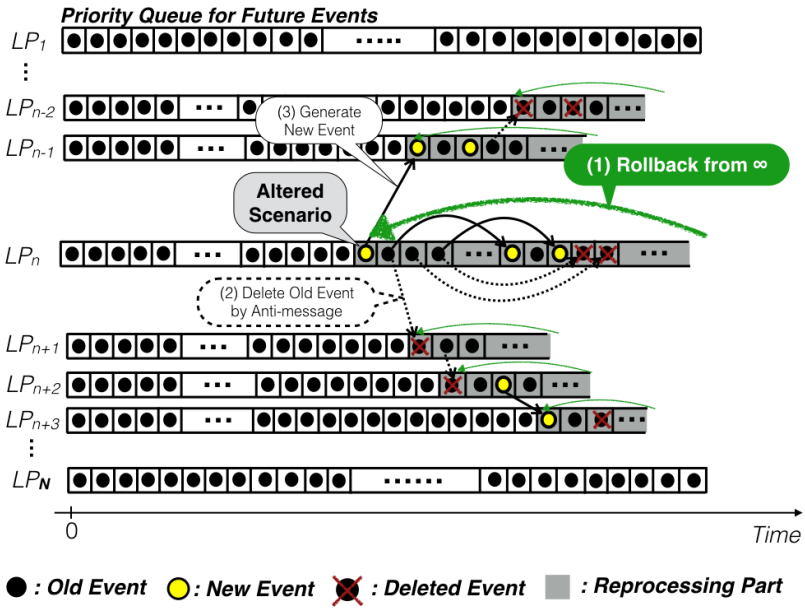


Fig. 1. Repeating Simulation

execution) and send anti-messages to the neighbor LPs in the same way as a rollback in traditional optimistic synchronization. After that, the starting LPs reprocess scenarios based on the local clock time. When new events are assigned to other LPs, they are immediately sent to the assigned LPs. When the assigned LPs receive events and/or anti-messages, they recover causal violation in the same way as traditional optimistic synchronization. Specifically, if they receive events and/or anti-messages which have a timestamp earlier than their local clock, they correct their clock and send new anti-messages recursively. On the other hand, if they receive events and/or anti-messages that have a timestamp later than their local clock, they insert the events to the future event priority queue (in the case of events) or remove the events (in the case of anti-messages). The GVT is calculated in the same way as optimistic PDES starting from the altered scenario time, and if the GVT reaches finish time, then the repeating simulation is finished.

Figure 1 illustrates an example of this process. In this example, the repeating simulation starts from the "*Altered Scenario*" at LP_n. Firstly, (1) the LP_n corrects its local clock time and rolls back its state. After that, (2) the LP_n sends anti-messages to LP_n and LP_{n+1} to delete old events making the causality violation. Then, (3) it processes events from the corrected time and generates new events sending to LP_{n-1} and LP_n. In LPs which receive new events or anti-messages (i.e. LP_{n-1} and LP_{n+1}), they roll back their local clock time and state and process their events recursively. Finally, the influence cascades to some parts of entire LPs shown as "*Reprocessing Part*", and the simulation reaches finish time and terminates.

Compared to a traditional (complete, non-differential) repeating simulation, which starts from the beginning and processes all scenarios, this approach could significantly reduce the number of processed events.

3.3 Static Analysis of Exact-Differential Simulation

This subsection discusses the performance characteristics of the exact-differential simulation. We first formulate speed up based on the number of processing events and reprocessing rate. We then analyze and discuss the reprocessing rate in the repeating execution. Holistic performance evaluation of the actual simulator including a simulation model, a synchronization method, thread management, the number of parallel processors, hardware environments, and so forth, is discussed and evaluated in Section 4 and Section 5. In this section, we focus on the relation between the performance and the number of processing events since this is a key deciding performance factor derived directly from the exact-differential simulation.

Table 1 lists notations used for the reprocessing rate formulation.

Table 1. Notation for Static Analysis

	Description
E	Set of events
$T(E)$	Wall-clock time to process events E
E_{all}	Set of all events in the repeating simulation (processed events in naive simulation)
$E_{ex\text{-}diff}$	Set of ex-diff reprocessing events in the repeating simulation
t_{all}	Average elapsed time of processing single event in naive simulation
$t_{ex\text{-}diff}$	Average elapsed time of processing single event in ex-diff repeating simulation
α	Overhead of exact-differential repeating simulation from naive simulation
S	Speed up of exact-differential repeating simulation from naive simulation
r	Reprocessing rate ($\left\langle\left\langle \lvert E_{ex\text{-}diff}\rvert \right\rangle\right\rangle / \langle\lvert E_{all}\rvert\rangle$)
$\lvert E_{local}(lp, m)\rvert$	Number of local events between m-th and $m+1$-th message communications at lp
i	Counter of Average-case Model
$\lvert LP_{all}\rvert$	Number of all LPs
$\lvert LP_{influence}(i)\rvert$	Number of influenced LPs in the repeating simulation in average model

3.3.1 Speed Up. First, we estimate the speed up of the exact-differential simulation over a simulation which naively processes all events (referred to as *naive simulation*). Let E be a set of events and $T(E)$ be the wall-clock time to process E. We use the operator $|*|$ as the number of elements in $*$. Suppose E_{all} is a set of all events in the repeating simulation, the wall-clock time of the repeating simulation by the naive simulation is denoted as $T(E_{all})$. t_{all} is defined as the average wall-clock time for processing a single event in the naive simulation. The wall-clock time using the naive simulation is expressed as follows.

$$T(E_{all}) \quad = \quad t_{all} \cdot |E_{all}|$$

Similarly, let $E_{ex\text{-}diff}$ ($\subseteq E_{all}$) be a set of processed events in the repeating execution by using the exact-differential simulation, and $t_{ex\text{-}diff}$ be the average wall-clock time of processing a single event in the exact-differential repeating simulation. The wall-clock time of the repetition using the exact-differential simulation is expressed as follows.

$$T(E_{ex\text{-}diff}) \quad = \quad t_{ex\text{-}diff} \cdot |E_{ex\text{-}diff}|$$

We define the *speed up* of the exact-differential simulation as the division of the naive simulation's wall-clock time (i.e. $T(E_{all})$) by the exact-differential simulation's wall-clock time (i.e. $T(E_{ex\text{-}diff})$).

The speed up is expressed as follows.

$$\begin{aligned}(Speed\ Up) &:= \frac{T(E_{all})}{T(E_{ex\text{-}diff})} \\ &= \frac{t_{all}}{t_{ex\text{-}diff}} \cdot \frac{|E_{all}|}{|E_{ex\text{-}diff}|} \\ &= \frac{\alpha}{r},\end{aligned}$$

where $\alpha := t_{all}/t_{ex\text{-}diff}$ is the overhead of event processing in the exact-differential simulation from the naive simulation, and we refer to r as *reprocessing rate*, which is defined as follows.

$$r := |E_{ex\text{-}diff}|/|E_{all}| \tag{1}$$

We will discuss its details in the next subsection.

3.3.2 Reprocessing Rate. Second, we estimate reprocessing rate, namely parameter r as stated before. From the view point of event processing in each LP, the reprocessing in the exact-differential simulation is basically caused by event communication, which is characterized by the frequency of events communication and the number of neighboring LPs to communicate events. For example, in Figure 1, the number of reprocessing events is obviously related to the number of events sent. If the frequency of sending events becomes higher, the number of reprocessing events can be expected to increase.

Therefore, in the following part, we only focus on the performance features caused by the frequency of event communication and the number of neighboring LPs to communicate events. Other performance factors, such as the simulation model, the synchronization method, and the hardware environment are simplified or omitted in this discussion. The holistic performance results are shown in Section 5.

Let $E_{local}(lp, m)$ be a set of processing events between message communications which causes new event at the destination LP as shown in Figure 2. $E_{local}(lp, m)$ has two independent variables, lp and $m \in \mathbb{N}$. lp is the identifier of the LP. m identifies a message. It starts with 0 and increments per message in each LP. If an event generates two messages, $E_{local}(lp, m)$ is empty (e.g. $|E_{local}(LP_n, 2)|$ in Figure 2). Then, the total number of reprocessing events using the exact-differential simulation is represented as follows.

$$|E_{ex\text{-}diff}| = \sum_{lp \in LPs} \sum_{m=0}^{|Msg(lp)|} |E_{local}(lp, m)|,$$

where $|Msg(lp)|$ is the number of messages that are sent from lp and causes new event at the destination.

For further discussion, we consider an average case to simplify the model as follows.

- Every LP processes $\langle |E_{local}| \rangle$ events in average between two message communications.
- Let i be a counter which initializes at zero and increments every time when each influenced LP processes $\langle |E_{local}| \rangle$ events. The simulation will finish at end (i.e. $0 \leq i \leq end$).
- The number of influenced LPs at i is defined as $|LP_{influence}(i)|$.

Figure 3 shows the exact-differential simulation represented by the simplified average model. Under this assumption, the number of influenced events $|E_{ex\text{-}diff}|$ is expressed as follows.

$$|E_{ex\text{-}diff}| = \sum_{i=0}^{end} |LP_{influence}(i)| \cdot \langle |E_{local}| \rangle .$$

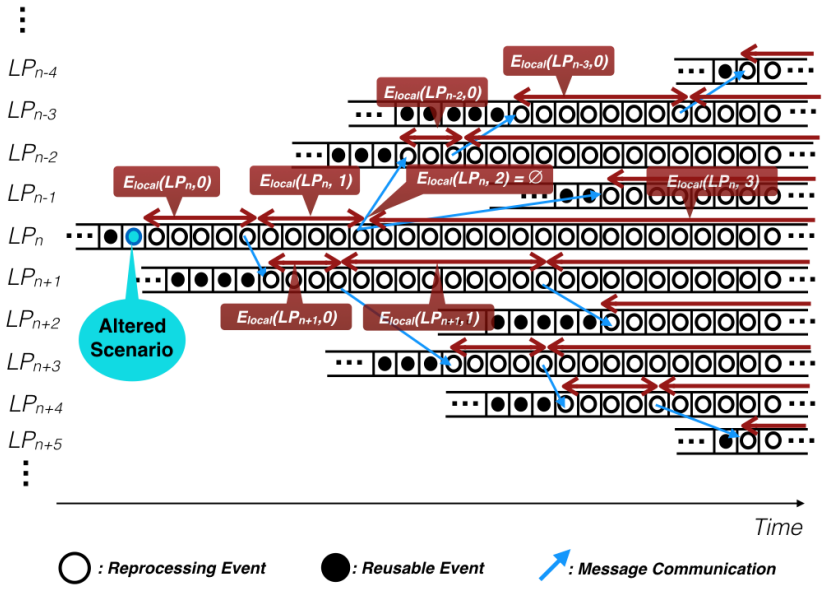


Fig. 2. Cascading of Reprocessing Events in Exact-Differential Simulation.

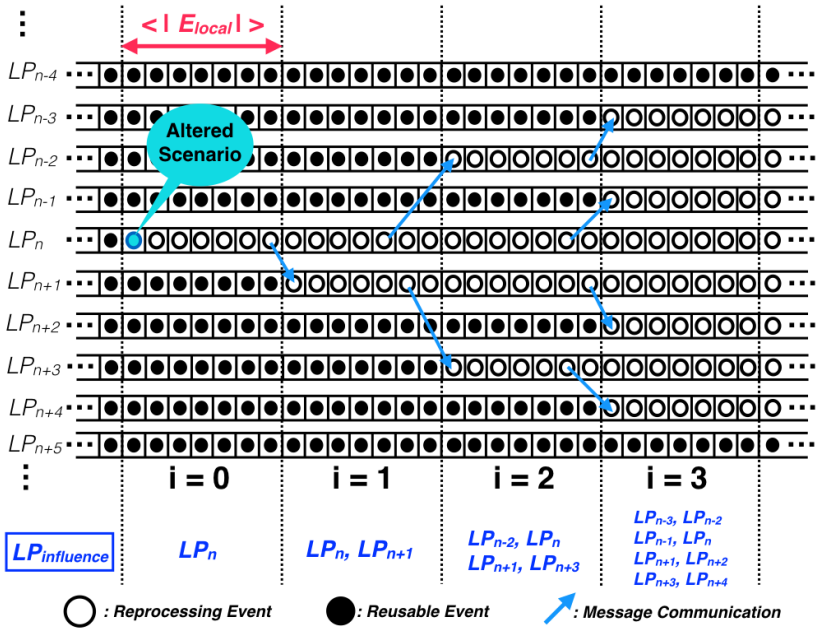


Fig. 3. Exact-Differential Simulation of Message Forwarding in Average Model.

The number of events in the naive simulation $|E_{all}|$ is expressed as follows.

$$\begin{aligned} |E_{all}| &= \sum_{i=0}^{end} |LP_{all}| \cdot \langle |E_{local}| \rangle \\ &= end \cdot |LP_{all}| \cdot \langle |E_{local}| \rangle , \end{aligned}$$

where $|LP_{all}|$ is the total number of LPs.

To summarize, the reprocessing rate is expressed as follows.

$$\begin{aligned} r &:= |E_{ex\text{-}diff}| / |E_{all}| \\ &= \frac{\sum_{i=0}^{end} |LP_{influence}(i)| \cdot \langle |E_{local}| \rangle}{end \cdot |LP_{all}| \cdot \langle |E_{local}| \rangle} \\ &= \frac{\sum_{i=0}^{end} |LP_{influence}(i)|}{end \cdot |LP_{all}|}. \end{aligned}$$

Reprocessing Rate in Random Forwarding (PHOLD benchmark): We further estimate the reprocessing rate in the case when the events are forwarded randomly. If there is one altered event and each LP sends a message to one LP after processing $\langle |E_{local}| \rangle$ events as shown in Figure 3 (this is, for example, the case in the PHOLD benchmark discussed later in Section 5), then the number of influenced LPs at i can be calculated as below. First, the number of LPs which have not been influenced at i is expressed as $|LP_{all}| - |LP_{influence}(i)|$. Then we can define a recurrence relation as follows.

$$|LP_{influence}(i+1)| = |LP_{influence}(i)| + |LP_{influence}(i)| \cdot \frac{|LP_{all}| - |LP_{influence}(i)|}{|LP_{all}|}.$$

Suppose $|LP_{influence}(0)| = 1$ (i.e. the what-if scenario alters the state of one LP), then the recurrence relation can be calculated as follows.

$$\begin{aligned} \frac{|LP_{all}| - |LP_{influence}(i+1)|}{|LP_{all}|} &= \left(\frac{|LP_{all}| - LP_{influence}(i)|}{|LP_{all}|} \right)^2 \\ \therefore |LP_{influence}(i)| &= |LP_{all}| - |LP_{all}| \cdot \left(1 - \frac{1}{|LP_{all}|} \right)^{2^i} \end{aligned}$$

Here, we are interested in how the influenced LPs expand over the simulation. Thus, we suppose $|LP_{all}|$ is large enough (i.e. $1/|LP_{all}| \to 0$) and 2^i is relatively small (otherwise, $|LP_{influence}|$ would easily reach to $|LP_{all}|$, and we do not need to analyze the behavior of the influence any more). In this case, the result becomes much more straightforward.

$$\begin{aligned} |LP_{influence}(i)| &= |LP_{all}| - |LP_{all}| \cdot \left(1 - \frac{1}{|LP_{all}|} \right)^{2^i} \\ &\sim |LP_{all}| - |LP_{all}| \cdot \left(1 - \frac{2^i}{|LP_{all}|} \right) \\ &= 2^i. \end{aligned}$$

The reprocessing rate in the average model is formulated as follows.

$$\begin{aligned} r &:= |E_{ex\text{-}diff}|/|E_{all}| \\ &= \frac{\sum_{i=0}^{end} |LP_{influence}(i)|}{end \cdot |LP_{all}|} \\ &= \frac{\sum_{i=0}^{end} 2^i}{end \cdot |LP_{all}|} \\ &= \frac{2(2^{end}-1)}{end \cdot |LP_{all}|}. \end{aligned}$$

In actual scenarios, *end* is usually determined by the finish time of simulation. Suppose the timestamp interval in the i-th iteration is the same for each LP, *end* is defined as

$$end = \frac{|T_{finish}|}{c' \langle |E_{local}| \rangle}, \tag{2}$$

where T_{finish} is the finish time of simulation, and constant value c' is the average of timestamp interval between two sequenced events. Finally the formula is expressed as follows.

$$\begin{aligned} r &:= |E_{ex\text{-}diff}|/|E_{all}| \\ &= \frac{2(2^{\frac{|T_{finish}|}{c'\langle |E_{local}| \rangle}}-1)}{\frac{|T_{finish}|}{c'\langle |E_{local}| \rangle} \cdot |LP_{all}|} \\ &= \frac{2c' \langle |E_{local}| \rangle (2^{\frac{|T_{finish}|}{c'\langle |E_{local}| \rangle}}-1)}{|T_{finish}| \cdot |LP_{all}|}. \end{aligned} \tag{3}$$

Reprocessing Rate in Grid LPs (Traffic Simulation): In this subsection, we estimate the reprocessing rate in the case when the events are communicated on a grid structure as shown in Figure 4 (this is a similar case to the traffic simulation discussed in Section 5). In this case, if the LPs send events to all 4 neighbors at every interval, then the number of influenced LPs at i is expressed as $|LP_{influence}(i)| = |LP_{influence}(i)| + 4i$. Suppose $|LP_{influence}(1)| = 1$ (i.e. the what-if scenario alters the state of one LP), then the recurrence relation can be calculated as follows.

$$|LP_{influence}(i)| = 2(i-1) \cdot i + 1,$$

where $|LP_{influence}(i)| \leq |LP_{all}|$. Hence, the reprocessing rate can be calculated as follows.

$$
\begin{aligned}
r &:= |E_{ex\text{-}diff}|/|E_{all}| \\
&= \frac{\sum_{i=0}^{end} |LP_{influence}(tw)|}{end \cdot |LP_{all}|} \\
&= \frac{\sum_{i=0}^{end} \{2(i-1) \cdot i + 1\}}{end \cdot |LP_{all}|} \\
&= \frac{3^{-1} \cdot (2end^3 + end)}{end \cdot |LP_{all}|} \\
&= \frac{2end^2 + 1}{3|LP_{all}|} \\
&= \frac{1}{3|LP_{all}|} \left\{ 2 \left(\frac{|T_{finish}|}{c' \langle |E_{local}| \rangle} \right)^2 + 1 \right\}. \qquad (4)
\end{aligned}
$$

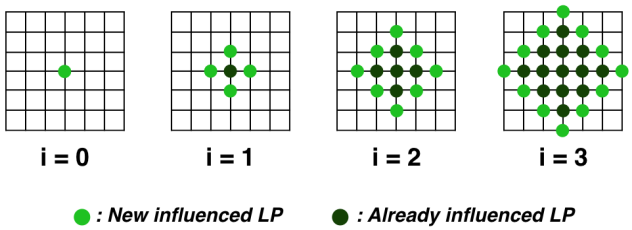


Fig. 4. Influenced LPs in a Grid Structure.

Summarizing the above equations (Equation (3)(4)), these two results indicate that under the fixed finish time of simulation, the performance is improved by maximizing $\langle |E_{local}| \rangle$, which can be achieved, for instance, by the reduction of message communication and the increase of the number of events that are processed locally. In addition, their respective performance profiles have completely different characteristics. The performance improvement of random forwarding is more difficult to achieve than that of Grid LPs since the reprocessing rate of random forwarding (i.e. r) is affected by the term $\langle |E_{local}| \rangle \cdot 2^{1/\langle |E_{local}| \rangle}$ as Equation (3), whereas the reprocessing rate of Grid LPs is affected by the term of $\langle |E_{local}| \rangle^{-2}$ in Equation (4). The respective speed ups (i.e. α/r) for Equation (3) and Equation (4) are highly different especially for small $\langle |E_{local}| \rangle$ since $(\langle |E_{local}| \rangle \cdot 2^{1/\langle |E_{local}| \rangle})^{-1} \ll \langle |E_{local}| \rangle^2$ when $\langle |E_{local}| \rangle$ is small.

These results are mainly determined by the number of connected neighbors in each LP. In the random-forwarding case, each LP can potentially communicate with any other LP; in the Grid case, however, each LP may only communicate with up to 4 neighboring LPs. Hence, in general, in the former case, the set of influenced LPs would expand exponentially over the simulation, leading to an exponential increase in the reprocessing rate (Equation (3)); whereas in the latter case, the influenced LPs would expand polynomially, resulting in an polynomial increase in the reprocessing rate (Equation (4)). These behaviors are observed also in the experimental analysis in Section 5.

The theoretical analysis in this section may provide a good guidance for the modeler to effectively utilize the exact-differential simulation. In order to achieve a good performance improvement, a clustered distribution of the simulation states to LPs is required so that connectivity and messages exchanged between LPs are minimized.

4 IMPLEMENTATION

We have implemented the proposed exact-differential simulation by extending an optimistic PDES simulator. An overview of the system architecture is illustrated in Figure 5 and Figure 6. The system consists of three modules: application module, simulation runtime module, and communicator module.

In the application module, the actual simulation logic and algorithms, such as the event handlers for the PHOLD benchmark and the traffic models, are included. In addition, the exact-differential simulation mechanism is completely independent of the application code and logic. This module only processes an event from the simulation runtime module, and then returns new events and a new updated state to the simulation runtime module by using a simple C/C++ interface.

```
void eventHandler(Event** newEvents, State* newState,
                  const Event& event, const State& state);
```

The function must be implemented by the modeler to be referentially transparent between the baseline simulation and the repeating simulation. In other words, among the baseline and the repetitions, the event handler always gives the same new events and new state for the same argument. By doing so, the exact-differential simulation can safely skip the redundant part of the baseline simulation. It ensures the results of the exact-differential simulation and the entire simulation (i.e. the traditional way to repeat simulations by processing every scenario) are identical for the same what-if scenario. Thus in the stochastic model, for instance, the modeler needs to implement the random behavior in such a way that the event handler returns the same value for the same argument (in practice, such mechanism can be implemented by using the same random seed between the baseline and repeating simulations)

The simulation runtime module manages storage and retrieval of simulation logs such as events, anti-messages, and state data. It also controls global ordering of events based on optimistic PDES, including rollback, restarting, and GVT calculation. The state data is obtained by naive copy state saving in which the LPs simply copy their state value in every event processing task and manage them through a timestamp-ordered queue. This module also manages the event processing threads by using a thread pool and the "least timestamp first" queue (LTSF queue) for parallel processing that is maintained in shared memory. The core mechanism of exact-differential simulation is implemented in this module and it is not required to add application specific code.

Finally, the communicator module controls node to node communications, which include events communication and GVT calculation. Its implementation is based on an asynchronous messaging interface of MPI (i.e. `MPI_Isend` and `MPI_Irecv`).

More details of the implementation are available in the public repository of our optimistic PDES simulator, *ScaleSim* [12].

4.1 Implementation of Initial Baseline Simulation

In the initial baseline simulation, as shown in Figure 5, events are basically processed in the same way as the traditional optimistic PDES described in Section 2, where the LPs speculatively process the events in parallel with the use of rollback and restarting operations. During the process, GVT is calculated periodically in order to identify which events are no longer rolled back and/or deleted. Unlike the original optimistic PDES, events, anti-messages, and state values with a timestamp

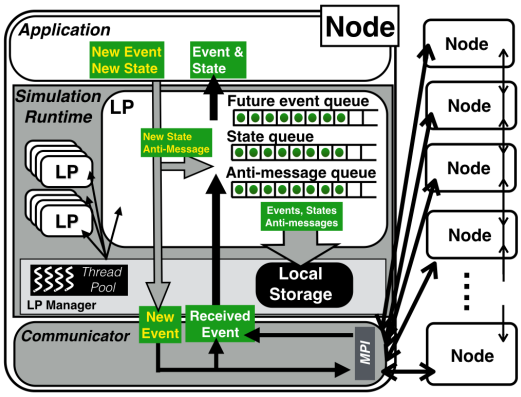


Fig. 5. Initial Baseline Simulation.

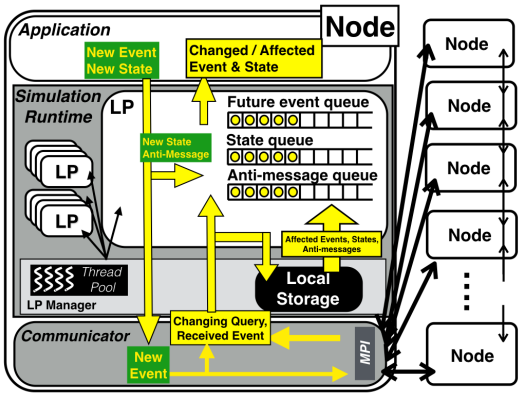


Fig. 6. Exact-Differential Repeating Simulation.

smaller than GVT are stored in the local storage for later repeating simulation, instead of being released. The events, anti-messages, and state values are basically stored in memory space. Therefore, one of the limitations of the proposed approach is the memory capacity of te system. This can be overcome, for example, by using a system with a larger memory space or extending the local storage to secondary disk, where it could, however, lead to performance degradation due to the additional overhead of the disk accesses.

4.2 Implementation of Repeating Simulation

In repeating simulation, the simulator first accepts as input a what-if scenario defined by the virtual time and the LP ID of an altering scenario and a query type such as ADD, DELETE, or UPDATE_STATE. Algorithm 1 describes the processing steps of the what-if scenario. In the case of ADD (line 5 – 9), a new event generated from the ADD query is inserted into the event queue and the local clock time is corrected to the new event's time (i.e. rollback). After that, anti-messages are sent to all affected LPs. In the case of DELETE (line 10 – 15), an old event generated from the DELETE query is removed. The local clock time is corrected to the old event's time (i.e. rollback). Finally, anti-messages are sent to the affected LPs. In the case of UPDATE_STATE (line 16 – 20), the state is updated by using a value of the scenario and its local time is corrected to the updated time (i.e. rollback). After processing the what-if scenario, events are processed according to the corrected time (line 25 – 27).

Figure 6 shows the event processing flow in the repeating simulation. Unlike usual optimistic PDES, during the event process, events, anti-messages, and state values are required to be loaded from storage if necessary. Algorithm 2 describes a mechanism to load the events, anti-messages, and state values. We extend the function of receiving event presented in line 3 – 24 of Algorithm 2. In our simulation, events received from other LPs are buffered before they are inserted to the event queues (line 1). If a newly received event has a receiving time smaller than the minimum loaded time (which is initialized to infinity), then the stored events, anti-messages, 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 newly received event is inserted to the event queue as usual (line 25).

Algorithm 1 What-if Scenario Processing

```
/* What-if Scenario Processing */
while hasWhatIfScenario() do
    scenario ← getWhatIfScenario()
    lp ← getLP(scenario.lp_id)
    if scenario.type = ADD then
        newEvent ← scenario.event
        lp.insert(newEvent)
        lp.rollback(newEvent.time)
        lp.sendAntiMessagesToNeighbors()
    else if scenario.type = DELETE then
        oldEvent ← scenario.event
        lp.delete(oldEvent)
        lp.sendAntiMessage(oldEvent)
        lp.rollback(oldEvent.time)
        lp.sendAntiMessageToNeighbors()
    else if scenario.type = UPDATE_STATE then
        newState ← scenario.state
        lp.updateState(newState)
        lp.rollback(scenario.time)
        lp.sendAntiMessagesToNeighbors()
    end if
end while

/* Event Processing */
while getGlobalVirtualTime() < TIME_TO_FINISH do
    Reprocess influenced events with optimistic PDES
end while
```

5 EVALUATION

In this section, we present an experimental evaluation of the proposed exact-differential simulation using two case studies: (1) the PHOLD, which is a standard synthetic benchmark of PDES, and (2) a microscopic traffic simulation of the city of Tokyo. The objective of the evaluation is to measure the performance of the exact-differential simulation under both a synthetic workload as well as a real-world application. According to the theoretical discussion in Section 3, the key theoretical performance factor is the reprocessing rate (as defined in Equation (1)). Thus, we evaluate not only the elapsed time of the simulation but also the reprocessing rate since the elapsed time involves other factors unrelated to the algorithm itself, such as the computational platform and the specific implementation. The reprocessing rate gives an indicator of the performance improvement that is only due to the algorithm.

Table 2 presents the details of the evaluation testbed. We used the TSUBAME 2.5 supercomputer [30] located at the Tokyo Institute of Technology, Japan, which consists of a 12-core CPU and 54GB main memory per node, while the nodes are connected by InfiniBand. The SUSE Linux Enterprise Server 11 sp 3, Open MPI 1.6.5, and GCC 5.2 (C++11) were used as its software environment. To perform the simulations, we employed the *ScaleSim* [12] simulator, which supports exact-differential simulation.

Algorithm 2 Extension of Receiving Event in Exact-Differential Simulation

```
1: while receiveEventBuffer.hasEntity() 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:     oldAntiMessages ← store.getAntiMessage(from, to)
9:     oldStates ← store.getState(from, to)
10:    while oldEvents.hasEntity() do
11:      loadedEvent ← oldEvens.dequeue()
12:      eventQueue.insert(loadedEvent)
13:    end while
14:    while oldAntiMessages.hasEntity() do
15:      loadedAntiMessages ← oldAntiMessages.dequeue()
16:      antiMessageQueue.insert(loadedAntiMessage)
17:    end while
18:    while oldStates.hasEntity() 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
```

Table 2. Cluster Configurations.

Service	TSUBAME 2.5 in Tokyo Tech.
# of Nodes	2, 4, 8, 16, 32
CPU	Intel Xeon X5670/2.93GHz × 2
Memory	54GB per Node (49 GB for application)
Network	QDR InfiniBand Interconnect
OS	SLES 11 SP3
MPI	Open MPI 1.6.5
C/C++ Compiler	GCC 5.2 (C++11)
Simulator	ScaleSim Ver. at Aug 29, 2016

5.1 Evaluation with the PHOLD Benchmark

PHOLD is a standard benchmark widely used by the PDES research community [10]. A small portion of PHOLD's workload is used for generating random numbers, which are required for calculating destination LPs of new events and events' simulation time increment. On the other hand, much of the overall execution involves time synchronization between LPs and data communication between nodes.

Leveraging on previous studies [1, 2, 5, 24], we define PHOLD's workload using two parameters as follows.

- An event with timestamp t is forwarded to the next LP with a new timestamp $t' = t + \Delta t$ per event handling in each LP.
- Δt is randomly generated using exponential distribution with a parameter λ.
- The destination LP is randomly selected using *Remote Ratio*.

More specifically, Δt is determined by the following equation.

$$Pr[a \leq \Delta t \leq b; \lambda] = \int_a^b \lambda e^{-\lambda x} dx,$$

where a, b are constant non-negative real numbers. Since the mean of the exponential distribution is λ^{-1}, λ essentially implies the event density. The event density increases with an increasing λ.

Remote Ratio is the frequency of sending events to other LPs and is simply determined by the rate at which another LP is selected as the next destination. For example, a Remote Ratio of 10% means that 10% of newly generated events are sent to other LPs, while 90% are sent back to the same LP.

This PHOLD benchmark aims to show the degree by which its parameters impact the reprocessing rate and the execution time. Table 3 presents the default parameters of the evaluation and Table 4 provides the execution results. These default values are set based on the previous work [1, 2, 5, 24]. We evaluate the simulation scenario with 100,000 LPs and 1,600,000 initial events (each LP has 16 events initially) using 16 nodes with 192 cores in total. The default value of λ is set to 1.0 and that of Remote Ratio is set to 2^{-4}. The simulation end time is fixed to 5.0 in all setting so that the memory usage becomes up to 49GB, which is the limitation of our computational environment. The number of resulting events are 7,275,345 in the default setting. To set up a what-if scenario, we randomly select one state and change it from the beginning of the simulation. The execution of each setup is conducted 10 times, and the median values of the results are presented in Table 4.

Table 3. Default Parameters.

Parameter	Default Value
# of LPs	100,000
# of Initial Events	1,600,000
Simulation End Time	5.0
λ	1.0
Remote Ratio	2^{-4}
# of Nodes	16 Nodes (192 Cores)
What-if Scenario	Randomly Change 1 state from T=0

Table 4. Default Execution Results.

Entire Baseline Simulation	
– # of Resulting Events	7,275,345
– Elapsed Time	50,893 ms
Exact-Differential Simulation	
– # of Resulting Events	321,690
– Elapsed Time	12,672 ms

5.1.1 Impact of Event Density. We first evaluate the impact of event density λ. We investigated different various values of λ and set other parameters to a constant value as presented in Table 3. Figure 7 shows the number of resulting events with λ extending from 0.25 to 1. In this figure, the left vertical axis displays the value range for the number of resulting events, which is also represented by bars; and the right vertical axis displays the value range for the reprocessing rate, which is also represented by a line.

According to the results, the reprocessing rates of the exact-differential simulation are 0.000028, 0.00011, 0.0019, and 0.044 when λ equals to 0.25, 0.5, 0.75, and 1, respectively. The results show that

the reprocessing rate increases as the event density and the number of influenced events increase. They also indicate that such increase is exponential under linear increase of λ.

These results are consistent with the theoretical analysis as discussed in Section 3.3. Since λ indicates the event density, λ and $1/c'$ in Equation (2) have a linear relationship. Namely, with a linear increase of λ, the reciprocal of timestamp interval between communications (i.e. $1/c'$) increases accordingly. Thus, with the fixed simulation end time (i.e. with $|T_{finish}|$ in Equation (2) a constant), the theoretical reprocessing rate defined in Equation (3) increases exponentially with a linear increase of λ.

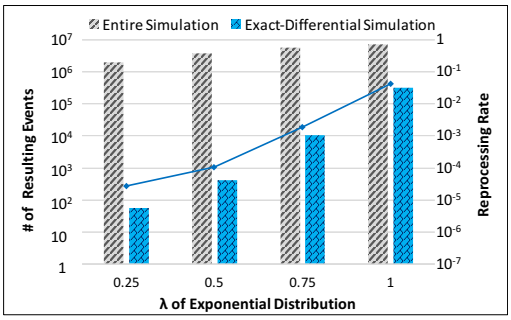


Fig. 7. # of Resulting Events Comparing with Different Event Density (λ of Exponential Distribution).

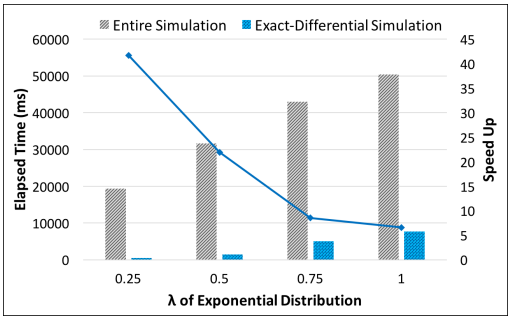


Fig. 8. Elapsed Time Comparing with Different Event Density (λ of Exponential Distribution).

Figure 8 shows the elapsed time of the proposed simulations. More precisely, the bars represent the elapsed time of simulations and the line represents the speedup of simulation, which is defined as (Elapsed Time of Entire Simulation) / (Elapsed Time of Exact-Differential Simulation). The results clearly indicate that the exact-differential simulation outperforms the entire simulation.

The trend of exact-differential simulation's elapsed time is somewhat similar to that of the reprocessing rate. When λ increases, the elapsed time of the exact-differential simulation increases as the number of resulting events increases. In addition, we observe that the number of events increases approximately by a factor of 4 since the simulation end time is fixed, but the elapsed time is not increased by the same factor, although theoretically this should be the case. This is due to the fact that our system handles the scenarios more efficiently when λ is large. The events with closer timestamp are processed collectively and their respective concurrency management is done jointly. As a result, the cost of the event processing becomes lower than in the case of a smaller lambda. Thus, even if the number of events increases by a factor of 4, the elapsed time is not increased accordingly Furthermore, the exact-differential simulation achieves approximately 41.7x, 21.9x, 8.6x, and 6.5x performance improvement when λ equals to 0.25, 0.5, 0.75, and 1, respectively..

5.1.2 Impact of Remote Ratio. This section presents the evaluation of the performance of the proposed exact-differential simulation based on remote ratio. In the simulations, the remote ratio extends from 2^{-8} to 2^{-1} and other parameters are set to constant as presented in Table3.

Figure 9 presents the number of resulting events (as represented by bars) and the processing rate (as represented by a line) at different remote ratios. Clearly, the number of influenced LPs and the reprocessing rate increase as the remote ratio increases. It is also worth noting that, when the remote ratio increases from 2^{-4} to 2^{-1}, the reprocessing rate is saturated. This is because the fraction of LPs not affected by the change in state falls exponentially.

In the case of remote ratio from 2^{-8} to 2^{-4}, the experimental results are consistent with the theoretical analysis in Section 3.3. As the remote ratio increases linearly, so does the reciprocal

of the average number of local processed events between communications (i.e. $1/\langle|E_{local}|\rangle$ in Equation (3)). Thus, under a fixed simulation end time (i.e. with $|T_{finish}|$ in Equation (2) a constant), the theoretical reprocessing rate increases exponentially as remote ratio increases linearly.

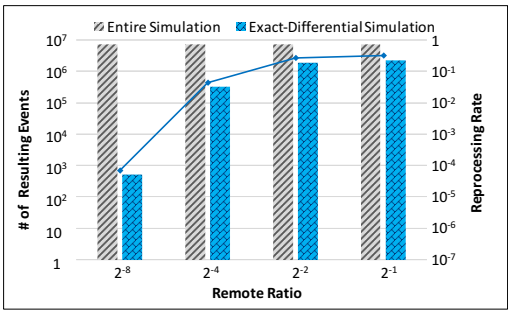


Fig. 9. # of Resulting Events Comparing with Different Remote Ratio.

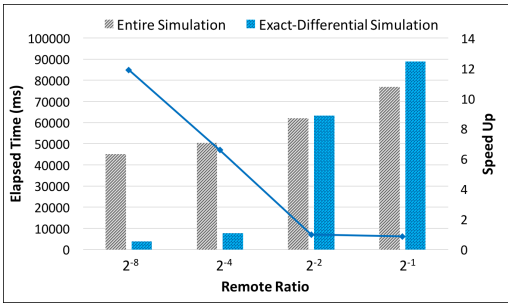


Fig. 10. Elapsed Time Comparing with Different Remote Ratio.

The elapsed time for several different remote ratios is shown in Figure 10, where the bars represent the elapsed time and the line represents the speedup of simulation. The results clearly show that when the remote ratio increases, the elapsed time becomes larger, which leads to smaller speedup. The exact-differential simulation yields approximately 11.88x, 6.59x, 0.98x, and 0.87x speed up when remote ratio equals to 2^{-8}, 2^{-4}, 2^{-2}, and 2^{-1}, respectively. When remote ratio is smaller than 2^{-2}, the exact-differential simulation outperforms the entire simulation. However, when the remote ratio is larger than 2^{-2}, the number of reprocessing events becomes too large to achieve a better performance than the entire baseline simulation. Although the exact-differential simulation processes fewer events than the entire simulation, the overhead of rollback, canceling, and reprocessing reduces runtime performance.

Most importantly, the results indicate that it is necessary to reduce the remote ratio in order to get good performance in the exact-differential simulation. Reduction in remote ratio can be achieved by partitioning of simulation states to LPs in a way to minise inter-LP communication.

5.1.3 Impact of the Number of Nodes. Finally, we evaluate the strong scalability of the exact-differential simulation of the PHOLD benchmark using our simulator with different number of nodes and the same input data. Figure 11 shows the elapsed time of simulation with the number of working nodes ranging from 2 to 32 (i.e. from 24 cores to 384 cores). The results clearly show that the performance of the exact-differential simulation always outperforms that of the entire simulation. However, the performance improvement decreases when the number of working nodes increases. This is because the problem size (i.e. the number of processed events) is not large enough to process using multiple nodes. The performance improvement in the exact-differential simulation is saturated at 16 nodes, and the performance reduces when more nodes are involved in the simulation. Basically, the entire simulation is executed only once, and the exact-differential repeating simulations are executed many times. Thus, in practice, the saturation point would be the best setting for the number of nodes.

The evaluation with the PHOLD benchmark has provided very useful insights about the performance of the proposed exact-differential simulation approach. However, PHOLD is a very simple synthetic benchmark, where event processing only involves random message communications with very little state saving and the events are distributed uniformly among LPs. For a more realistic and

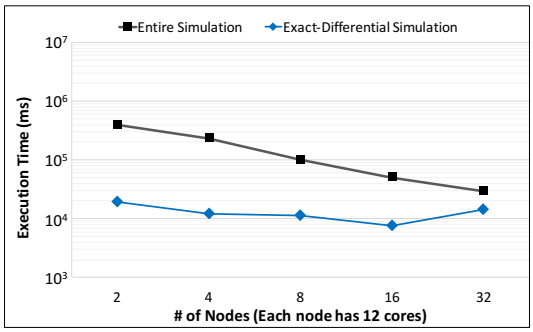


Fig. 11. Strong Scaling.

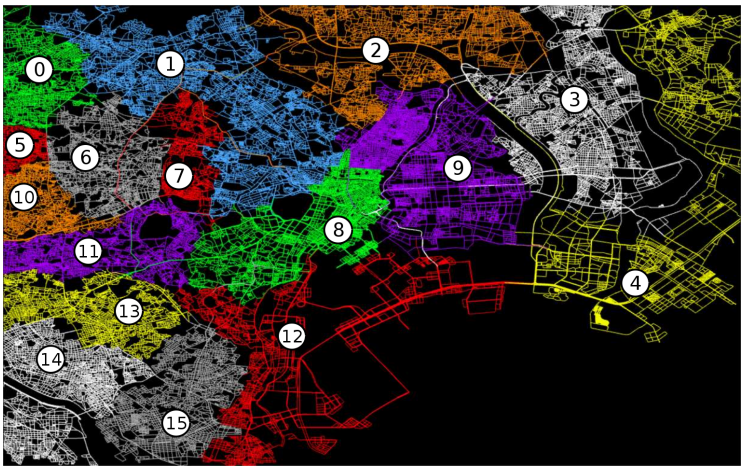


Fig. 12. Road Map of Tokyo Center Area and its 16 Partitions to Computational Nodes using METIS.

thorough evaluation, we have utilized a traffic simulation for the city of Tokyo. In this simulation, models are more complex, and the events (i.e. vehicles) are skewedly distributed among LPs (i.e. roads and junctions). The following section presents the results of this analysis

5.2 Evaluation with the Tokyo Traffic Simulation

We first examine the performance of a microscopic traffic simulation of Tokyo on top of the ScaleSim simulator [12]. Table 5 presents the configuration of the simulation scenarios. We simulate the traffic of Tokyo center area, which includes 161,364 junctions and 203,363 roads. The roads are divided by a k-way graph partitioning algorithm [20, 21]. Figure 12 shows the overview of the Tokyo city, where each line represents the actual road, and the road network is partitioned into 16 computational nodes denoted by different colors.

We create movements of vehicles based on Tokyo's statistical data collected by the Ministry of Land, Infrastructure, Transport and Tourism (MLIT) in 2011, where totally 5000 vehicles depart from their origin within 3 hours. The Origin Destination data (*OD data*) of each vehicle is randomly selected, and the path is preprocessed using a shortest path algorithm. Figure 13 shows the distribution of the number of passing vehicles in each junction (i.e. each LP). Note that this is a

log-log graph. The distribution is highly skewed. In around 45% of the junctions, there are only up to 4 passing vehicles, whereas there exist a few junction which over 100 vehicles pass through. The maximum number of the passing vehicles in a junction is 476. The average number of the passing vehicles is 8. Figure 14 shows the distribution of the vehicle's path lengths (i.e. the number of LPs a vehicle passes through). The major path length of the vehicles is from 101 to 200. And from 201, its frequency decreases as the increase of the path length. The maximum and average length of the vehicle paths are 1139 and 283, respectively.

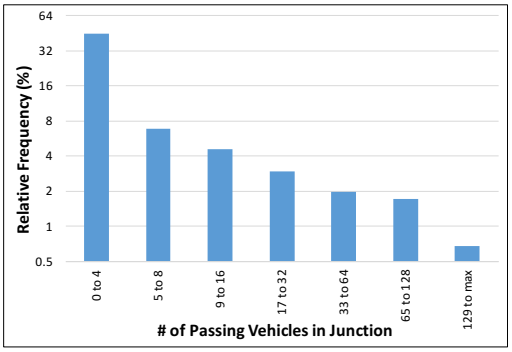


Fig. 13. # of Passing Vehicles in Each Junction.

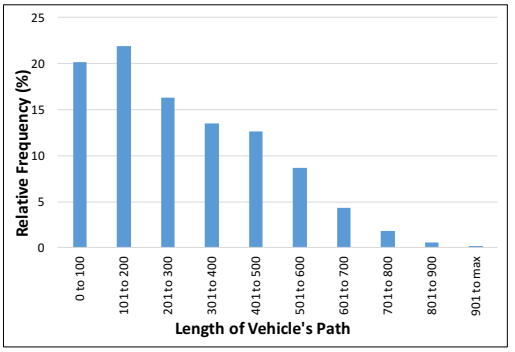


Fig. 14. Vehicle's Path Length Distribution.

The modeling of traffic movement on optimistic PDES is based on SCATTER [28, 31], in which one state represents a junction and its out-going roads, as illustrated in Figure 15. The ID assigned is assigned to an LP based on the junction ID of original data, where IDs are basically random. Moreover, the traffic in the simulation scenario has a few characteristics. The simulation time resolution is defined as one second. The vehicle's movement is modeled by a simple car that accelerates and decelerates according to the vehicles in front of it. In addition, if a vehicle reaches the end of a road, it will be sent immediately to the next junction (LP). Specifically, the vehicle is inserted to the next LP's queue as one event. If an LP receives a vehicle, its state is updated, and is be used for the next vehicle moving event. Based on this traffic scenario, the simulation outputs 798,177 events in total. Since this is an optimistic simulation, the actual events that need to be handled would most certainly exceed 798,177 as some of the processed events suffer rollback. The resulting events, anti-messages, and states are stored on the 49 GB main memory.

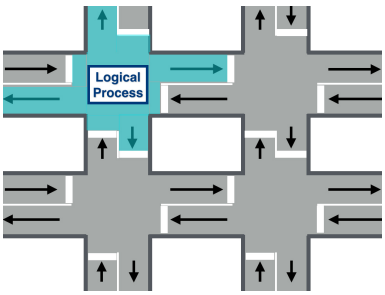
Two types of what-if scenarios are defined in the evaluation. Scenario 1 simulates what happens if a traffic control occurs at one junction. Specifically, we select one of the junctions from ID 0 to ID 1610 (1/100 of all) and reduce the speed limit to half from the start to the end. The selection is at random since LP ID is randomly initialized. This scenario is similar to a traffic accident, where roads and/or junctions are blocked. Scenario 2 simulates what happens if one of the vehicles changes its trip pattern. In this case, one of the vehicles is removed and the influence is simulated. Since the vehicle's departure time is randomly assigned, the time to change is made at random accordingly. More complex scenarios, such as route or destination changing, can be modeled by deleting the original vehicles and adding new vehicles with a different route.

5.2.1 Reprocessing Rate. Figure 16 presents the average number of resulting events. In scenario 1, where one of the states is changed, only 61,206 resulting events are simulated on average and the reprocessing rate is approximately 0.077. Note that the error bars in the figure represent the full error range of the results. In the worst case, where 297,181 resulting events are simulated, the reprocessing rate becomes 0.37.

Table 5. Traffic Simulation Configuration.

Road Map	Tokyo Center Area (Figure 12)
– # of Junctions	161,364
– # of Roads	203,363
Scenario of Tokyo's Traffic	
– Sum of Departing Vehicles	5,000
– Trips Origin/Destination	Random
– Simulation Period	3 hours
Result of Entire Simulation	
– Total Resulting Events	798,177
What-if Scenario	
– Scenario 1	Change speed limit of one state
– Scenario 2	Remove one vehicle trip

Fig. 15. Road Map and LP.



The frequency of the what-if scenarios on different numbers of resulting events is shown in Figure 17. The results show that approximately 45% of the what-if scenarios generate only 0 to 10 events, which implies that 45% of the junctions affect at most 10 events. This is due to the characteristics of the road map, where hub junctions connect large areas and minor junctions connect only local parts. Since most vehicles would pass the hub junctions, there is a significant difference in the influence between the hub and minor junctions.

In scenario 2, where one of the vehicles is modified, only 44,261 resulting events are simulated on average and the reprocessing rate is 0.055. In the worst case, the number of resulting events becomes 233,749 and the reprocessing rate becomes 0.29. In the scenario, we found that the impact of departing time is insignificant, contrary to the expectations based on the discussion in Section 3.3, and that the influence of each what-if scenario spreads over time, but not widely. This is due to the fact that vehicles move around in only a small part of the city during the 3 hours of simulation, and therefore their influence does not spread widely over the road map. Figure 17 shows the frequency of what-if scenarios for different number of events. The results indicate that 29% of the what-if scenarios generate up to 10 events, 6.8% of the what-if scenarios generate 11 to 100 events, 6.6% of them generate 101 to 1,000 events, 10% of them generate 1,001 to 10,000 events, 27% of them generate 10,001 to 100,000 events, and 20% of them generate 100,001 to 233,749 events. Similar to the analysis of Scenario 1, 29% of the vehicles move around local parts and do not affect other vehicles.

In conclusion, the exact-differential simulation has less than 0.1 reprocessing rate on average (that is, 0.077 in scenario 1 and 0.055 in scenario 2). That implies that it can reduce more than 90% of the events on average. In the worst case, the exact-differential simulation has 0.3 reprocessing rate, which implies that it can reduce more than 60% of the events.

5.2.2 Performance Evaluation. Figure 18 and Figure 19 shows the strong scalability and speedup of the entire Tokyo traffic simulation. We evaluate the elapsed time using scenario 1 (where one of the states is modified from the beginning) with the number of nodes extending from 2 to 16. Based on the previous evaluation, we select two scenarios: a worst case scenario, where the number of resulting event is 297,181, and an average case scenario, where the number of resulting events is 61,530.

According to Figure 18, the runtime performance of the exact-differential simulation outperforms the entire simulation in all cases, regardless of how many nodes are employed in the simulations. It is also worth noting that, although the performance of the exact-differential simulation in the average case scenario always outperforms the entire simulation, when more nodes are added in

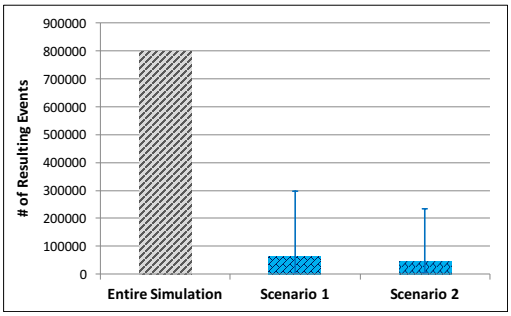


Fig. 16. Average Number of Resulting Events in Scenario 1 and Scenario 2.

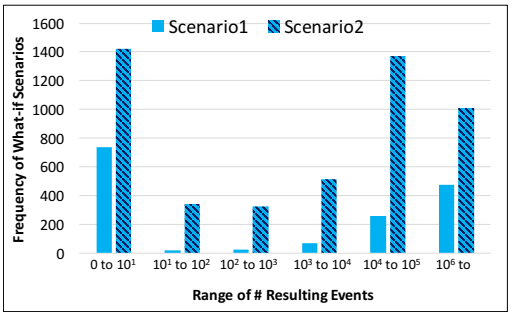


Fig. 17. Frequency of What-if Scenarios with Different Number of Resulting Events.

the simulation, its performance improvement is not as good as that of the worst scenario. This is because the problem size of the average case scenario is not large enough to fully leverage the processing power of the computing nodes. In fact, the performance is saturated at 16 nodes in the average case scenario. Furthermore, the performance saturation is also caused by rollbacks. As the number of nodes increases, the frequency of rollbacks and event reprocessing increases, rendering their overhead significant.

Figure 19 shows that the performance improvement of the entire simulation decreases as the number of nodes increases. The average case scenario has 7.27x, 5.20x, 3.64x, and 3.20x speedup when the number of nodes equals to 2, 4, 8, and 16, respectively. The worst case scenario has 2.22x, 1.80x, 1.55x, and 1.46x speed up when the number of nodes equals to 2, 4, 8, and 16, respectively. The results show similar characteristics to strong scalability, where the performance improvement is generally good but is saturated when more nodes are added to the simulation and the system becomes communication bound and susceptible to more rollbacks.

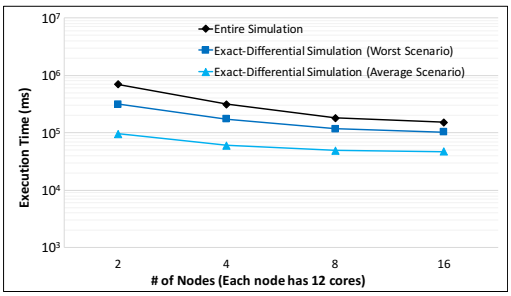


Fig. 18. Elapsed Time of Traffic Simulation.

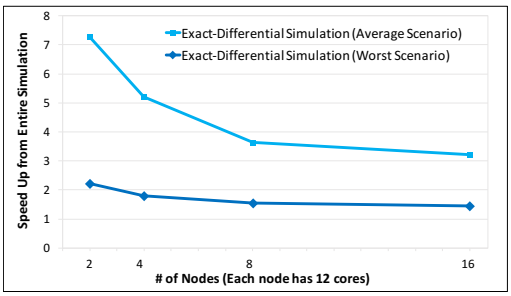


Fig. 19. Speed Up from Entire Simulation.

To summarize, the exact-differential simulation achieves a better performance than the entire simulation until a cut-off point after which the system becomes communication bound with increased synchronization overheads. It achieves 7.27 times performance improvement in the average case, and 2.22 times in the worst case.

6 RELATED WORK

An approach highly related to our work is updateable simulation [9]. This approach simulates a part of events and states in a repeating execution fashion by canceling and reprocessing events

in a similar way as optimistic PDES. The approach defines a "reuse function", which estimates the influence of reprocessing events in a repeating simulation and thus enabling the efficient reprocessing of a part of simulation. In addition to the target application domains, updateable simulation differs from the exact-differential simulation approach proposed in this paper in two major ways. First, our approach always yields the exact same results as the original simulation. On the other hand, the reuse function does not always ensure the exactly same results. Its exactness is determined by the design of the reuse function. Second, exact-differential simulation is implemented as a transparent scalable middleware layer with no need for any additional code; in updateable simulation, users need to define reuse function in addition to the application logic itself.

Cloning techniques [6, 7, 16–18, 22, 32] are also ways to reuse the simulation logs for efficient repeating simulation. Especially for traffic simulation, the paper [27] shows efficient repeating of traffic scenario based on the cloning technique. In cloning, the simulation state at some decided time is replicated, and thenceforth the simulation is branched with different parameters or scenarios. The difference from our proposal is that such techniques do not have a "differential" feature, namely, the cloning technique does not simulate a part of the entire state space but instead the entire simulation is replicated from the designated time. In our previous work we have shown how cloning can be used with our exact-differential approach in traffic simulation scenarios [14].

Differential processing approaches have been proposed for large-scale parallel and distributed computing systems other than simulation. The papers [3, 33] present differential processing approaches for MapReduce[8]. The paper [4] shows differential processing of Pregel, a general and simplified model of parallel and distributed large-scale graph processing [23]. These approaches follow a similar philosophy as the exact-differential simulation, whereby the system stores intermediate data of baseline processing and then reuses it for later processing by using some influence detections. The biggest difference between our approach and these techniques lies in the synchronization method among multiple computers, which is responsible for the influence detections. MapReduce and Pregel use the Bulk Synchronous Parallel model with barrier synchronization, while the exact-differential simulation uses asynchronous synchronization making use of the optimistic-PDES-based influence detections.

7 CONCLUSION

This paper has presented a differential processing approach for large-scale PDES called exact-differential simulation. The approach is based on optimistic PDES techniques such as rollback and restarting, and is able to identify and process only the modified parts of repeating simulations, in order to improve the overall performance of the simulation system. In addition, it can be implemented as a transparent middleware where no additional code is required for the differential processing. An extensive quantitative analysis using the PHOLD and a full Tokyo traffic simulation on 32 computing nodes with 384 cores has revealed significant performance improvement.

Our future work lies in two main directions: optimization of our algorithm and adaptation to other models. Further redundancy reduction can be achieved by the lazy processing, where the anti-messages are sent after checking the baseline results, and if the results are the same as the baseline, the subsequent reprocessing can be ignored. Moreover, we will investigate techniques to reduce memory consumption of the proposed algorithm since this is one of the limitation in the exact-differential simulation. Algorithm-level and system-level approaches can be considered. In algorithm level, we are convinced that applying existing optimization techniques for optimistic PDES (e.g. incremental state saving, reverse computing, etc.) would provide more efficient memory handling. We will also investigate new memory optimization techniques for the proposed algorithm. In system level, new hardware technologies for expanding memory space such as big memory, deep hierarchical memory, NVRAM, and SSD-based local scratch space could also be employed

to improve its performance. Memory optimization techniques from other domain, such as, Big data processing and database systems, could be adopted to deal with a large amount of simulation results.

We plan to deploy the exact-differential simulation to other large-scale PDES applications and other large-scale simulation models such as agent-based simulations. We will also investigate the integration of our approach with other synchronization protocols such as asynchronous conservative synchronization and barrier synchronization. We will also extend our idea to an "incremental" simulation, where repeating simulations are executed with a sequence of what-if scenarios, and the result of each repeating simulation is used for the subsequent execution. The results of the baseline simulation are accumulatively updated by the first what-if scenario, the second one, the third one, and so forth. The incremental simulation is useful, for example, when what-if scenarios are provided in real time at a certain interval and simulation results are required in a faster than real time fashion.

ACKNOWLEDGMENTS

The research was partly supported by Singapore Ministry of Education (MoE) Academic Research Fund, Tier 1 Grant, number RG 136/14 and by Japan Science and Technology Agency, Core Research for Evolutional Science and Technology (JST CREST).

REFERENCES

[1] P.D. Barnes, Jr., C.D. Carothers, D.R. Jefferson, and J.M. LaPre. 2013. 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)*. ACM, 327–336.

[2] D.W. Bauer Jr., C.D. Carothers, and A. Holder. 2009. 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)*. IEEE, 35–44.

[3] P. Bhatotia, A. Wieder, R. Rodrigues, U. A. Acar, and R. Pasquin. 2011. Incoop: MapReduce for Incremental Computations. In *Proceedings of the 2nd ACM Symposium on Cloud Computing (SOCC'11)*. ACM, 7:1–7:14.

[4] Z. Cai, D. Logothetis, and G. Siganos. 2012. Facilitating Real-time Graph Mining. In *Proceedings of the Fourth International Workshop on Cloud Data Management (CloudDB'12)*. ACM, 1–8.

[5] C.D. Carothers and K.S. Perumalla. 2010. On Deciding Between Conservative and Optimistic Approaches on Massively Parallel Platforms. In *Proceedings of the 2010 Winter Simulation Conference (WSC'10)*. IEEE, 678–687.

[6] D. Chen, S.J. Turner, W. Cai, B.P. Gan, and M.Y.H. Low. 2004. Incremental HLA-based Distributed Simulation Cloning. In *Proceedings of the 36th Conference on Winter Simulation (WSC'04)*. Winter Simulation Conference, 386–394.

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

[8] J. Dean and S. Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (2008), 107–113.

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

[10] R.M. Fujimoto. 1990. Performance of Time Warp under synthetic workload. In *Proceedings of the SCS Multiconference on Distributed Simulations*, Vol. 22. 23–28.

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

[12] M. Hanai. 2018. ScaleSim - General Purpose Large-Scale Parallel & Distributed Discrete Event Simulator. (2018). https://github.com/masatoshihanai/ScaleSim (Last access: 19 Nov. 2018).

[13] M. Hanai, T. Suzumura, G. Theodoropoulos, and K.S. Perumalla. 2015a. Exact-Differential Large-Scale Traffic Simulation. In *Proceedings of the 2015 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS'15)*. ACM, 271–280.

[14] M. Hanai, T. Suzumura, G. Theodoropoulos, and K.S. Perumalla. 2015b. Towards Large-Scale What-if Traffic Simulation with Exact-Differential Simulation. In *Proceedings of the 2015 Winter Simulation Conference (WSC'15)*. IEEE, 748–756.

[15] M. Hanai, T. Suzumura, A. Ventresque, and K. Shudo. 2014. An Adaptive VM Provisioning Method for Large-Scale Agent-Based Traffic Simulations on the Cloud. In *Proceedings of IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom'14)*. IEEE, 130–137.

[16] M. Hybinette. 2004. Just-in-time Cloning. In *Proceedings of the 22nd ACM/IEEE/SCS Workshop on Principles of Advanced and Distributed Simulation (PADS'04)*. IEEE, 45–51.
[17] M. Hybinette and R.M. Fujimoto. 2001. Cloning Parallel Simulations. *ACM Transactions on Modeling and Computer Simulation* 11, 4 (2001), 378–407.
[18] M. Hybinette and R.M. Fujimoto. 2002. Scalability of parallel simulation cloning. In *Proceedings of the 35th Annual Simulation Symposium (SS'02)*. IEEE, 275–282.
[19] D.R. Jefferson. 1985. Virtual Time. *ACM Transaction Programming Languages and Systems* 7, 3 (1985), 404–425.
[20] G. Karypis and V. Kumar. 1998. Multilevel k-way Partitioning Scheme for Irregular Graph. *J. Parallel and Distrib. Comput.* 48, 1 (1998), 96–129.
[21] G. Karypis and V. Kumar. 2013. METIS - A Software Package for Partitioning Unstructured Graphs, Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices-Version 5.1.0. (2013). http://glaros.dtc.umn.edu/gkhome/metis/metis/overview (Last access: 19 Nov. 2018).
[22] X. Li, W. Cai, and S.J. Turner. 2015. Cloning Agent-based Simulation on GPU. In *Proceedings of the 3rd ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS'15)*. ACM, 173–182.
[23] G. Malewicz, M.H. Austern, A.J.C Bik, J.C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. 2010. Pregel: A System for Large-scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data (SIGMOD'10)*. ACM, 135–146.
[24] E. Mikida, N. Jain, E. Gonsiorowski, C.D. Carothers, P.D. Barnes Jr., and D. Jefferson. 2016. Towards PDES in a Message-Driven Paradigm: A Preliminary Case Study Using Charm++. In *Proceedings of the 2016 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation (PADS'16)*. ACM, 99–110.
[25] T. Osogami, T. Imamichi, H. Mizuta, T. Morimura, R. Raymond, T. Suzumura, R. Takahashi, and T. Ide. 2012. *IBM Mega Traffic Simulator*. Technical Report. Technical Report RT0896, IBM Research–Tokyo.
[26] T. Osogami, T. Imamichi, H. Mizuta, T. Suzumura, and T. Ide. 2013. Toward simulating entire cities with behavioral models of traffic. *IBM Journal of Research and Development* 57, 5 (2013), 6:1–6:10.
[27] P. Pecher, M. Hunter, and R Fujimoto. 2015. Efficient Execution of Replicated Transportation Simulations with Uncertain Vehicle Trajectories. In *Proceedings of 2015 ICCS Workshop on Dynamic Data Driven Applications Systems (DDDAS'15)*. Elsevier, 2638 – 2647.
[28] K.S. Perumalla. 2006. A Systems Approach to Scalable Transportation Network Modeling. In *Proceedings of the 2006 Winter Simulation Conference (WSC'06)*. IEEE, 1500–1507.
[29] T. Suzumura and H. Kanezashi. 2013. Accelerating Large-Scale Distributed Traffic Simulation with Adaptive Synchronization Method. In *Proceedings of the 20th ITS World Congress*. ITS Japan. Paper No.4083.
[30] Tokyo Tech. 2018. TSUBAME. (2018). http://www.t3.gsic.titech.ac.jp/en (Last access: 19 Nov. 2018).
[31] S.B. Yoginath and K.S. Perumalla. 2008. 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)*. IEEE, 33–42.
[32] G. Zhang, M. Fang, M. Qian, and S. Xu. 2012. 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)*. IEEE, 73–80.
[33] Y. Zhang, S. Chen, Q. Wang, and G. Yu. 2015. i^2 MapReduce: Incremental MapReduce for Mining Evolving Big Data. *IEEE Transactions on Knowledge and Data Engineering* 27, 7 (2015), 1906–1919.