Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 680283, 15 pages
http://dx.doi.org/10.1155/2015/680283

Research Article

A Scalable GVT Estimation Algorithm for PDES: Using Lower Bound of Event-Bulk-Time

Yong Peng, Long Qin, and Quanjun Yin

College of Information Systems and Management, National University of Defense Technology, Changsha, Hunan 410073, China

Correspondence should be addressed to Yong Peng; yongpeng@nudt.edu.cn

Received 5 April 2015; Accepted 18 August 2015

Academic Editor: Chaudry Masood Khalique

Global Virtual Time computation of Parallel Discrete Event Simulation is crucial for conducting fossil collection and detecting the termination of simulation. The triggering condition of GVT computation in typical approaches is generally based on the wall-clock time or logical time intervals. However, the GVT value depends on the timestamps of events rather than the wall-clock time or logical time intervals. Therefore, it is difficult for the existing approaches to select appropriate time intervals to compute the GVT value. In this study, we propose a scalable GVT estimation algorithm based on Lower Bound of Event-Bulk-Time, which triggers the computation of the GVT value according to the number of processed events. In order to calculate the number of transient messages, our algorithm employs Event-Bulk to record the messages sent and received by Logical Processes. To eliminate the performance bottleneck, we adopt an overlapping computation approach to distribute the workload of GVT computation to all worker-threads. We compare our algorithm with the fast asynchronous GVT algorithm using PHOLD benchmark on the shared memory machine. Experimental results indicate that our algorithm has a light overhead and shows higher speedup and accuracy of GVT computation than the fast asynchronous GVT algorithm.

1. Introduction

Due to its remarkable availability, reproducibility, and cost-effectiveness, parallel simulation is a crucial approach for studying, developing, and evaluating mathematical models of systems in different domains, such as economics [1], medicine [2], human behaviors [3], and military [4]. Parallel Discrete Event Simulation (PDES) is realized through partitioning simulation problems into several distinct objects and then allocating the resulting partitions to different LPs that run concurrently. Such a parallel mechanism requires synchronization protocols to prevent causality violations or to guarantee state consistency in case of causality violations. In general, the existing synchronization protocols fall into two categories: conservative synchronization protocol and optimistic synchronization protocol (Time Warp) [5].

In optimistic synchronization protocols, Global Virtual Time (GVT) is defined as the minimum among the local virtual times of all processes and the timestamps of all messages in transit [6, 7]. A transient message is a delayed message that has been sent but has not been received. Any processed event with a timestamp smaller than the GVT value will not be rolled back. Moreover, the memory associated with it can be safely reclaimed. An efficient GVT algorithm is crucial for the optimistic time synchronization strategy to handle causality errors, reclaim memory, and detect the termination of simulation execution [8]. Numerous GVT algorithms have been devised and implemented within PDES simulators, such as GTW [9], ROSS [10], WARPED [11], ParaSol [12], and ROOT-Sim [13]. Because of its high performance, scalability, simple programming model, and application transparent parallelization, ROOT-Sim is probably the most advanced and widely used open source speculative PDES platform [14, 15].

Designs of GVT algorithms focus on either shared memory or distributed memory computers [16, 17]. Shared memory algorithm assumes that certain variables are accessible by all processors. So they perform well on symmetric multiprocessing machines [15, 18, 19]. Distributed memory algorithm does not use global variables and therefore is more scalable. For them both, time interval approach is a common solution for triggering GVT computation. However, the GVT value depends on the timestamps of events rather than wall-clock time or logical time intervals. Therefore, it is difficult for traditional algorithms to select appropriate time intervals to obtain an accurate GVT value.

In this study, we propose a scalable GVT estimation algorithm based on Lower Bound of Event-Bulk-Time (LB-EBT). In our algorithm, the start of GVT computation is triggered by the end of the Event-Bulk (EB) instead of wall-clock time or logical time intervals. Processors use EBs to record the number of events sent and received by LPs, as well as the LVT of each LP and the timestamps of events in transit. GVT computation via the EB approach can avoid message acknowledgement, which is a common solution for the problem of transient event in distributed GVT algorithms [16, 17, 20, 21]. In our algorithm, the size of Event-Bulk is negatively correlated with the accuracy of GVT computation but positively correlated with the performance of GVT computation. Through tuning the size of Event-Bulk, we can achieve a balance between the performance and accuracy of GVT computation. In addition, to make the algorithm scalable and efficient, we adopt an overlapping computation approach to distribute the workload of GVT computation to all worker-threads.

In experiments, we test our algorithm using PHOLD benchmark [22] in a prototypal test-bed on the shared memory machine. We conduct three sets of experiments to study the overhead, speedup, and accuracy of GVT computation. In addition, we compare our algorithm with a fast asynchronous GVT algorithm (FA-GVT) that triggers the GVT computation according to wall-clock time intervals. Experimental results indicate that our algorithm outperforms the FA-GVT algorithm in terms of speedup and accuracy.

2. Related Work

2.1. Distributed Memory Algorithms. In GVT algorithms for distributed memory platforms, LPs communicate their LVTs and the timestamps of events in transit with each other by exchanging messages. GVT algorithms have to tackle two problems: the LVT simultaneous reporting problem and the transient message problem [20, 21]. The LVT simultaneous reporting problem is that it is difficult to measure LVTs of all LPs running concurrently on different processors (or threads) at the same wall-clock time. The transient message problem is caused by delay messages. Processors that send or receive messages do not consider the timestamps of transient messages when they compute LVT values [23].

In the early studies of GVT algorithms, researchers often apply overlapping intervals and message acknowledgement techniques to solve these two problems above. For example, wall-clock time slices are used as LPs' intervals to compute the GVT value [23–25]. The startup of GVT computation is triggered by control messages that are broadcasted [24, 26, 27] or circulated [25, 28, 29] among LPs. The limitation of overlapping intervals is the difficulty in selecting appropriate time intervals. In case of a large time interval, the accuracy of GVT value decreases. At the same time, the memory for storing processed events and the states of LPs increases. Contrarily, in case of a small time interval, the number of control messages transmitted among LPs increases. Message acknowledgement scheme is a widely used technique for the transient message problem in early GVT algorithms [25–27]. However, this technique comes with some limitations. For example, acknowledging individual messages scheme doubles the number of messages to decrease the performance of simulations. Although some methods, such as acknowledging batches of messages [30] and piggy-back acknowledgement [21, 24], were devised to reduce the acknowledgement overhead, the acknowledgement scheme becomes complex.

Another more efficient solution than the earlier GVT algorithms is to combine consistent snapshot and the global reduction method. Consistent snapshot (or two cuts method) GVT algorithm was firstly proposed by Mattern [31]. A vector structure called *vector clock* is used to monitor the number of transient messages in his algorithm. A token carrying the vector clock is passed among LPs to construct the cuts. LVTs of LPs and timestamps of messages in transit are recorded in cuts to calculate the GVT value. Consistent snapshot method is more efficient than message acknowledgement technique in solving transient message problem. However, the size of the vector clock is dependent on the number of LPs, which hampers the scalability of the algorithm. An improved version of Mattern's algorithm was proposed by Choe and Tropper [32], which uses a scalar counter instead of vector clock to monitor the number of transient messages. Other similar algorithms with Mattern's algorithm can be found in [17, 28, 33]. Global reduction algorithms do not pass tokens among LPs but use reduction operation at the synchronization point to collect the timestamps of the transient messages and the LVTs of LPs. A global reduction GVT algorithm was proposed by Perumalla and Fujimoto [34]. Other similar GVT algorithms based on global reduction method are in [35, 36]. The main drawback of these algorithms is that all LPs have to synchronize for reduction operations.

Other GVT algorithms with different ideas are represented as follows. To compute GVT value, D'Souza et al. applied a GVT manager to collect information from all LPs [37]. Similarly, Chen and Szymanski used hierarchy GVT masters to collect GVT reports from LPs passively and then distribute the new GVT value to LPs [17]. Bauer et al. assumed that the maximum delay of messages transmitted on network is known [38]. They devised network atomic operation method to create zero-cost consistent cuts to calculate GVT. Deelman and Szymanski proposed a continuously monitored GVT algorithm that allows LPs to calculate the GVT value based on the local information [39].

2.2. Shared Memory Algorithms. In shared memory algorithms, the shared variable approach is a general method for calculating the GVT value. For example, fast asynchronous GVT algorithm (FA-GVT) proposed by Xiao et al. [19] is a typical GVT algorithm for shared memory machines. The FA-GVT algorithm triggers GVT computation according to wall-clock time intervals and relies on shared variables that are accessed by worker-threads to compute the GVT value under the control of a critical section. Due to the existence of the critical section, FA-GVT algorithm is neither scalable nor

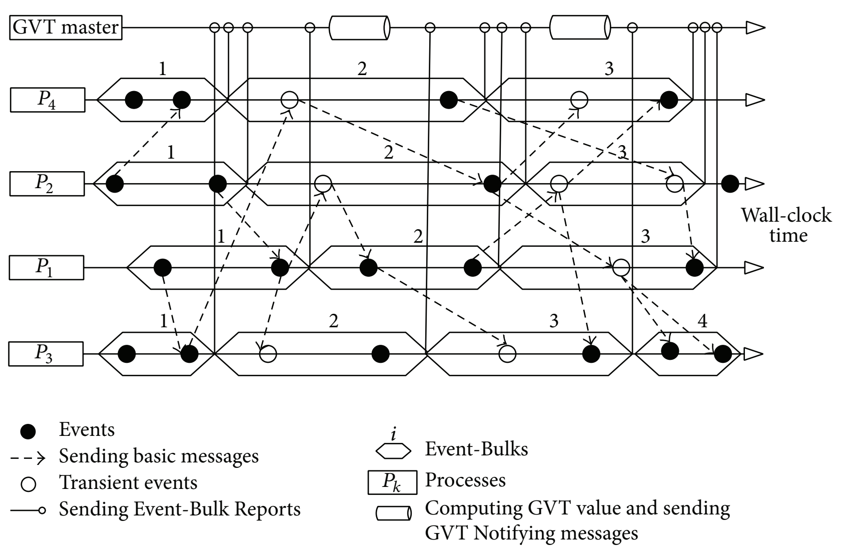


Figure 1: The principle of LB-EBT GVT algorithm.

wait-free for LPs. Fujimoto and Hybinette proposed a similar approach [18].

In order to improve the scalability of the algorithm proposed by Fujimoto and Hybinette, Pellegrini and Quaglia proposed a wait-free GVT computation algorithm on shared memory machines. Pellegrini and Quaglia used memory atomic operations on certain shared counters/flags to track the advancement of worker-threads within the different phases of GVT computation [15]. Experiments reveal that Pellegrini's approach is more efficient and more scalable than the algorithm in [18]. Rizvi et al. proposed a GVT algorithm based on Unacknowledged Message List (UML) scheme [21]. They assumed that certain variables are accessible by all processors. In addition, Rizvi et al. used a dedicated controller LP to monitor the GVT computation process.

3. Design of LB-EBT GVT Algorithm

3.1. Algorithm Description

3.1.1. Principle of Our Algorithm. We assume that there is a GVT master process in our algorithm, which is in charge of computing the GVT value and reporting the new GVT value to processors. Figure 1 gives an example to illustrate the principle of our algorithm. The process of a round of GVT computation in our algorithm consists of two phases.

In the first phase, each processor constructs Event-Bulks (EBs) and sends the data of EBs to the GVT master via EB Report messages. At the startup of a round of GVT computation, each processor maintains a scale counter to record the number of event/antievent processed by it. Once the value of the counter reaches the size of EB (the size of EB is two in this example shown in Figure 1), the processor ends current EB and starts to construct the subsequent EB. At the same time, the processor sends an EB Report message to GVT master. EB Report message carries the data that is necessary for computing the GVT value. Therefore, our algorithm uses EBs (instead of wall-clock or logical time intervals) to trigger the computation of GVT value. The size of EB of a processor at a round of GVT computation is only determined by the processor according to the requirements of memory management, the accuracy of GVT value, and the frequency of committing events.

In the second phase, GVT master computes the GVT value and sends the new GVT value to processors. During the simulation execution, the GVT master is listening to EB Report messages from processors. Once receiving EB Report messages from all processors, GVT master performs GVT computation and then sends the new GVT value to all processors via GVT Notifying messages.

Due to all workload of GVT computation being processed in GVT master, GVT master becomes the performance bottleneck when there is a larger amount of processors. The algorithm mentioned above is a centralized algorithm and is not scalable [17]. To overcome this drawback, we remove the GVT master and distribute the workload of GVT computation to all processors via a communication topology of processors. The communication topology acts as a route to pass EB Report messages and GVT Notifying messages among processors.

The communication topology of processors is represented as a complete k-ary tree [40]. Parameter k is an integer and subject to $0 < k < N$ (N is the number of processors). In graph theory, k-ary tree is a rooted tree in which each node has k children at most. A complete k-ary tree is a k-ary tree, which is maximally space efficient. It must be completely filled on every level except the last level [40]. The communication topology of our algorithm is represented as $T(N, k)$ and depicted in Figure 2.

After employing the communication topology of processors, a round of GVT computation in our algorithm is extended to three phases. In the first phase, whenever any processor (including leaf processors) ends its EB, it sends control messages to all leaf processor(s) to request them to

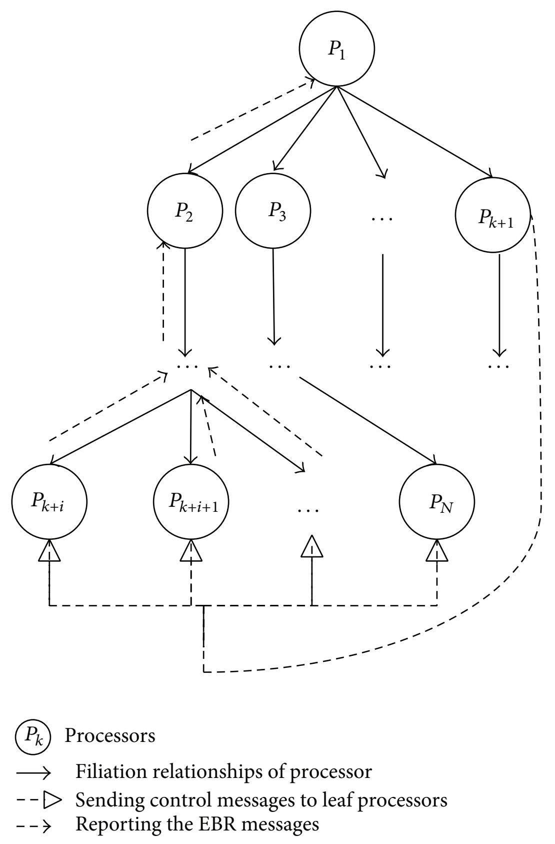

P_1
P_2
P_3
P_{k+1}
P_{k+i}
P_{k+i+1}
P_N
P_k Processors
Filiation relationships of processor
Sending control messages to leaf processors
Reporting the EBR messages

FIGURE 2: Communication topology of LB-EBT GVT algorithm.

report EB Report messages (rather than sending an EB Report message to GVT master as mentioned before). Once receiving a control message, a leaf processor has two choices according to the index of its current EB. In case that the index is smaller than the index in the control message, the leaf processor ends current EB and sends an EB Report message to its parent processor. Otherwise, the leaf processor just discards the control message. In the communication topology, EB Report messages in any GVT computation round are sent by the leaf processors firstly.

In the second phase, a nonleaf processor records its EB and starts to construct the subsequent EB at the end of current EB. The processor does not send EB Report messages to its parent processor until it receives EB Report message(s) from all its children processors. However, once receiving EB Report message(s) from all its children processors and even the number of processed events is less than the size of EB, the processor ends its current EB. At the same time, the processor handles the data of its current EB and the data of all EB Report messages from all its children processors and then sends the result to its parent processor via an EB Report message.

In the final phase, when the root processor receives EB Report messages for one round of GVT computation from all its children processor, it performs GVT computation and then sends the new GVT value to its children processor via GVT Notifying messages. Once a processor receives a GVT Notifying message, it updates its local GVT value and then sends GVT Notifying messages to its children processors. GVT Notifying messages are sent to descendant processors until all leaf processors get the new GVT value.

By using the communication topology of processors, the workload of GVT computation is distributed to all processor. Our algorithm does not use a GVT monitor or GVT master to perform GVT calculation like other algorithms [17, 21, 32, 37]. Each processor maintains a counter indicating the index of the current EB and increases the counter at the end of the EB. Once a processor ends an EB, it then starts subsequent EB immediately and does not need to be blocked or synchronized. The startup of subsequent round of GVT computation can be triggered before the completion of the previous round. While a processor performs GVT computation for one round, other processors can process *EBR* messages for another GVT computation round. Theoretically, a processor can process any number of basic messages and construct any number of EBs before a GVT computation round finishes. According to the indices of *EBR* messages and the communication topology, the root processor can calculate the number of transient messages in each round and knows which round of GVT computation is completed. This solution is suitable for large-scale simulation systems and can reduce the overhead of GVT computation.

In addition, EB Report messages and basic event/anti-event messages are labeled with the index of EB before they are sent to destination processors. Processors can correctly process EB Report messages and even EB Report messages or event/antievent messages are not transferred in First-In-First-Out (FIFO) manner. In other words, our approach is not dependent on the FIFO communication channels, which is the precondition of some literature algorithms, such as the Time Quantum GVT algorithm [17] and the Hypercube GVT algorithm [26].

3.1.2. Design Detail. We define some concepts before describing the algorithm in detail. In our algorithm, the messages used to carry remote events/antievents are called basic messages formalized in Table 1. In addition, EB messages and the GVT messages (including EB Report messages, GVT Notifying messages, and control messages) are also defined in Table 1. According to the definition of EB, we define the notion of Event-Bulk-Time (EBT). The EBT of an EB consists of the *LVT* and the *MTS* of EB. The Lower Bound of EBT is therefore the minimum of these two parts.

We define three operators for a processor node P_i in the communication topology: $p(i)$, $c(i)$, and $isLeaf(i)$, to describe our algorithm in detail. Operator $p(i)$ described by (1) is used to obtain the parent processor node of P_i:

$$p(i) = \left\{ j \mid j = \left\lceil \frac{(i-2)}{k} + 1 \right\rceil,\ j > 0 \right\}, \tag{1}$$

where i and j are the identifiers of child processor and parent processor, respectively. k is the factor of $T(N, k)$. $\lceil x \rceil$ is the integer part of x.

Operator $c(i)$ in (2) obtains the set of children nodes of a processor P_i:

$$\begin{aligned} c(i) &\\ &= \left\{ j \mid j = i * k + 1 - m,\ m = 0, 1, \ldots, k-1,\ j \leq N \right\}. \end{aligned} \tag{2}$$

Table 1: Formations of messages.

Name	Formalization	Explanation
Basic message	$BM(ebi, ts, ed)$	*ebi* is the index of the message, *ts* is the timestamp of the event, and *ed* is the data of positive event or antievent.
Event-Bulk	$EB(ebi, Size, LVT, MTS, SntC, RecvMap)$	*ebi* is the index of the EB, *Size* is the size of EB, *LVT* is the local virtual time of the processor, *MTS* is the minimum timestamp of the events sent by the processor during the EB, *SntC* is the number of events sent by the processor during the EB, and *RecvMap* is used to record the number of remote events received by the processor during the EB. *RecvMap* is a map whose key field is the index of the in-coming remote events and value field is the number of in-coming remote events.
Event-Bulk Report message	$EBR(i, ebi, LVT, MTS, TMMAP)$	*i* is the identifier of the processor that sends the message; the definitions of ebi, *LVT*, and *MTS* are the same as those in EB; *TMMAP* is a data structure similar to *RecvMap* of EB and is used to record the number of events that may be in transit.
GVT Notifying message	$GN(ri, gvt)$	*ri* is the index of GVT computation round and *gvt* is the new GVT value.
Control message	$CM(ebi)$	*ebi* is the index of the EB wanted to be reported.

Node P_j is the first child of P_i in case of $m = k - 1$. If having no first child, a processor cannot have any children and becomes a leaf node. Whether processor P_i is a leaf node processor can be determined by this condition. The operator $isLeaf(i)$ is described as

$$isLeaf\ (i) = (i - 1) * k + 2 > N \ ?\ \text{false:true}. \quad (3)$$

Beyond these three operators, the set of leaf nodes is described by the operator $leafP(N, k)$ according to the following equation:

$$leafP\ (N, k) = \begin{cases} \{N\}, & k = 1 \\ \{j \mid a + b \le j \le N\}, & k \ne 1, \end{cases} \quad (4)$$

where $a = (k^l - 1)/(k - 1) - k^{l-1} + 1$ is the ID of the first processor node on the deepest nonleaf level of $T(N, k)$ and $b = \lceil (N-(k^l-1)/(k-1))/(k-1) \rceil + 1$ is the number of nonleaf processor nodes on the deepest nonleaf level of $T(N, k)$, with $l = \lceil \log_k^{N(k-1)+1} \rceil$. $a+b$ is the identifier of the first leaf processor node.

All operators described above are only executed at the startup of the simulation to construct the communication topology and will not be executed any more during the simulation execution unless the topology is changed.

The pseudocode of LB-EBT GVT algorithm is given in Algorithm 1. At the startup of simulation, the algorithm initializes parameters including the number of worker-threads, the number of LPs, the *Stop Time* of a simulation, and the size of EB. Then, the algorithm records the number of processed basic messages and updates *LVT* (see line S4). Besides, the algorithm tracks the number of remote basic messages that are sent and received (see line S6 and line S10). The minimum timestamp of remote basic messages is also recorded in *MTSMAP* (see line S9). Once the number of processed basic messages reaches the size of current EB or the *LVT* is greater than the *Stop Time* of the simulation, the processor ends the current EB and then requests the leaf processors to report their *EBR* messages (see lines S13–S17).

Once receiving an *EBR* message, a processor records the number of transient message(s) carried by the *EBR* message and calculates the minimum of *LVT*s and the minimum timestamp of transient messages (see lines S24–S28). If all children have reported *EBR* messages, nonroot processors will send an *EBR* message to its parent processor (see lines S29–S39).

Once the root processor processed all *EBR* messages from its children processors for one round of GVT computation, the minimum of each LB-EBT is stored in *MTSMAP* or *LVTMAP* of the root processor (see lines S32-S33). And then the root processor can obtain the number of transient events through calculating the data in *TMCMAP*. If there are transient events, GVT value is determined by the minimum value of all LB-EBT. Otherwise, the GVT value is the minimum of *LVTs* of the EBs in one round of GVT computation (see line S36). Finally, the new GVT value will be calculated by the root processor and then sent to its children processors via *GN* messages.

3.2. Example. We describe the process of GVT computation in our algorithm step by step through an example shown in Figure 3. We assume that there are four processors in a system. Factor k in the communication topology of processor $T(N, k)$ is two. Processor P_1 is the root processor node that has two children processor nodes: P_2 and P_3. Processor P_4 is the child of P_2. Then, P_3 and P_4 are leaf processors.

The process of the ith round of GVT computation shown in Figure 3 is described below.

Step 1. All processors begin to construct their ith EBs.

```
// LB-EBT GVT algorithm for the $processor_i$
S0  $LVT = 0$; $GVT = 0$; $ebi = 0$; $eventProcessed = 0$; eventSnt = 0; $MTS = \infty$; $TMMAP$ is empty; $EBMap$ is
    empty; $TMCMAP[ebi] = 0$; $MTSMAP[ebi] = \infty$; $LVTMAP[ebi] = \infty$; /* initialization of processor $p_i^*$/
S1  Configuring the schedules, generating the communication topology, and generating LPs.
S2      while ($GVT < StopTime$)
S3          for any basic messages $BM(j, ts, ed)$
S4              process($BM.ed$); $eventProcessed$++; $LVT = BE.ts$;
S5              if $BM(j, ts, ed)$ is a remote basic message from $processor_j$
S6                  $TMMAPTMCMAP[BM.j] -= 1$; /* record the count of received remote messages */
S7              end if
S8              for any remote basic message $BM(ebi, ts, ed)$ sent
S9                  $MTS = \min(MTS, BE.ts)$;
S10                 $eventSnt$++;
S11             end for
S12         end for
S13         if ($eventProcessed >= EB.Size \| LVT > StopTime$) /* finish an EB */
S14             $TMMAP[ebi] += eventSnt$; record the current EB to $EBMap$;
S15             $eventSnt = 0$; $eventProcessed = 0$; $ebi$++;
S16             send control messages to $leafP(N, k)$/* request leaf P to send an EBR messages */
S17         end if
S18         for any control message
S19             if $LeafP(i)$ and index of control message is not less than $ebi$
S20                 send $EB$ report message $EBR(i, ebi, LVT, MTS, TMMAP)$ to $p(i)$
S21             end if
S22         end for
S23         for any EB Report message $EBR(j, ebi, LVT, MTS, TMMAP)$
S24             for each key $k$ in $EBR.TMMAP$
S25                 $TMCMAP[k] += EBR.TMMAP[k]$;
S26             end for
S27             $MTSMAP[EBR.ebi] = \min(MTSMAP[EBR.ebi], EBR.MTS)$;
S28             $LVTMAP[EBR.ebi] = \min(LVTMAP[EBR.ebi], EBR.LVT)$;
S29             if all processor in $c(i)$ have reported $EBR$ messages
S30                 record the current EB to $EBMap$;
S31                 $TMCMAP[ebi] += EBMap[ebi].SntC$;
S32                 $MTSMAP[EBR.ebi] = \min(MTSMAP[EBR.ebi], EBMap[ebi].MTS)$;
S33                 $LVTMAP[ebi] = \min(LVTMAP[ebi], EBMap[ebi].LVT)$;
S34                 $eventSnt = 0$; $eventProcessed = 0$; $ebi$++;
S35                 if $Processor_i$ is the root processor /* calculate the new value of GVT */
S36                     $GVT = \min_{k<=ebi}(\min_k(MTSMAP[k] | TMCMAP[k] > 0, LVTMAP[k]))$;
S37                     send $GN(ebi, gvt)$ to $c(i)$; /* send GVT notifying message to its children */
S38                 else
S39                     send $EBR(i, ebi, LVT, MTS, TMMAP)$ to $p(i)$; /* send EB report to parent */
S40                 end if
S41             end if
S42         end for
S43         for any GVT notifying message $GN(j, gvt)$
S44             $GVT = GN.gvt$;    /* update GVT */
S45             if not$LeafP(i)$
S46                 send $GN(j, gvt)$ to $c(i)$; /* send GVT notifying message to its children */
S47             end if
S48         end for
S49     end while
```

Algorithm 1: LB-EBT GVT algorithm.

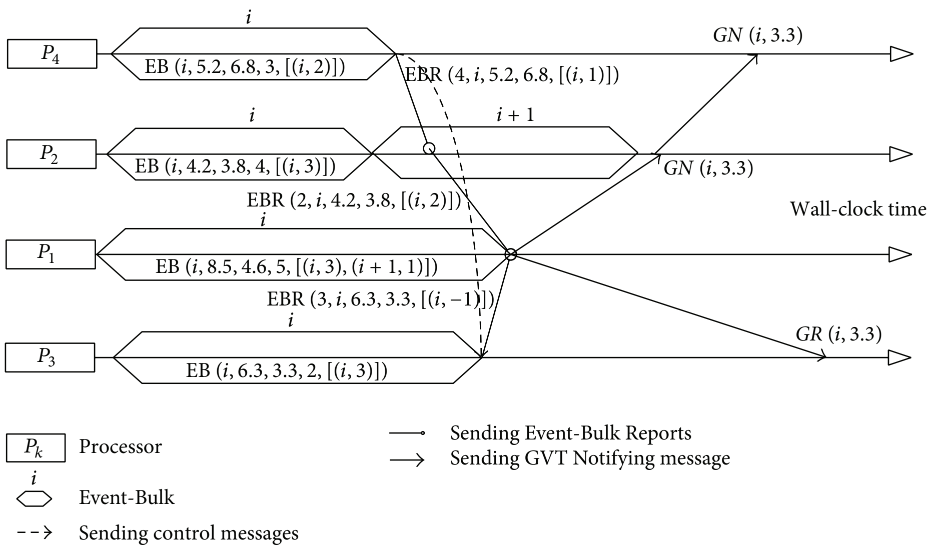

P4
P2
P1
P3
i
i + 1
EB (i, 5.2, 6.8, 3, [(i, 2)])
EBR (4, i, 5.2, 6.8, [(i, 1)])
EB (i, 4.2, 3.8, 4, [(i, 3)])
EBR (2, i, 4.2, 3.8, [(i, 2)])
EB (i, 8.5, 4.6, 5, [(i, 3), (i + 1, 1)])
EBR (3, i, 6.3, 3.3, [(i, −1)])
EB (i, 6.3, 3.3, 2, [(i, 3)])
GN (i, 3.3)
GN (i, 3.3)
GR (i, 3.3)
Wall-clock time
Pk Processor
Event-Bulk
Sending control messages
Sending Event-Bulk Reports
Sending GVT Notifying message

Figure 3: GVT calculation process of LB-EBT GVT algorithm.

Step 2. Before sending a basic message, a processor sets the index of the basic message as the index of current EB. Once receiving a basic message, the processor adds a key-value pair record to *RecvMap* of its current EB. The key is the index of the basic message. The number of basic messages stored in the record of *RecvMap* of the current EB increases by one.

Step 3. Once the number of processed events in the ith EB of a processor reaches the size of the ith EB, the processor ends the ith EB. At the same time, the processor sends control messages to ask the leaf processors to report their *EBR* messages. As shown in Figure 3, processor P_4 ends its ith EB and then sends a control message to another leaf processor P_3. Since processor P_4 is a leaf processor, it sends an *EBR* message, namely, $EBR(4, i, 5.2, 6.8, [(i, 1)])$, to its parent processor P_2 at the end of its ith EB. The *LVT* of P_4 is 5.2 and three basic messages have been sent during the ith EB. The minimum timestamp of these three basic messages is 6.8. P_4 receives two basic messages during the EB. These two messages are sent during the EB. The number of transient messages known to a processor during the ith EB is the number of the sent messages subtracted from the number of the received messages by the processor. Therefore, the number of transient messages known to P_4 is one.

Step 4. A processor does not send an *EBR* message to its parent until receiving *EBR* messages from all its children (e.g., processor P_2). Each processor maintains three map data structures (*TMCMAP*, *MTSMAP*, and *LVTMAP*) to record the data of *EBR* messages from its children processors. The key field of these three maps records the indices of *EBR* messages. Once a parent processor receives *EBR* messages from all its children, it updates its *TMCMAP*, *MTSMAP*, and *LVTMAP* with *EBR* messages according to the following rules (an *EBR* message from a child processor is denoted as $EBR(c, i)$).

Rule 1. For each key k in $EBR(c, i).RecvMap$,

$$TMCMAP[k] += EBR(c, i).RecvMap[k]. \quad (5)$$

Rule 2. If $(MTSMAP[i] > EBR(c, i).MTS)$,

$$MTSMAP[i] = EBR(c, i).MTS. \quad (6)$$

Rule 3. If $(LVTMAP[i] > EBR(c, i).LVT)$,

$$LVTMAP[i] = EBR(c, i).LVT. \quad (7)$$

For example, after processing the *EBR* message from P_4 and its ith EB, $TMCMAP[i]$ of processor P_2 is 2. $MTSMAP[i]$ is 3.8. Similarly, $LVTMAP[i]$ of P_2 is 4.2. The *EBR* message of P_2 sent to P_1 is therefore $EBR(2, i, 4.2, 3.8, [(i, 2)])$.

Step 5. When the root processor P_1 has received all the ith *EBR* messages from all of its children processors during the ith round of GVT computation, the new GVT value can be calculated from *TMCMAP*, *MTSMAP*, and *LVTMAP* of the root processor according to the following equation:

$$gvt = \min_{k \le i} \left(\min_{k} \left(\left(MTSMAP[k] \mid TMCMAP[k] > 0 \right), LVTMAP[k] \right) \right). \quad (8)$$

For the example shown in Figure 3, the *TMCMAP* of P_1 is $[(i, 3), (i+1, 1)]$ and the *MTSMAP* is $[(i, 3.3)]$. The *LVTMAP* of P_1 is $[(i, 4.2)]$. These three maps indicate that all remote messages sent before the ith round of GVT computation have been received and processed. There are three transient messages in the ith round of GVT computation. The new GVT value is the smaller one between $MTSMAP[i]$ and $LVTMAP[i]$. Therefore, the GVT value is 3.3 in the ith round of GVT computation.

Step 6. At the end of the ith round of GVT computation, the root processor sends *GN* messages to its children processors to notify them of the new GVT value. Then, *GN* messages are sent to its descendant processors until all leaf processors get the new GVT value.

3.3. Algorithm Implementation. We implemented our algorithm on a multicore shared memory machine for prototypical test. As the multicore machine is prevalent, multithreading techniques are commonly used in the speculative PDES [19, 41, 42].

Our algorithm is implemented within a simulation engine that consists of a number of *schedulers*. Each scheduler is bound to a worker-thread and in charge of carrying out the function of a processor. At the initial phase of a simulation, the simulation engine firstly uses the pthread multithreaded library to configure the number of schedulers according to the available idle CPU cores of the machine and then generates the communication topology for schedulers according to their identifiers. Finally, the engine creates all LPs and averagely dispatches them to the schedulers.

A scheduler maintains two queues for storing and processing events. One of these two queues, called *msg_cached_queue*, is a concurrent queue which is used to temporarily store basic messages sent by the LPs hosted on some other schedulers. The other queue is called *event_priority_queue*, which is a priority queue. A scheduler uses the *event_priority_queue* to schedule LPs for processing event according to Lowest-Timestamp-First scheme [43]. In case a LP sends a remote event (via the basic message) to the LP hosted by another scheduler, the sender LP directly inserts the event into the *msg_cached_queue* of the scheduler hosting the receiver LP. After a scheduler dispatching a given number (the size of EB) of events or discovering that the *event_priority_queue* is empty, it will pop events from *msg_cached_queue* and push them into the *event_priority_queue* for processing.

During simulation execution, a scheduler records the data (the number of events sent/received and minimum timestamp of events sent in one GVT computation round) to its current EB. In addition, a scheduler maintains three map data structures (*TMCMAP*, *MTSMAP*, and *LVTMAP*) to record the data of *EBR* messages from its children schedulers. Once a scheduler is notified with the new GVT value by its parent scheduler, it forces all LPs hosted on it to perform fossil collection via calling the *commit* function of each LP.

Although there are some more simple GVT algorithms than ours on shared memory machines, they rely on the observability property of Time Warp systems [15, 18, 19]. The observability property of Time Warp systems requires the *msg_cached_queue* being observable to the scheduler, which is impossible for Time Warp systems on distributed memory machines. Our algorithm does not rely on this property, which makes our algorithm more suitable for distributed memory machines. Therefore, although our algorithm is carried out on shared memory machines due to the convenience of programming, debugging, and testing, it can be migrated to distributed memory systems with trivial modifications.

Table 2: Description of experimental parameters.

M	The number of LPs in the PHOLD model.
γ	The remote event ratio of the PHOLD model.
λ	The mean time delay of an event in PHOLD model, which is set to 1 second in the simulation time.
N	The number of worker-threads (processor core, the worker-thread, and the processor represent the same meaning in this paper) used in experiments. Each scheduler of LB-EBT simulator is mapped onto a worker-thread during the whole simulation execution.
S	The size of the EB, which is identical for all schedulers in LB-EBT GVT algorithm.
k	The branch factor of the communication topology in our algorithm.
Δt	Δt is the check point period of FA-GVT. The check point period of LB-EBT GVT algorithm is not constant and depends on the size of EB.

4. Experimental Results

4.1. Experiment Configuration. We use PHOLD benchmark to conduct three experiments on shared memory machines to compare FA-GVT algorithm as the baseline algorithm with our LB-EBT GVT algorithm. In first experiment, we study the overhead of our algorithm and optimize parameter k of communication topology $T(N,k)$. In the second experiment, we compare the speedup and scalability of our LB-EBT GVT algorithm with those of FA-GVT. In the final experiment, we evaluate the accuracy of our algorithm and compare the results with FA-GVT algorithm. The number of initial events per LP is set to 16 in all experiments. The parameter setting in experiments is depicted in Table 2.

All experiments are run on a DELL R610 server with 24 GB RAM and two Intel Xeon processors running on 2.4 GHz. Each processor has six cores and 12 threads (for a total of 24 threads) that share a L1 cache with 2 MB and a L2 cache with 12 MB. The operation system is Microsoft Windows 2008 server R2. All algorithms are programmed in C++ language and compiled with Microsoft Visual Studio 2008.

4.2. Parameter Optimization. Parameter k of communication topology $T(N,k)$ is a key factor that influences the time span and the number of GVT messages of a round of GVT computation in our algorithm. We try to explore an optimized value of parameter k for different values of parameter N in

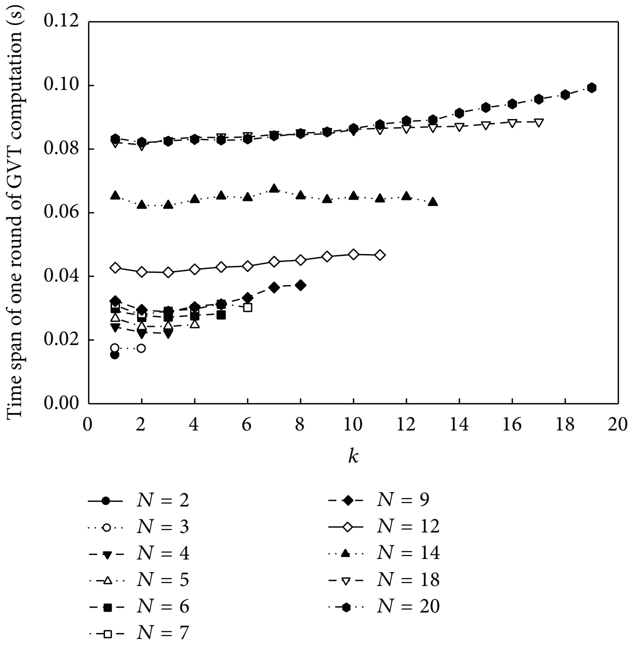


Figure 4: Time span of one round of GVT computation ($M = 396900$, $\lambda = 1$, $\gamma = 25\%$, and $S = 512$).

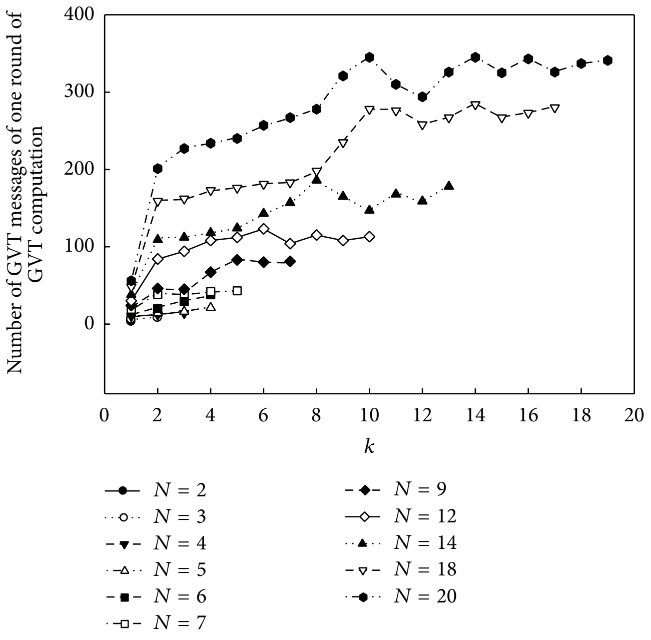


Figure 5: GVT messages of one round of GVT computation with k ($M = 396900$, $\lambda = 1$, $\gamma = 25\%$, and $S = 512$).

experiments. In addition, the number of LPs M, mean delay time λ, and the remote event ratio γ in PHOLD model are set as 396900 (the largest simulation), 1 second, and 25%, respectively.

We firstly study the relationship between k and the time span TS of one round of GVT computation. TS is the wall-clock time interval beginning at the time point when the first leaf scheduler reports the *EBR* message and ending at the time point when the last scheduler receives the *GN* message. Results shown in Figure 4 depict the evolution of TS with k and N. It indicates that TS gradually drops down in the value interval [1, 3] of parameter k but rises up from 4 to $N - 1$. The plots reach the bottom in case that k equals two or three. The bottom of plots demotes the minimum overhead of algorithms. Overall, it is shown in Figure 4 that TS rises up when k increases.

The relationship between the number of GVT messages (including *EBR* messages, *GN* messages, and control messages) and k is illustrated in Figure 5. When k is larger than 4, the number of GVT messages increases quickly. Because the number of *EBR* messages and *GN* messages remains the same for different k in one round of GVT computation, we infer that the control message contributes to the increase of the number of GVT messages.

The relationship between the time span of one round of GVT computation and the size of EB is described in Figure 6.

It is shown in Figure 6 that the time span of one round of GVT computation increases along with the increase of the size of EB. The curves denote that TS rises up slowly when the number of schedulers is small but goes up quickly when the number of worker-threads is larger than nine. This phenomenon can be explained in twofold. On the one hand, the increase of the number of worker-threads results in more

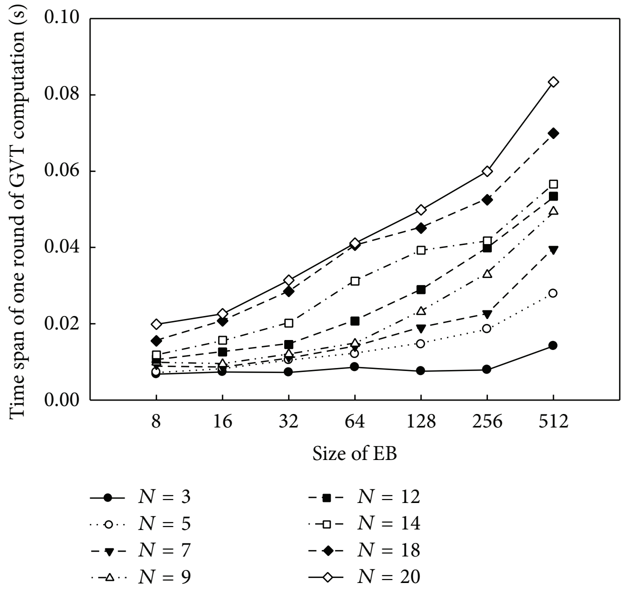


Figure 6: Time span of one round of GVT computation with size of EB ($M = 396900$, $\lambda = 1$, $\gamma = 25\%$, and $k = 2$).

GVT messages to be processed. On the other hand, when the size of EB becomes larger, the construction of an EB requests more basic messages being processed. However, the size of EB is not linear to the time span. For example, when the sizes of EB are 128 and 256, the corresponding time spans of one round of GVT computation for 20 worker-threads are 0.0499 seconds and 0.0600 seconds in wall-clock time, respectively. The time span only increases by 20.4%. It is concluded that more GVT calculation of worker-threads is overlapped when the size of EB increases.

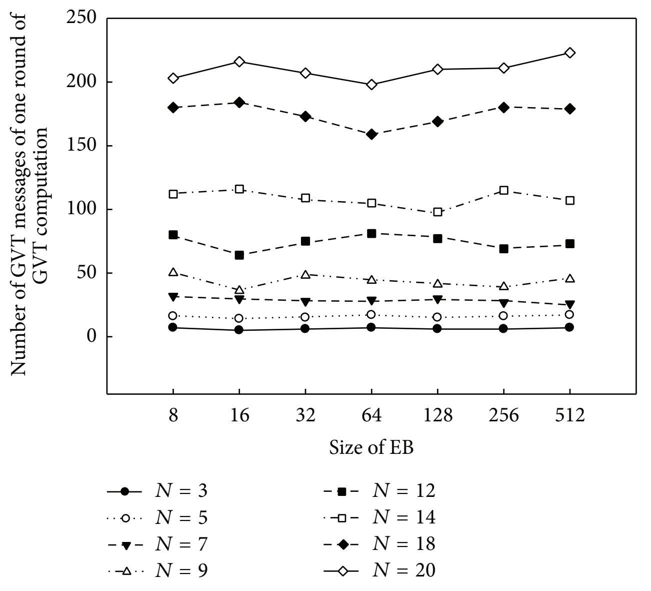


Figure 7: Number of GVT messages with size of EB ($M = 396900$, $\lambda = 1$, $\gamma = 25\%$, and $k = 2$).

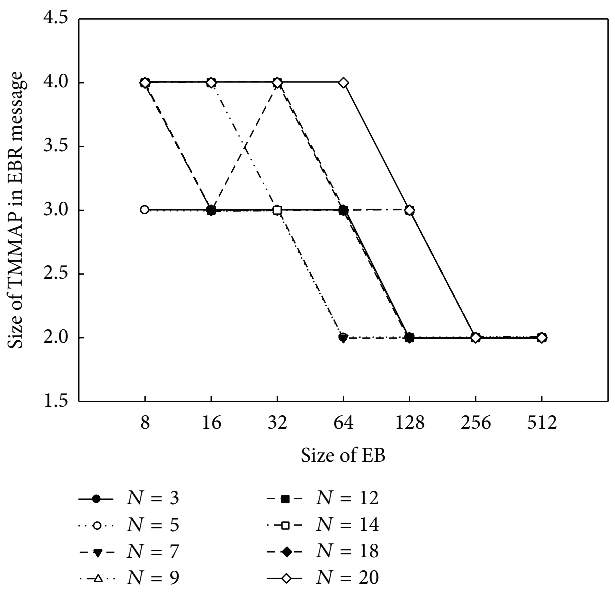


Figure 8: Size of TMMAP in *EBR* message ($M = 396900$, $\lambda = 1$, $\gamma = 25\%$, and $k = 2$).

The relationship between the number of GVT message and the size of EB is shown in Figure 7. We find that the number of GVT messages slightly fluctuates with the increase of the size of EB. This indicates that the number of basic messages is the main impact factor on the increase of time span of one round of GVT computation rather than the number of GVT messages.

Beyond the number of GVT messages, the size of *EBR* messages can also introduce overhead to our algorithm. The size of *EBR* messages is not constant and increases in case a processor receives basic messages with different indices during an EB. The reason is that the number of basic messages received by a processor is recorded with an index-number pair in *TMMAP* of *EBR* message. The evolution of the size of *TMMAP* in *EBR* messages with different sizes of EB is described in Figure 8. The size of *TMMAP* is the maximum value obtained by ten tests. We observe that the size of *TMMAP* is directly proportional to the size of EB but is inversely proportional to the number of worker-threads. In addition, the maximum size of *TMMAP* is smaller than four in all cases, which means that the overhead introduced by *TMMAP* is small enough and can be considered as a constant.

Overall, we discover in this experiment that the overhead of LB-EBT GVT algorithm is minimum when k is equal to two. We will use this value for the following experiments. The size of *TMMAP* of *EBR* message is less than four in all cases (different number of LPs and different number of worker-threads), which can be considered as a constant.

4.3. Speedup and Scalability. Speedup and scalability are two significant metrics to evaluate the performance of GVT algorithms, which can be measured in terms of the committed event rate.

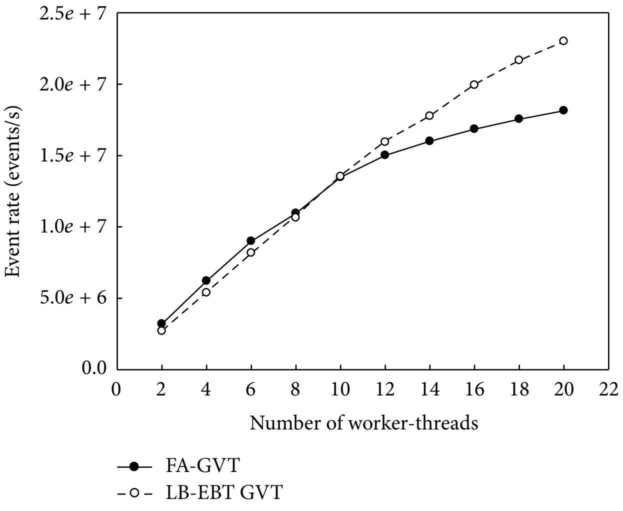


Figure 9: Results of the committed event rate ($M = 396900$, $S = 256$, $\lambda = 1$, $\Delta t = 0.1$, and $k = 2$ for $N > 2$).

In this experiment, we compare the scalability of our algorithm with FA-GVT. The results are shown in Figure 9, where all samples have been obtained as the average over five runs. Different pseudorandom seeds are used in each run. However, the same seed is used for the corresponding runs with two different GVT algorithms. Given that the large amount of event can be processed per wall-clock time unit, the check point period Δt of FA-GVT is set to 0.1 seconds in wall-clock time.

In Figure 9, we find that committed event rate is slightly lower in our algorithm than in FA-GVT algorithm in case that the number of worker-threads is smaller than ten. This phenomenon is explained as the cost of GVT messages and the cost of roll-back in our algorithm are larger than

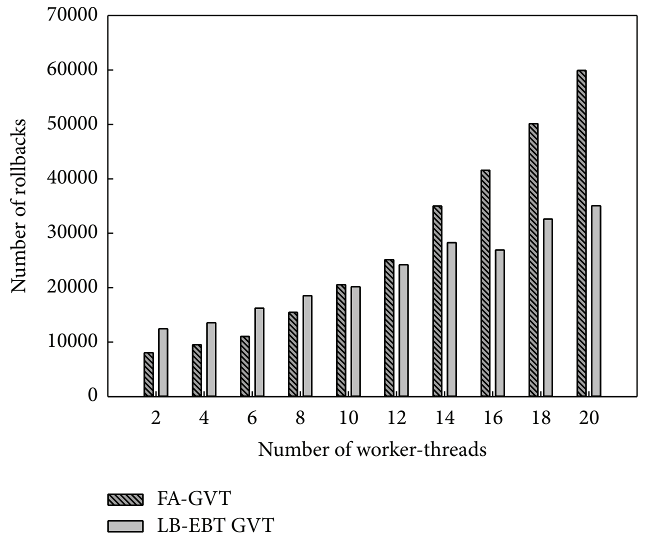


Figure 10: Results of rollbacks ($M = 396900$, $S = 256$, $\lambda = 1$, $\Delta t = 0.1$, and $k = 2$ for $N > 2$).

those of FA-GVT algorithm. The critical section in FA-GVT algorithm does not become the performance bottleneck yet. Moreover, with the increase of the number of worker-threads, the committed event rate in our algorithm slightly decreases but significantly decreases in FA-GVT algorithm.

The number of rollbacks of our algorithm and FA-GVT is shown in Figure 10. When the number of worker-threads is smaller than ten, the number of rollbacks of our algorithm is larger than that of FA-GVT. The reason causing this phenomenon is that the overhead in our algorithm is greater than that of FA-GVT. With the increase of the number of worker-threads, the number of rollbacks slightly increases in our algorithm but sharply increases in FA-GVT. FA-GVT is not a wait-free algorithm. So worker-threads in FA-GVT have to wait for each other to access the critical section to compute the GVT value. This leads to *stranger* events that result in rollbacks. However, worker-threads in our algorithm are not blocked for computing GVT value. The probability of *stranger* events caused by GVT computation our algorithm is lower than that of FA-GVT.

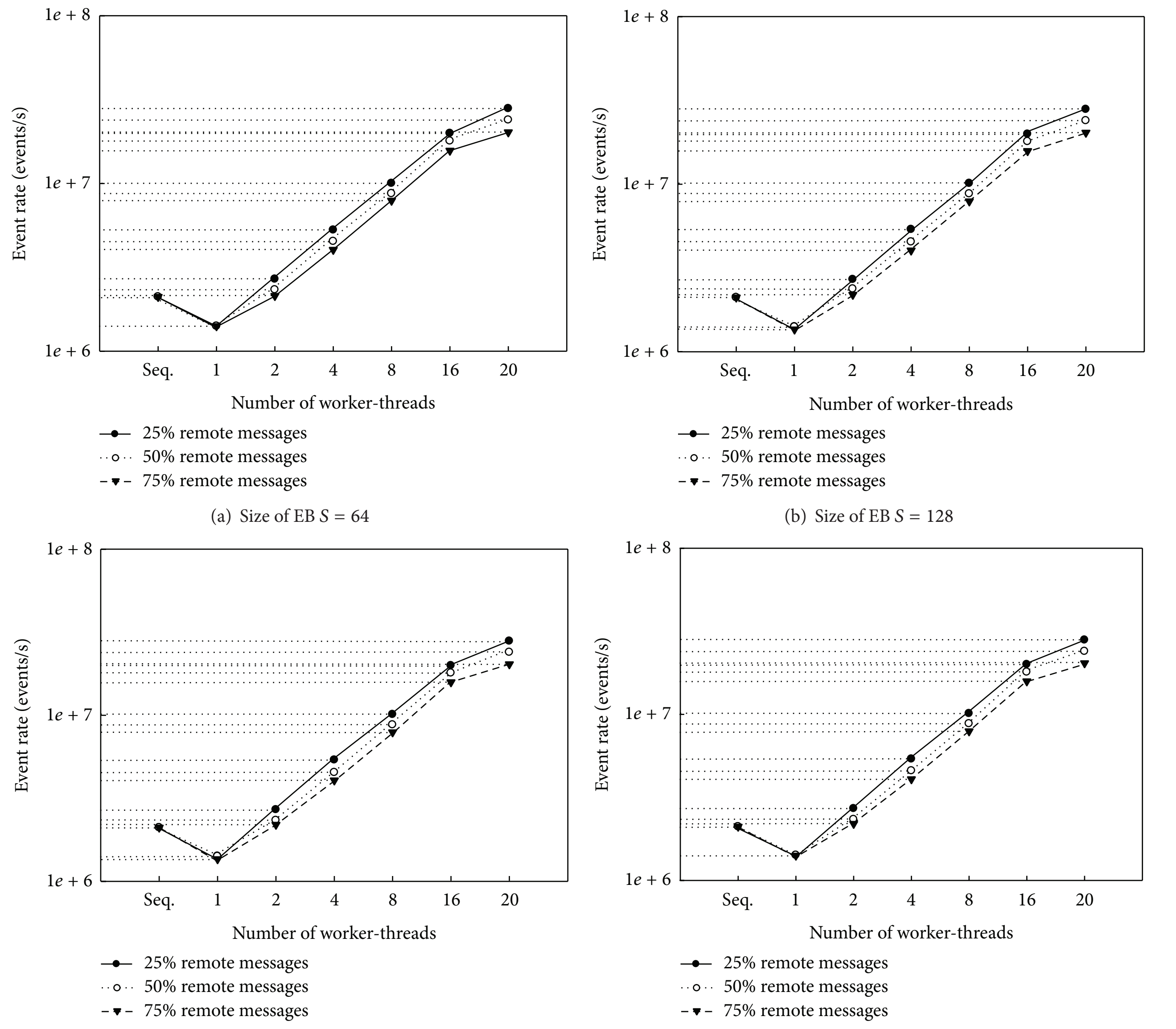
Figure 11 illustrates the committed event rates of our algorithm for the largest simulation with 396900 LPs. About 2.571×10^8 messages are committed in the experiment that spends 1200 units of simulation time. In case that the number of worker-threads is greater than three, k is set as two. For other cases, k is set as one. As shown in Figure 11, when the remote messages ratio of PHOLD model is 25%, committed event rate increases nearly linearly with the number of worker-threads. The algorithm is scalable with the number of worker-threads. The speedup reaches up to about 16 in case that the number of running worker-threads is 20. When the remote messages ratio reaches up to 50% and 75%, committed event rate drops by 21% and 30%, respectively. It is evident in Figure 11 that although there is a performance drop in terms of committed event rate, the performance of algorithm still grows almost linearly with number of worker-threads. We also find in Figure 11 that committed event rate changes slightly with the different size of EB.

Compared with a sequential algorithm, our algorithm has a performance drop in case of a single worker-thread. The committed event rate with one worker-thread in our algorithm decreases 40% compared with the sequential algorithm. We think there are three factors for this phenomenon. Firstly, it takes time to send GVT messages and calculate the GVT value. Secondly, there is an overhead of memory for storing and releasing the processed events in parallel algorithm when rollbacks occur [17]. Thirdly, parallel algorithm takes more time to tackle remote events between processors, while the sequential simulator does not have such kind of messages. However, the overhead introduced by all above factors is constant, so the performance of our algorithm grows linearly with the number of worker-threads.

Figure 12 shows the change of the committed event rate caused by different number of worker-threads (shown in axis x) and different number of LPs (shown in various line style). We observe that the performance degrades when the number of LPs decreases. As shown in Figure 12(a), when the number of LPs reduced from 396900 to 40000, the committed event rate decreased 23%. From Figures 12(a), 12(b), and 12(c), we find that there is a slight drop in the committed event rate when the number of remote messages increases. This phenomenon is caused by the increase of remote basic messages and GVT messages.

4.4. Accuracy of GVT Algorithm.

We define the accuracy of a GVT algorithm as the mean ratio of estimated GVT value to actual GVT value. The actual GVT value can be only obtained by the most efficient optimistic GVT algorithm, which can speculatively process events and restrain the occurrence of rollbacks. The estimated GVT value is an approximation of the actual GVT value. A higher accurate GVT value is helpful for the algorithm to commit more events processed and release more memory allocated for the events. It is impossible to continually measure the actual GVT at all time points during the simulation execution. However, we can obtain an approximate actual GVT (pseudoactual GVT) by setting



(a) Size of EB $S = 64$

(b) Size of EB $S = 128$

(c) Size of EB $S = 256$

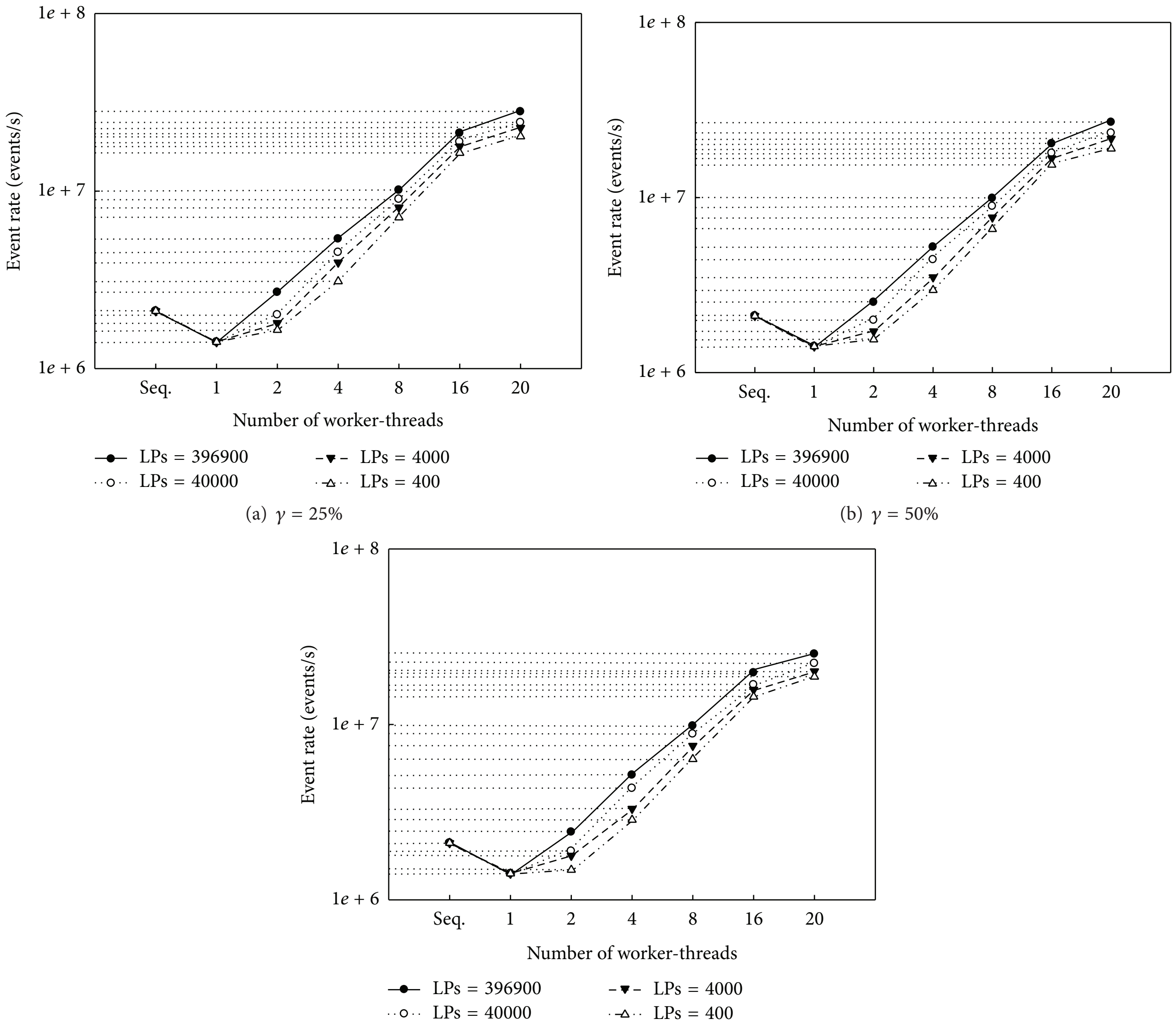
(d) Size of EB $S = 512$

FIGURE 11: Result of the committed event rate ($M = 396900$, $\lambda = 1$, and $k = 2$ for $N > 2$).

the remote event rate as zero to ensure that there is no rollback in the simulation execution.

The pseudoactual GVT value in our experiment is sampled per 0.1 seconds in wall-clock time. We compare the accuracy of our algorithm with that of FA-GVT. The results are shown in Figure 13. The accuracy of our algorithm is 95.9% in case of $S = 512$. In other words, the mean approximate value estimated by our algorithm possesses 95.9% precision with respect to pseudoactual GVT. When the size of EB S is equal to 1024, the accuracy of our algorithm drops to 93.6%. For FA-GVT, the accuracy is 65.8% in case of $\Delta t = 0.5$. However, when Δt increases to 1.0, the accuracy of FA-GVT increases to 66.3%. In Figure 13, we see that the GVT computation period of LB-EBT GVT algorithm ($S = 512$) is approximately equal to that of FA-GVT ($\Delta t = 1.0$). However, the accuracy of our algorithm is higher than that of FA-GVT, which also reflects that the overhead of our algorithms is lower than that of FA-GVT.

The relationship between the accuracy of our algorithm and the size of EB is shown in Figure 14. The accuracy of our algorithm decreases sharply when the size of EB exponentially increases. In case of the size of EB $S = 32$, the accuracy reaches a limitation. We then discover that the accuracy could not be improved any more even if the size of EB further decreases. The accuracy of LB-EBT GVT algorithm can be increased by decreasing the size of EB. However, when the accuracy reaches a threshold, it is difficult to improve it, even if the size of EB is small enough. This is explained as the accuracy is limited by the delay of messages and the overhead of our algorithm. As shown in Figure 14, the ratio of GVT messages to basic messages decreases exponentially with the exponential increase of the size of EB.



(a) $\gamma = 25\%$

(b) $\gamma = 50\%$

(c) $\gamma = 75\%$

Figure 12: Result of the committed event rate ($S = 512$, $\lambda = 1$, and $k = 2$ for $N > 2$).

5. Conclusions and Future Works

We propose a LB-EBT GVT algorithm for PDES in this study. GVT computation is partitioned into rounds by Event-Bulks. Event-Bulk is defined to track transient messages by recording the number of messages sent and received by LPs. In this way, the overhead introduced by message acknowledgement for solving transient message problem can be eliminated. High accurate GVT value can be obtained through decreasing the size of EB. In addition, through distributing the workload of GVT computation to all worker-threads according to a k-ary tree communication topology, we adopt an overlapping GVT computation approach in our algorithm to accelerate the computation of the GVT value.

Experiments and comparisons are performed to evaluate the overhead, scalability, and accuracy of our algorithm. An optimized value of parameter k in communication topology is obtained by the experiments. Experimental results indicate that our algorithm introduces slight overhead and outperforms the FA-GVT algorithm in terms of speedup and accuracy. Moreover, through adjusting the size of EB, our algorithm can obtain higher accurate GVT value than that of FA-GVT while the overhead of algorithm is still small enough.

Although our algorithm shows high performance and is scalable on the test-bed, there are still some issues that should be studied in the future. Firstly, the size of EB should be self-regulated according to the memory usage of a simulation, the event granularity, and the accuracy requirement of the GVT value. Secondly, the PDES platform of our algorithm is a prototypal test-bed, which is quite distant from real advanced PDES optimistic simulation platforms in terms of capabilities and actual support for model development. In the future, we will carry out our algorithm within the famous PDES

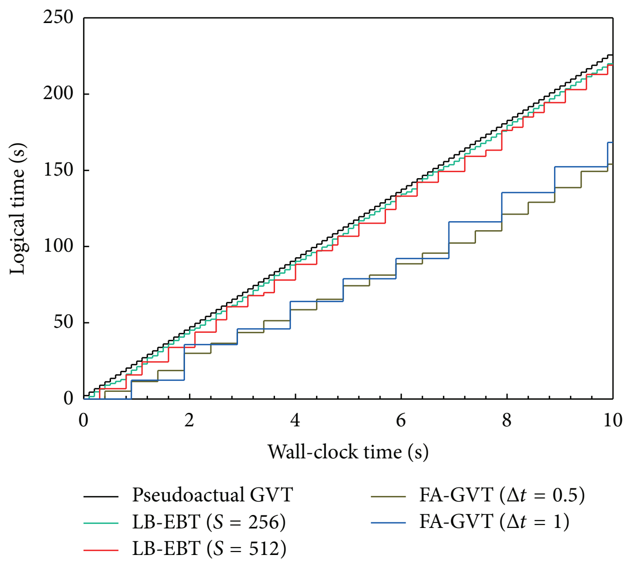


Figure 13: Comparison of exact GVT with estimated GVT ($M = 396900$, $\lambda = 1$, $\gamma = 25\%$, $N = 20$, and $k = 2$).

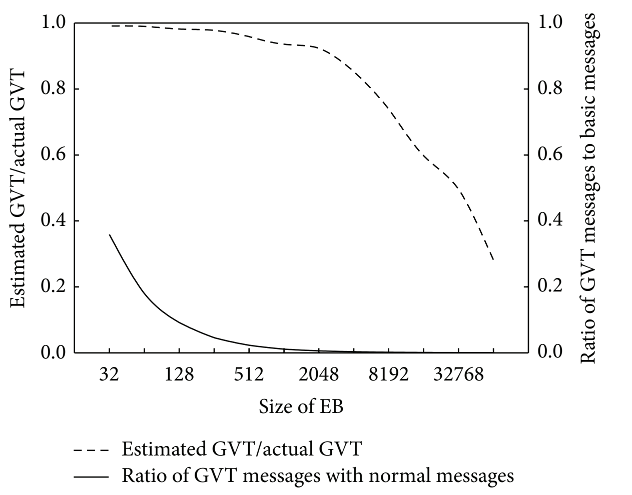


Figure 14: Accuracy of estimated GVT value and ratio of GVT messages to basic messages ($M = 396900$, $\lambda = 1$, $\gamma = 25\%$, $N = 20$, and $k = 2$).

platform, such as ROOT-Sim [13, 14] and ROSS [10], and confirm its performance and scalability on a really distributed architecture, for example, Tianhe super parallel computation system [44].

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

The authors gratefully acknowledge the financial support of the National Science Foundation of China (nos. 61473300, 61374185, and 61403402). A lot of thanks should be given to referees and editors; their valuable comments greatly improved the quality of the paper.

References

[1] J. C. Li, J. Y. Li, D. X. Niu, and Y. N. Wu, "A parallel adaptive particle swarm optimization algorithm for economic/environmental power dispatch," *Mathematical Problems in Engineering*, vol. 2012, Article ID 271831, 14 pages, 2012.

[2] K. S. Perumalla and S. K. Seal, "Discrete event modeling and massively parallel execution of epidemic outbreak phenomena," *Simulation*, vol. 88, no. 7, pp. 768–783, 2012.

[3] X. Wang, J. H. Zhang, and M. Scalia, "Parallel motion simulation of large-scale real-time crowd in a hierarchical environmental model," *Mathematical Problems in Engineering*, vol. 2012, Article ID 918497, 15 pages, 2012.

[4] F. Cicirelli, A. Furfaro, and L. Nigro, "An agent infrastructure over HLA for distributed simulation of reconfigurable systems and its application to UAV coordination," *Simulation*, vol. 85, no. 1, pp. 17–32, 2009.

[5] S. Jafer, Q. Liu, and G. Wainer, "Synchronization methods in parallel and distributed discrete-event simulation," *Simulation Modelling Practice and Theory*, vol. 30, pp. 54–73, 2013.

[6] D. R. Jefferson, "Virtual time," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 3, pp. 404–425, 1985.

[7] K. Perumalla, "Usik-a micro-kernel for parallel/distributed simulation," in *Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*, pp. 59–68, 2005.

[8] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: a high-performance, low-memory, modular time warp system," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[9] S. Das, R. Fujimoto, K. Panesar, D. Allison, and M. Hybinette, "GTW: a time warp system for shared memory multiprocessors," in *Proceedings of the Winter Simulation Conference*, pp. 1332–1339, Lake Buena Vista, Fla, USA, December 1994.

[10] D. Bauer, G. Yaun, C. D. Carothers, M. Yuksel, and S. Kalyanaraman, "ROSS.Net: optimistic parallel simulation framework for large-scale Internet models," in *Proceedings of the Winter Simulation Conference: Driving Innovation*, vol. 1, pp. 703–711, IEEE, New Orleans, La, USA, December 2003.

[11] D. Martin, P. Wilsey, R. Hoekstra et al., "Redesigning the WARPED simulation kernel for analysis and application development," in *Proceedings of the 36th Annual Simulation Symposium*, pp. 216–223, IEEE, Orlando, Fla, USA, March-April 2003.

[12] E. Mascarenhas, F. Knop, and V. Rego, "ParaSol: a multithreaded system for parallel simulation based on mobile threads," in *Proceedings of the Winter Simulation Conference (WSC '95)*, pp. 690–697, Arlington, Va, USA, December 1995.

[13] A. Pellegrini and F. Quaglia, "The ROme OpTimistic simulator: a tutorial," in *Euro-Par 2013: Parallel Processing Workshops*, vol. 8374 of *Lecture Notes in Computer Science*, pp. 501–512, Springer, Berlin, Germany, 2014.

[14] A. Pellegrini, R. Vitali, and F. Quaglia, "The ROme OpTimistic Simulator: core internals and programming model," in *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SimuTools '11)*, pp. 96–98, March 2011.

[15] A. Pellegrini and F. Quaglia, "Wait-free global virtual time computation in shared memory timewarp systems," in *Proceedings of the 26th IEEE International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD '14)*, pp. 9–16, IEEE, Jussieu, France, October 2014.

[16] P. M. Dickens, D. M. Nicol, P. F. Reynolds Jr., and J. M. Duva, "Analysis of bounded time warp and comparison with YAWNS," *ACM Transactions on Modeling and Computer Simulation*, vol. 6, no. 4, pp. 297–320, 1996.

[17] G. Chen and B. Szymanski, "Time quantum GVT: a scalable computation of the global virtual time in parallel discrete event simulations," *International Journal for Parallel and Distributed Computing*, vol. 8, pp. 423–436, 2007.

[18] R. M. Fujimoto and M. Hybinette, "Computing global virtual time in shared-memory multiprocessors," *ACM Transactions on Modeling and Computer Simulation*, vol. 7, no. 4, pp. 425–446, 1997.

[19] Z. Xiao, F. Gomes, B. Unger, and J. Cleary, "A fast asynchronous GVT algorithm for shared memory multiprocessor architectures," in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pp. 203–208, IEEE, Lake Placid, NY, USA, June 1995.

[20] S. Leye, A. M. Uhrmacher, and C. Priami, "A bounded-optimistic, parallel beta-binders simulator," in *Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT '08)*, pp. 139–148, October 2008.

[21] S. S. Rizvi, A. Riasat, and K. M. Elleithy, "An efficient optimistic time management algorithm for discrete event simulation system," *International Journal of Simulation Model*, vol. 9, no. 3, pp. 117–130, 2010.

[22] R. M. Fujimoto, "Performance of time warp under synthetic workloads," in *Proceedings of the Multiconference on Distributed Simulation*, pp. 23–28, Society for Computer Simulation, January 1990.

[23] R. M. Fujimoto, "Parallel simulation: distributed simulation systems," in *Proceedings of the 35th Winter Simulation Conference*, pp. 124–134, December 2003.

[24] R. Baldwin, M. J. Chung, and Y. Chung, "Overlapping window algorithm for computing GVT in time warp," in *Proceedings of the 11th International Conference Distributed Computing Systems*, pp. 534–541, 1991.

[25] S. Bellenot, "Global virtual time algorithms," in *Proceedings of the SCS Multi-conference on Distributed Simulation*, pp. 122–127, San Diego, Calif, USA, January 1990.

[26] S. K. Das and F. Sarkar, "A hypercube algorithm for GVT computation and its application in optimistic parallel simulation," in *Proceedings of the Simulation Symposium*, pp. 51–60, Phoenix, Ariz, USA, 1995.

[27] B. Samadi, *Distributed Simulation, Algorithms and Performance Analysis*, Computer Science Department, University of California, Los Angeles, Calif, USA, 1985.

[28] H. Bauer and C. Sporrer, "Distributed logic simulation and an approach to asynchronous GVT-calculation," in *Proceedings of the 6th Workshop Parallel and Distributed Simulation*, pp. 205–208, Newport Beach, Calif, USA, 1992.

[29] Y. C. Zhang and G. Li, "SafeBTW: a scalable optimistic yet non-risky synchronization algorithm," in *Proceedings of the ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation (PADS '12)*, pp. 75–77, Zhangjiajie, China, July 2012.

[30] Y. B. Lin and E. D. Lazowska, "Determining the global virtual time in a distributed simulation," in *Proceedings of the International Conference Parallel Processing*, pp. 201–209, Boston, Mass, USA, 1990.

[31] F. Mattern, "Efficient algorithms for distributed snapshots and global virtual time approximation," *Journal of Parallel and Distributed Computing*, vol. 18, no. 4, pp. 423–434, 1993.

[32] M. Choe and C. Tropper, "An efficient GVT computation using snapshots," in *Proceedings of the Conference on Computer Simulation Methods and Applications*, pp. 33–43, Orlando, Fla, USA, 1998.

[33] A. I. Tomlinson and V. K. Garg, "Algorithm for minimally latent global virtual time," in *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pp. 35–41, San Diego, Calif, USA, May 1993.

[34] K. Perumalla and R. M. Fujimoto, "Virtual time synchronization over unreliable network transport," in *Proceedings of the 15th Workshop Parallel and Distributed Simulation*, pp. 129–136, Lake Arrowehead, Calif, USA, 2001.

[35] S. Srinivasan and P. F. Reynolds Jr., "Non-interfering GVT computation via asynchronous global reductions," in *Proceedings of the Winter Simulation Conference*, pp. 760–769, Los Angeles, Calif, USA, 1993.

[36] J. S. Steinman, C. A. Lee, L. F. Wilson, and D. M. Nicol, "Global virtual time and distributed synchronization," in *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95)*, pp. 139–148, Lake Placid, NY, USA, June 1995.

[37] L. M. D'Souza, X. Fan, and P. A. Wilsey, "pGVT: an algorithm for accurate GVT estimation," in *Proceedings of the 8th Workshop Parallel and Distributed Simulation*, pp. 102–109, Edinburgh, UK, July 1994.

[38] D. Bauer, G. Yaun, C. D. Carothers, M. Yuksel, and S. Kalyanaraman, "Seven-o'Clock: a new distributed GVT algorithm using network atomic operations," in *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation (PADS '05)*, pp. 39–48, Monterey, Calif, USA, June 2005.

[39] E. Deelman and B. K. Szymanski, "Continuously monitored global virtual time," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA '97)*, pp. 1–10, Las Vegas, Nev, USA, June-July 1997.

[40] J. L. Gross, J. Yellen, and P. Zhang, *Handbook of Graph Theory*, CRC Press, Boca Raton, Fla, USA, 2nd edition, 2014.

[41] R. Vitali, A. Pellegrini, and F. Quaglia, "Towards symmetric multi-threaded optimistic simulation kernels," in *Proceedings of the 26th Workshop on Principles of Advanced and Distributed Simulation (PADS '12)*, pp. 211–220, IEEE, Zhangjiajie, China, July 2012.

[42] R. Vitali, A. Pellegrini, and F. Quaglia, "Load sharing for optimistic parallel simulations on multi core machines," *SIGMETRICS Performance Evaluation Review*, vol. 40, no. 3, pp. 2–11, 2012.

[43] Y. B. Lin and E. D. Lazowska, "Processor scheduling for time warp parallel simulation," in *Proceedings of the 23rd SCS Multi Conference on Advances in Parallel and Distributed Simulation*, pp. 11–14, IEEE Computer Society, 1991.

[44] X.-J. Yang, X.-K. Liao, K. Lu, Q.-F. Hu, J.-Q. Song, and J.-S. Su, "The TianHe-1A supercomputer: its hardware and software," *Journal of Computer Science and Technology*, vol. 26, no. 3, pp. 344–351, 2011.