Int j simul model 9 (2010) 3, 117-130
ISSN 1726-4529 Original scientific paper

AN EFFICIENT OPTIMISTIC TIME MANAGEMENT ALGORITHM FOR DISCRETE-EVENT SIMULATION SYSTEM

Rizvi, S. S.[*]; Riasat, A.[**] & Elleithy, K. M.[*]
[*] Computer Science and Engineering Department, University of Bridgeport, Bridgeport, CT 06604, USA
[**] Department of Computer Science, Institute of Business Management, Karachi, 75100, Pakistan
E-Mail: srizvi@bridgeport.edu, aasia.riasat@iobm.edu.pk, elleithy@bridgeport.edu

Abstract
Time Wrap algorithm is a well-known mechanism of optimistic synchronization in a parallel discrete-event simulation (PDES) system. It offers a run time recovery mechanism that deals with the causality errors. For an efficient use of rollback, the global virtual time (GVT) computation is performed to reclaim the memory, commit the output, detect the termination, and handle the errors. This paper presents a new unacknowledged message list (UML) scheme for an efficient and accurate GVT computation. The proposed UML scheme is based on the assumption that certain variables are accessible by all processors. In addition to GVT computation, the proposed UML scheme provides an effective solution for both simultaneous reporting and transient message problems in the context of synchronous algorithm. To support the proposed UML approach, two algorithms are presented in details, with a proof of its correctness. Empirical evidence from an experimental study of the proposed UML scheme on PHOLD benchmark fully confirms the theoretical outcomes of this paper.
(Received in June 2009, accepted in April 2010. This paper was with the authors 5 months for 3 revisions.)

Key Words: Discrete Event Simulation, GVT Computation, Optimistic Algorithm, Parallel and Distributed Systems, Time Wrap Algorithm

1. INTRODUCTION

The main problem associated with the distributed system is the synchronization among the discrete events that run simultaneously on multiple machines [1]. If the synchronization problem is not properly handled, it can degrade the performance of parallel discrete event simulation (PDES) [2]. Historically, two main methods have been introduced to deal with this problem: conservative [3, 4] and the optimistic synchronization algorithms (or Time Wrap) [5]. Two of the most common synchronization protocols for parallel simulation are the Chandy-Misra protocol [3] and the Time Warp protocol [5] (different approaches for parallel and discrete-event simulation and its applications are discussed elsewhere [6-12]). An introduction to the Chandy-Misra protocol and the Time Warp protocol can be found in [6, 13]. The conservative synchronization ensures that the local causality constrain requirement must not be violated by the logical processes (LPs) within the simulation system [14]. On the other hand, optimistic synchronization allows the violation of the local causality constraint requirement. However, such violation can not only be detected at run time but can also be dealt by using the rollback mechanism provided by optimistic algorithms [14-16].

The Time Wrap [5, 17] is one of the mechanisms of optimistic time management algorithm (TMA) which includes rollback, anti-message, and global virtual time (GVT) computation techniques [1]. GVT defines a lower bound on any unprocessed event in the system and defines the point beyond which events should not be reclaimed [15]. GVT computation is perhaps the only global operation in Time Warp. All other operations, such as rollbacks, state saving, and sending and handling of anti-messages, can be carried out locally.

Therefore, GVT computation is known to be the least scalable component of Time Warp and it is no surprise that the accuracy and overhead of the GVT computation may dominate the overall performance of Time Warp [18]. The rollback mechanism is used to remove causality errors by dealing with straggling events. The straggling events are referred to those events whose time-stamp is less than the current simulation time of an LP. In addition, the occurrence of a straggling event may cause the propagation of incorrect events messages to the other neighboring LPs. The anti-message is one of the techniques of the Time Wrap algorithm that deals with the incorrect event messages by cancelling them out.

2. RELATED WORK

In order to achieve optimized performance from the Time Wrap algorithm, it is essential that the GVT computation should operate as efficiently as possible. The GVT computation method, widely used in earlier algorithms [19-21], is generally based on the two rounds of message transmission. In the first round, a start message is transmitted for initiating the GVT computation. After the transmission of the first message, the initiator goes into the wait stage unless it receives all responses from the LPs. Once the initiator becomes active, a stop message is transmitted to announce the new value of GVT. Both rounds define an interval for GVT computation.

In addition to define a lower bound on the unprocessed events, a GVT algorithm must also address the following two problems: transient message and simultaneous reporting problems [22]. Transient messages are those that have been sent but have not been yet received [1]. Since a transient message is a delayed message, neither the sender nor the receiver considers the time stamp of the message in their respective GVT computation. Thus, in order to calculate a correct value of GVT, these messages must be accounted in the GVT computation for by either a sender or a receiver or both. The simultaneous reporting problem arises because not all LPs report their lower bound on time stamp (LBTS) value at precisely the same instant in wall clock time [2].

Some earlier GVT computation algorithms [19-21] provide a simple solution of message acknowledgments for dealing with these two problems. In this solution, any message whose acknowledgment has not been received will be considered as a transient message and its corresponding timestamp must be considered during the GVT computation. It has been observed [15, 18] that the earlier GVT algorithms [19-21] provide a significant transmission overhead in terms of the number of messages that an LP needs to exchange and maintain during the GVT computation process.

Samadi's algorithm [19] provides a foolproof solution to all problems cited above as long as the algorithm is implemented as described. However, the algorithm itself does not guarantee that the GVT computation is fast enough that it can minimize the execution time or the number of GVT messages. In addition, the primary problem associated with the Samadi's algorithm is that it requires acknowledgement messages to be sent for each message and anti-message. Besides a large number of these acknowledgement messages that each LP needs to transmit, Samadi's algorithm requires that each LP maintains at least three separate queues, so that the LP can transmit the following information to the controller upon receiving the GVT computation message: the minimum time stamp of all the unprocessed event-message within the LP, all unacknowledged and anti-messages it has sent, and all marked acknowledgement messages it has received. This implies that the Samadi's algorithm not only requires maintaining a large number of queues but also demands transmitting comparatively large amount of information to the controller in response to the GVT computation message.

The performance degradation in optimistic algorithm is due to the fact that the large transmission of messages across LPs cause frequent state saving and rollbacks. Under heavy

load of network messages, optimistic simulators pay heavy synchronization cost in terms of large number of state saving and rollbacks. Since the messages are randomly exchanged across the LPs, there is a high probability of frequent occurrence of rollbacks as well as the size of the state grows with respect to an increase in the event-message traffic.

3. ANALYSIS OF GLOBAL VIRTUAL TIME (GVT) ALGORITHMS

The notion of Global Virtual Time (GVT) was first introduced by Jefferson [5] to track the earliest unprocessed events in the entire simulation. Formally, the GVT can be defined as a minimum time-stamp among all the unprocessed and partially processed event messages and anti-messages present in the simulation time at current clock time T_s. Any processed event with a timestamp earlier than the current GVT will not be rolled back under any circumstances, and therefore the memory associated with it can be safely released [18].

Without the use of GVT, the Time Warp mechanism would be impractical since it requires reasonably large memory. Thus, it is imperative that the GVT computation operates as efficiently as possible [15]. However, it is impossible to compute the exact GVT as it would require collecting information on distributed processors at exactly the same wall-clock time [18]. GVT is not only required for optimistic algorithm but it can also be used in few variants of conservative protocols, such as the conditional event approach [23] and the LBTS approach [24], which largely depend on the amount of Lookahead (we refer this to L value), that also need to compute LBTS which computationally is equivalent to GVT [18].

Designs of GVT algorithms focus on either shared-memory or distributed computers [18]. Shared-memory GVT algorithms assume that certain variables are accessible by all processors [25, 26], so they perform well on symmetric multi processing (SMP) machines. Distributed GVT algorithms do not use global variables and therefore are more scalable. Distributed GVT algorithms are further classified with respect to specific techniques they use such as overlapping intervals [20], two cuts [14, 27], or global reduction [28, 29].

The proposed unacknowledged message list (UML) scheme, however, differs from the other traditional schemes in such a way that it only requires each LP to maintain a single list for unacknowledged messages with one or more first-in-first-out (FIFO) queues of unprocessed event-messages. Due to a single list, the computation of local minimum for each LP requires fewer steps, thus provides a fast GVT computation. In addition to the fast GVT computation, a comparatively small amount of memory will be utilized per LP. For further optimization, the proposed scheme piggy-backs the acknowledgment messages in the regular outgoing event messages as described in [30]. Further reduction in message-overhead can achieve by using a sequence number as described in [8].

4. OVERVIEW OF UNACKNOWLEDGED MESSAGE LIST (UML)

The proposed scheme partially reduces the processor idle time at the expense of a very small amount of memory use by each LP to maintain an UML. Before we present the proposed scheme, it is worth mentioning some of our key assumptions.

4.1 System model and assumptions

We assume that the simulation system consists of n number of LPs where each LP maintains one FIFO queue per neighboring LPs that stores the corresponding incoming event-messages. The head of the FIFO queue contains the smallest time stamp event-message. For instance, if we assume a mesh topology for providing internetworking among LPs, then each LP must maintain at least one FIFO queue per neighboring LP resulting in a total of n-1 number of

FIFO queues for *n*-1. Moreover, we only consider those event-messages that are generated and scheduled for remote LPs. These event-messages can be referred as globally generated event-messages as opposed to the locally generated event-messages that are scheduled by an LP for itself. The simulation executive/engine is not only responsible to maintain FIFO queues within an LP but also responsible to search the head of each FIFO queue in order to determine the smallest time stamp event-message. This time stamp will be considered the time stamp of the next event-message that an LP will execute in a row. The computation of this time stamp is essential for solving the problem of transient messages and computing the LBTS values.

4.2 Algorithm description for UML

The primary difference between the proposed UML scheme and the other existing GVT algorithms is that it uses a dedicated controller LP to monitor the GVT computation process. The dedicated controller LP refers as *CR* in our proposed algorithm. Specifically, *CR* is responsible to initiate the GVT computation, collect local minimum values from each active LP, and finally announce the new GVT value as shown in Algorithm A. Non controller LPs are not directly involved in the GVT computation process except each LP is required to compute its local minimum and reports to *CR* as shown in Algorithm B. The details of each statement in both Algorithms A and B are provided by means of comments.

The proposed UML approach can be considered as a centralized approach since both steps of GVT computation is done by the dedicated LP which greatly simplifies the design of the GVT algorithm. In this perspective, proposed UML scheme is similar to the pGVT [31] and TQ-GVT [18] since both schemes require a single controller to monitor the GVT computation process. However, it differs from the TQ-GVT where GVT computation never initiated by the dedicated LP. Instead, the GVT master (i.e., the dedicated *CR* LP) passively listens to GVT messages and takes actions only when they come. In the proposed UML scheme, controller LP uses Algorithm A for both initiating the GVT computation and computing the new GVT value. Non controller LP uses Algorithm B for computing their local minimum and compiling the report for the *CR*.

The proposed UML scheme uses four different types of messages as shown in both Algorithms A and B. The first is the timestamp event message, denoted by $E(T_S, \overline{T}_S)$, which is

program controller (*CR*) LP *(n LBTSi)*

/**initialization phase for all the queues maintained by an LP*/
S1 Report [] = 0; LBTS = 0; GVT_{New} = 0; GVT_{Comp} = 0; Count = 0;
/**CR* initiates GVT computation by broadcasting a message for all LPs */
S2 **for all** LPs **do** GVT_{Comp}; /* $CR \xrightarrow{GVT_{Comp}} \{LP_1, LP_2, ..., LP_n\}$ */
/**CR* continuous to receive LBTS from each LP*/
S3 **while** (*CR* receive *Report* from each LP) **do**
/* A *Report* has received from LPi. Only first 2 elements are needed for GVT computation*/
S4 if Report $\left(LP_{ID}, LBTS, MST\left[S_T, L\right]\right)$ is received
S5 Report [LP_{ID}] = LBTS; /* local minimum for LPi is stored */
S6 Count = Count +1; /* count is incremented until it reaches to n */
end while /* *CR* counter reaches to *n*, indicating that *n* Report messages have received*/
/* computing new global minimum value*/
S7 GVT_{New} = Min (Report [LP_{ID}]);
/* *CR* announces the new GVT value by broadcasting a message for all LPs */
S8 **for all** LPs **do** GVT_{New}; /* $CR \xrightarrow{GVT_{New}} \{LP_1, LP_2, ..., LP_n\}$ */

Algorithm A: Controller LP simulation algorithm sketch for initiating and announcing GVT computation

the carrier of a positive event or an anti-event similar to the message structure described in [18]. The first variable T_s represents the timestamp of *E* whereas the $\overline{Ts}$ is the copy of the original time stamp stored in the UML. This message structure is implemented to ensure that the transient message(s) must be accounted in the GVT computation by each active LP. All variables $(T_S, \overline{T}_S)$ of *E* play an important role in dealing with the transient message and simultaneous reporting problems. The second is the GVT message transmits from *CR*, denoted by GVT_{Comp} that signals the start of the GVT computation (see line S2 of Algorithm A). This message type is similar to the GVT initiation process described in Fujimoto's shared memory GVT algorithm [25]. The third is the GVT message transmitted from *CR*, denoted by GVT_{New}, that simply contains the value of the new GVT estimate (see line S7 of Algorithm A). From the proposed algorithm perspective, both messages transmitted from *CR* to LPs use global variables to initiate GVT computation and announce the new estimated value of GVT.

Finally, the fourth is a report message which has the format of *Report* (LP_{ID}, *LBTS*, *MST* [S_T, *L*]). Upon completion of the local minimum computation, each LP has to compile the

(*Report*) **program** LP_i

/**initialization phase for all the queues maintained by an LP**/
S1 TS_{FIFO} = 0; TS_{UML} = 0; *UML* = { }; *LBTS* = 0; Report [] = 0;
S2 execute one or more events */*process both local and remote event messages*/*
S3 **for** any remote message E (Ts, $\overline{T}_S$) **do**
S4 UML [i] = $\overline{T}_S$; */*storing the copy of time stamp in UML for accounting transient message/*
S5 LP_i receives the GVT_{Comp} message from *CR /*receives the GVT request*/*
S6 **while** (*LPi not finish computing local minimum*) **do**
S7 **If** TS_{UML} = 1 **then** */*if there exists only one element in UML*/*
/*get *Min* (TS_{FIFO}) from *n* FIFOs*/
S8 TS_{FIFO} == *Min* {$H_{1(FIFO-1)}$, $H_{1(FIFO-2)}$,……, $H_{1(FIFO\ n-1)}$};
/*get TS_{UML} from UML_i for LP_i and initialize the $TS_{UML\ (Min)}$*/
S9 $TS_{UML\ (Min)}$ == TS_{UML};
/*compare $TS_{UML\ (Min)}$ and $TS_{FIFO\ (Min)}$ and select $LBTS_i$ for LP_i */
S10 $LBTS_i$ == *Min* {$TS_{UML\ (Min)}$, $TS_{FIFO\ (Min)}$};
S11 **return** (*Report*); /*report $LBTS_i$ from LP_i to *CR. Set* LP_{ID}*/
S12 **elseif** TS_{UML} > 1 **then** /*if there exists multiple elements in the UML*/
/*get *Min* (TS_{UML}) from UML_i for LP_i*/
S13 $TS_{UML\ (Min)}$ == *Min* {UML_1, UML_2,……,$UML_{(m-1)}$};
/*get $TS_{FIFO\ (Min)}$ from *n* FIFOs maintained by LP_i */
S14 TS_{FIFO} == *Min* {$H_{1\ (FIFO-1)}$, $H_{1(FIFO-2)}$,……, $H_{1(FIFO\ n-1)}$};
/*compare $TS_{UML\ (Min)}$ from $TS_{FIFO\ (Min)}$ and select $LBTS_i$ for LP_i */
S15 $LBTS_i$ == *Min* {$TS_{UML\ (Min)}$,$TS_{FIFO\ (Min)}$};
S16 **return** (*Report*); /*report $LBTS_i$ from LP_i to *CR. Set* LP_{ID} */
S17 **elseif** TS_{UML} = 0 **then** */*if there exists none element in the UML*/*
/*compute $TS_{FIFO(Min)}$ from *n* FIFOs*/
S18 TS_{FIFO} == *Min* {$H_{1(FIFO-1)}$, $H_{1(FIFO-2)}$,.., $H_{1(FIFO\ n-1)}$};
/*compute smallest time stamp from each FIFO queue*/
S19 $H_{1(FIFO\ J)}$ == Min {H_1, H_2 ,……., H_m};
S20 $LBTS_i$ == $TS_{FIFO\ (Min)}$; /* initialize the $LBTS_i$ with the TSFIFO (Min value) */
S21 **return** (*Report*); /*report $LBTS_i$ from LP_i to *CR. Set* LP_{ID} */
end if;
end while

Algorithm B: Algorithm for implementing unacknowledged message list (UML) in LP

report message which will be sent to *CR*. LP_{ID} in the report message is the processor id that reports its local minimum to *CR*. *LBTS* represents the lower bound on the timestamp of the event messages that could be delivered to the simulation in the future. *MST* [S_T, L] is the minimum sending time that represents the earliest time when a remote event message can be sent by an LP to one of its neighboring LPs. *MST* is the sum of the current simulation time (S_T) of an LP (also refer as local virtual time) and the Lookahead (L) value.

It should be noted that no other counters are needed to account the transient messages. Instead, in our proposed approach, this is typically done by UML that maintains a record of transient messages by simply storing the copy of the time stamp $\overline{T}_S$ of each outgoing message. A local minimum computation process of an LP can not be satisfied and completed unless the LP considers the minimum of all $\overline{T}_S$ stored in the UML in its LBTS computation (this is refer as UML_{Min} in our algorithms). The consideration of UML_{Min} in the local minimum computation is essential since this guarantees that the *CR* yields a correct value of GVT. The execution of report message and the UML in a LP is shown in Algorithm B (see lines S4-S17).

The main factor that contributes to the reduced message overhead for the proposed UML scheme is that the GVT computation does not interfere with the simulation activities except the transmission of GVT initiation message and the report collection. In addition, the participating LPs need not be engaged in the GVT computation process except computing their local minimum values and reporting to the *CR* when a request arrives. The reporting process by each LP should be done periodically without halting the normal execution of event-messages.

4.3 Proposed scheme for transient message problem

In our proposed scheme each LP maintains a list that contains the time stamp of each outgoing message as shown in Fig. 1. This list is referred as UML in our proposed scheme. The primary purpose of UML is to ensure that for each outgoing message, the sending LP ideally receives an acknowledgment. In addition, for each outgoing message, the sending LP must store the corresponding time stamp of the recently transmitted message in the UML. When the event-message is sent out, the copy of the time stamp of that event-message must be stored in the UML of that LP. On the other hand, when an LP schedules a remote event-message for one of its neighboring LPs, the receiving LP must send an acknowledgement back to the sending LP. Upon reception of an acknowledgment from the receiving LP for one of the previously sent messages, the sending LP eliminates the corresponding time stamp of the acknowledged message from the UML. This ensures that the LP does not need to account the acknowledged event-messages in the next GVT computation. It should also be noted that each LP does not require maintaining a list of incoming messages. Instead, the UML of each LP is responsible to take care of the unacknowledged event-messages only. When an LP is about to start computing its LBTS value, LP must ensure that the UML does not contain any time stamp of an unacknowledged message. However, if the UML is not empty, the smallest time stamp present in the UML must be considered by an LP in its LBTS computation.

Proof of correctness: If two or more time stamps are found in the UML, the smallest time stamp will be selected from the UML. The selected time stamp will then be compared to the time stamp of the next event-message that the LP is supposed to execute (i.e., the smallest time stamp within n-1 number of time stamps present at the head of the FIFO queues of an LP). Whichever is smallest will be selected as the time stamp of the next event-message that the LP will execute. As a result, the final time stamp will be considered by an LP during the LBTS computation. If only one time stamp is found in the UML for one of the

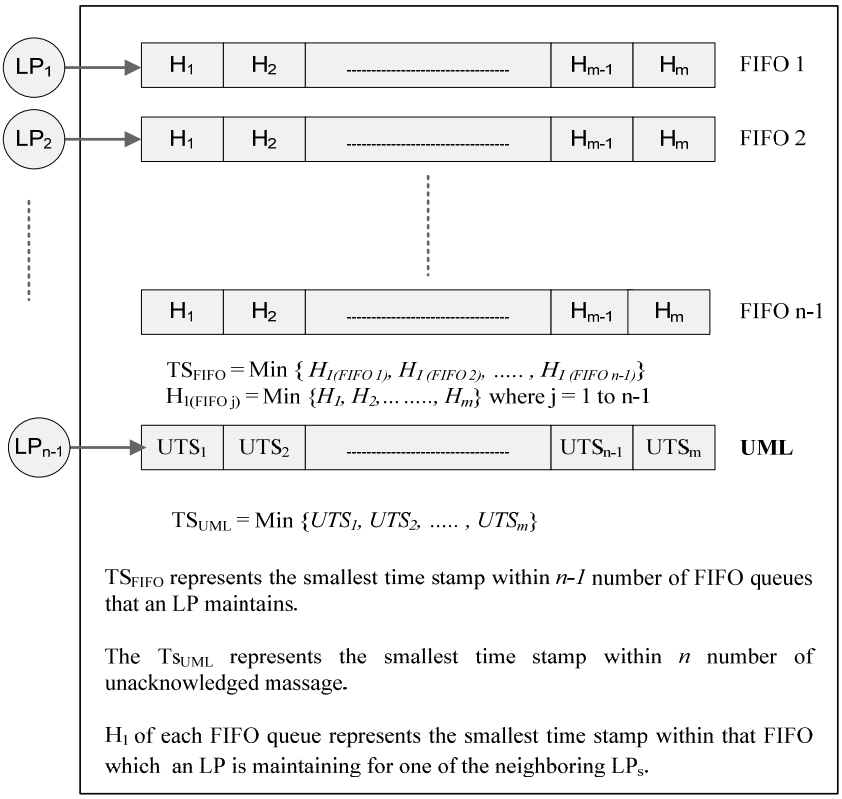

Figure 1: Internal architecture of an LP. LP maintains one FIFO queue per neighboring LP and one UML. Each LP can have a total of *m* number of event-messages ($0 \leq m \longrightarrow \infty$).

unacknowledged event-messages, then that time stamp must be compared with the time stamp of the next event message that an LP will execute. LP selects the smallest time stamp out of the two and uses that value in the LBTS computation. Finally, if none of the time stamp is found in the UML (i.e., no outstanding unacknowledged messages left in the list), the LP follows the regular procedure of computing its LBTS value by adding the time stamp of the next event-message with the corresponding Lookahead value. In particulate, this can be expressed as: LBTS $LP_k = TS_{FIFO} + LA_k$ where TS_{FIFO} represents the smallest time stamp of the next event message that an LP_k executes and LA_k represents the corresponding Lookahead value associated with the LP_k.

4.4 Solution of simultaneous reporting problem

An illustration of our proposed solution for the simultaneous reporting problem is shown in Fig. 2. The controller LP initiates the GVT computation by broadcasting a message for all LPs that exist within the simulation system. When an LP receives such message from the controller LP *CR*, it computes its local minimum value by using the proposed Algorithm B. Once the local minimum value is determined by an LP, the controller will be notified with a small synchronization message (we refer this message as *Report*). Once all the LPs transmitted their local minimum values to the controller, the controller selects the global minimum value. Once the global minimum value is determined, the controller LP broadcasts another message to notify each LP with the new value of global minimum. Upon reception of the new GVT value, each LP can then distinguish between the safe and unsafe event messages unless they receive another GVT computation message from the controller LP *CR*. If GVT messages do not come to time, active LPs are never delayed or blocked as in the case of some other algorithms. Since each LP uses UML that keeps track of all the unacknowledged

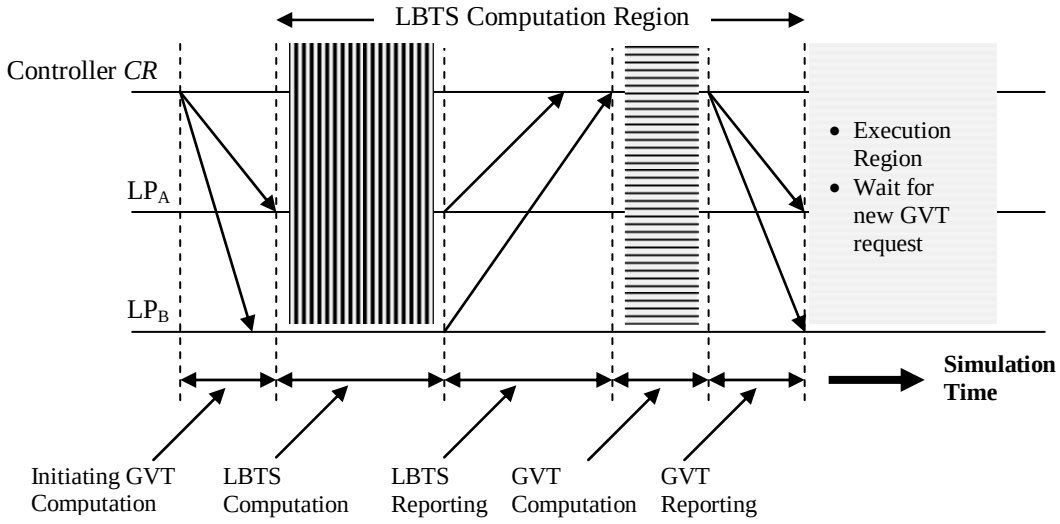

Figure 2: An illustration of proposed scheme dealing with the simultaneous reporting problem.

messages, the possibility that an LP does not consider the time stamp of one or more transient message is negligible. Thus, this guarantees that the use of UML in each LP yields a correct value of local minimum.

Proof of correctness: For instance, if LP_B receives a delayed GVT computation message from the controller, it does not cause the other LPs (such as LP_A) of the simulation system to report an incorrect local minimum value to *CR*. Since LP_A was bounded to the minimum value of either UML (we refer this to UML_{MIN} in Algorithm B) or TS_{FIFO} after receiving the GVT computation message, this forces the LP_A to stick with the same local minimum value within all the queues maintained in that LP (for implementation, see Fig. 3). For the same scenario (i.e., LP_B receives a delayed GVT computation message from the controller LP), even if LP_B sends an event-message to LP_A, LP_A will only accept it (i.e., it stores the event-message in its FIFO queue) but will not send any acknowledgement back to the sending LP (i.e., LP_B) with the assumption that the sending LP will consider the time stamp of this received event-message in its own local minimum computation. This also shows that when an LP receives a GVT computation message from the controller, it may accept the new event-message coming from the other neighboring LPs. However, it does not consider the time stamp of the newly arrived event-message in its local minimum computation. This is due to the fact that the receiving LP does not require maintaining the UML.

4.5 High level architecture of the proposed scheme

We assume that the simulation system consists of *n* number of LPs where each LP is assumed to execute the high level architecture of the proposed solution as shown in Fig. 3. Initially, the controller broadcasts a message to all LPs asking to initiate the GVT computation. Upon reception of this broadcast message, each LP initiates the LBTS computation based on the FIFO queues and the UML list that each LP maintains. Once the LBTS value for an LP is determined, it reports the new LBTS value to the controller.

The proposed scheme ensures that no LP advances its current simulation time beyond the value of the minimum time stamp in the entire simulation. For instance, if one of the LPs receives the GVT computation message from the controller, it performs the following steps to complete the computation process as shown in Fig. 3. Upon reception of the GVT initiation message from the controller, each LP initiates the LBTS computation. In order to determine the LBTS value, each LP must first visit the UML to count the number of unacknowledged

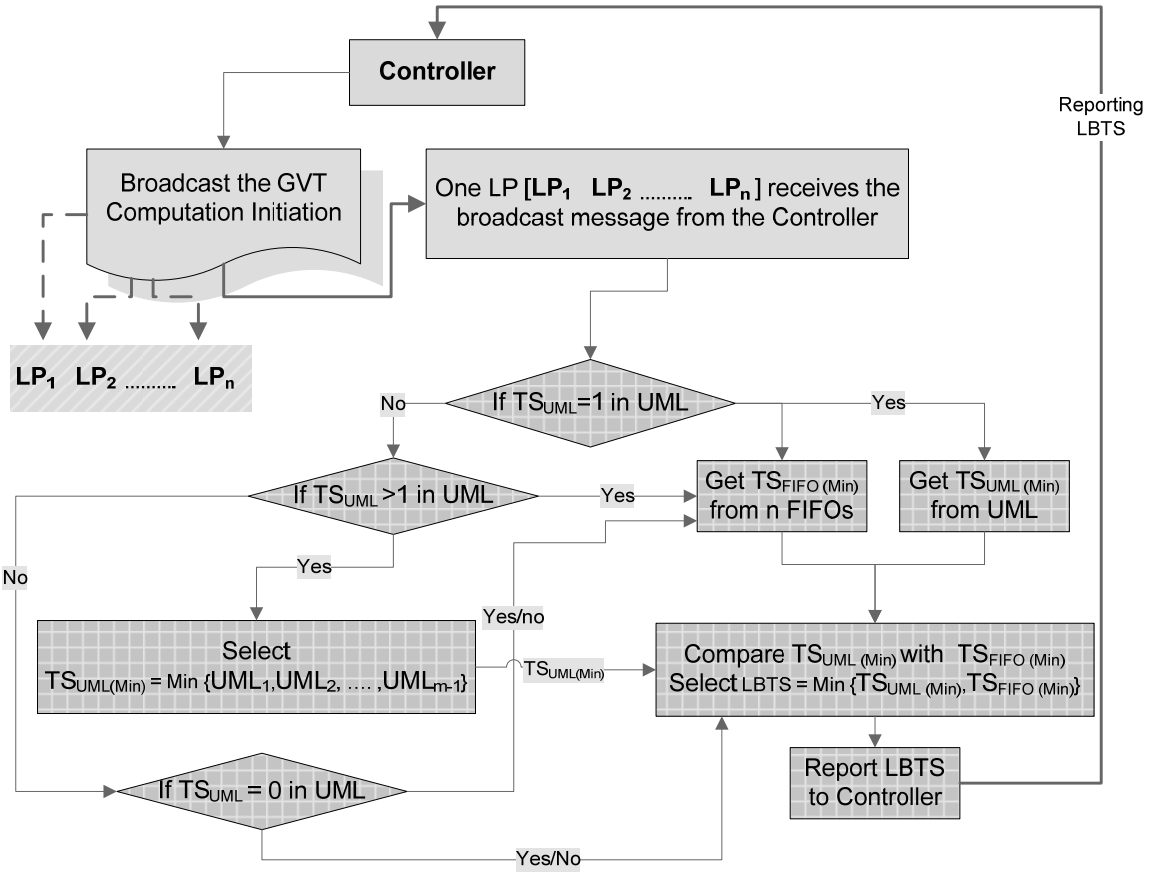

Figure 3: High level architecture of the proposed scheme.

messages present in the list. Therefore, the first conditional-box determines the possibility of the presence of only one unacknowledged message in the UML. The TS_{UML} represents the number of time stamps of the unacknowledged messages present in the list. If there is only one time stamp message exists in the UML, that time stamp will be selected and removed from the UML list and forwarded to the comparison/selection box. This time stamp is represented as $TS_{UML(Min)}$. At the same time, all the head of the FIFO queues will be exhaustively searched in order to determine the time stamp of the next event-message that the LP executes. Once determine, the time stamp will be forwarded to the comparison/selection box. This time stamp is represented as $TS_{FIFO(Min)}$. In the comparison/selection box, the two input values will be compared and the smallest one will be selected as the new local LBTS value of the LP. Finally, the resultant LBTS value will be reported to the controller.

On the other hand, if the first conditional box produces a false value, the control will be transferred to the second conditional box. In the second conditional box, the presence of multiple time stamps in the UML will be tested. If more than one time stamps are presented in the UML, the smallest value of the time stamp will be selected and forwarded to the comparison/selection box for further processing. The selection of minimum value of time stamp in the UML is presented in Fig. 3 such as: $TS_{UML(Min)} = Min\ \{UML_1, UML_2, \ldots, UML_{m-1}\}$ where m represents the total number of messages the UML has at the time of selection. At the same time, the control is transferred to search all the head of the FIFO queues, so that we determine the time stamp of the next event-message that the LP executes. Once the value of $TS_{FIFO(Min)}$ is determined, the selected time stamp will be forwarded to the comparison/selection box. The same comparison and selection will be performed between the two values and the resultant LBTS will be reported to the controller.

Finally, if the UML shows an empty list indicating that all the sent messages have been acknowledged by the receiving LPs, the control will unconditionally transfer to the comparison/selection box. At the same time, the minimum value of $TS_{FIFO(Min)}$ is determined and forwarded to the comparison/selection box. Since the UML shows an empty list, the $TS_{FIFO(Min)}$ value is selected as a new LBTS which consequently reported to the controller.

5. PERFORMANCE RESULTS OF THE PROPOSED UML SCHEME

Experiments were performed to compare the performance of proposed UML approach with both the tree and the butterfly barriers with respect to event processing rate and GVT message generation. All experiments were run on a dedicated SGI Origin 2000 server with 16 processors running on IRIX version 6.5. The SGI server has two R10000 processors per IP27 board running on 180 MHz clock speed with second level cache of 1 MB with fifteen 1.34 GHz Sun Ultra 25 workstation running version 10 of Sun Solaris operating system with the Sun's development tools. Each Sun Ultra 25 workstation carries UltraSPARC IIIi processor performance and enhanced connectivity.

For the sake of simulation results and experimental verifications, we use PHOLD benchmark, a synthetic workload generator proposed by Fujimoto [32]. PHOLD is a commonly used benchmark for testing the performance of Time Warp simulators [33, 34]. PHOLD has minimal event processing, minimal look ahead due to event scheduling being based on a random distribution and a random communication pattern. In general, PHOLD model consists of *n* fully connected LPs among which a fixed message population circulates [35]. It can be parameterized by: (i) the routing probabilities; (ii) the message population size; (iii) message size; (iv) timestamp increment distribution; and (v) a spin-delay simulating event granularity.

In PHOLD, processing of each message takes a finite amount of time, after which a new message is sent to another LP with a specified time stamp increment. The initial event messages have a timestamp that is exponentially distributed between 0 and 1. We used a variable size of message population with the initial size of 16 messages per LP. The number of LPs involved in the simulation model has been fixed at 256 and the model is executed on 16 machines with even distribution of the LPs on the machines resulting in 16 LPs per processor.

5.1 Performance evaluation of UML scheme

Figs. 4 and 5 show the aggregate event rate of PHOLD as a function of processor count. Fig. 5 shows that the UML scheme with the tree barrier continues to provide linear speedup than the butterfly barrier. For the 16 events per LP case, we observed a rate of 38,000 events/second on 10 processors (the other 6 processors are never used in this simulation), which remains almost linear and stable throughout the execution. Comparing the simulation results of Fig. 4 with Fig. 5, we observed that the introduction of variable message population results in the performance degradation in terms of event processing rate. The number of remote event messages increases slightly with the number of processors, since the amount of messages on each processor was fixed for Fig. 4 simulation results. However, in Fig. 5, there is a slight decrease in the event processing rate with respect to the number of processors. In addition to the introduction of variable size messages per processor, this slight drop in the event processing rate may also happen due to the fact that there is an overhead of memory and time used for storing and releasing the processed events in the parallel processor for using in case of rollbacks [18].

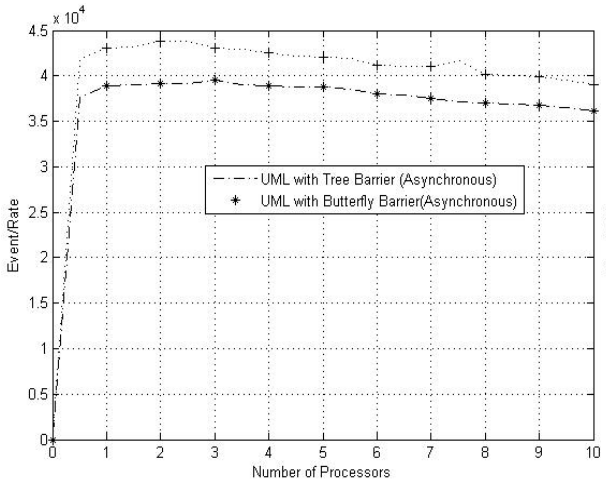

Figure 4: Event processing rate with the PHOLD across multiple runs on 16 processors for 16 messages per LP (fixed message population).

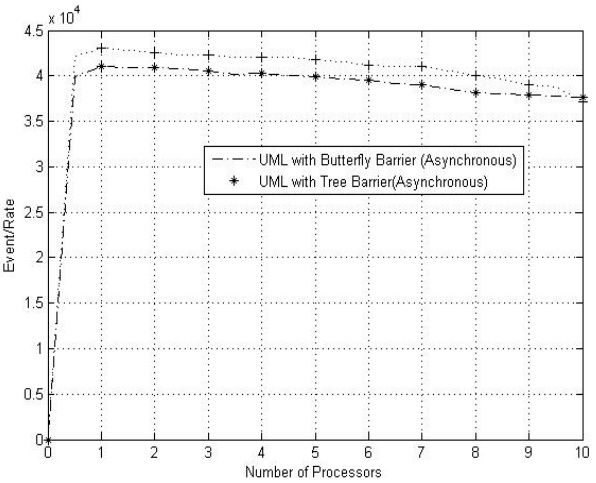

Figure 5: Event processing rate with the PHOLD across multiple runs on 16 processors for 16 messages per LP (variable message population starting from 16 messages per LP).

Figs. 6 and 7 show the GVT message generation for the corresponding event message generation presented in Figs. 4 and 5, respectively. For Fig. 6, the number of GVT messages almost remains same for the number of processors except there is a slight 30 % decrease of GVT messages. This slight reduction in GVT messages was caused since we found that the GVT computation was rarely initiated by the *CR* during the overall simulation process. It should be noted that these GVT messages are generated with a fixed message population on 16 processors. In harmony with our expectations, Fig. 7 shows a constant number of GVT messages even with a variable size of message load on each processor. When comparing the result of Fig. 7 with Fig. 5, we can observe that both suffer from slight performance degradation due to a variable load on each processor. The GVT messages for Fig. 7 are not only smooth but also stable for all values of processors except the same 35 % reduction can be

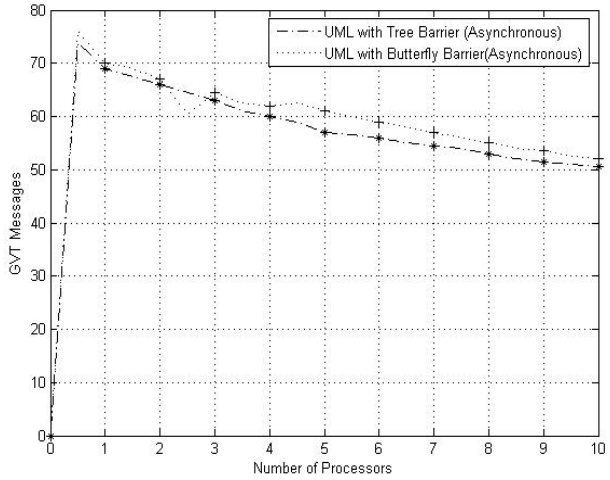

Figure 6: GVT messages for PHOLD across multiple runs on 16 processors for 16 messages per LP (fixed size of messages population per LP starting from 16 messages per LP with the increment of zero).

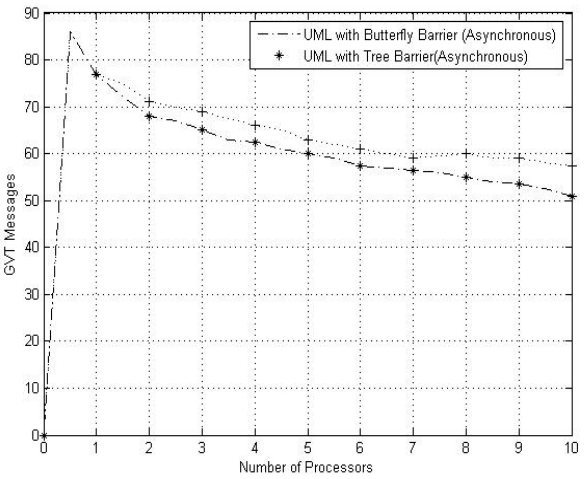

Figure 7: GVT messages for PHOLD across multiple runs on 16 processors for 16 messages per LP (with variable size message population per LP starting from 16 messages per LP with a random increment).

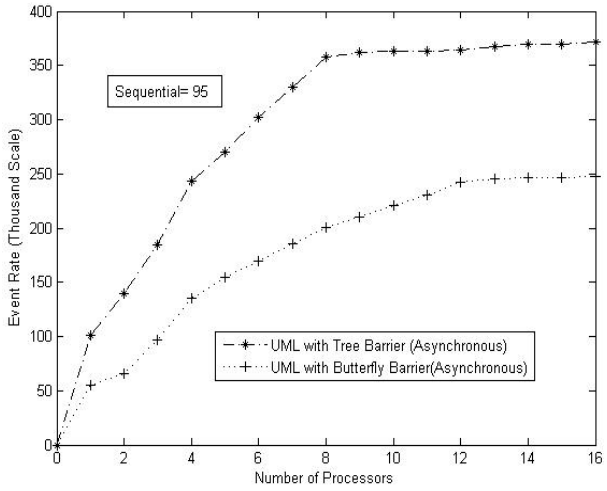

Figure 8: Event processing rate with the PHOLD using 16 processors with a variable message size population of 16 messages per LP on a total of 256 LPs.

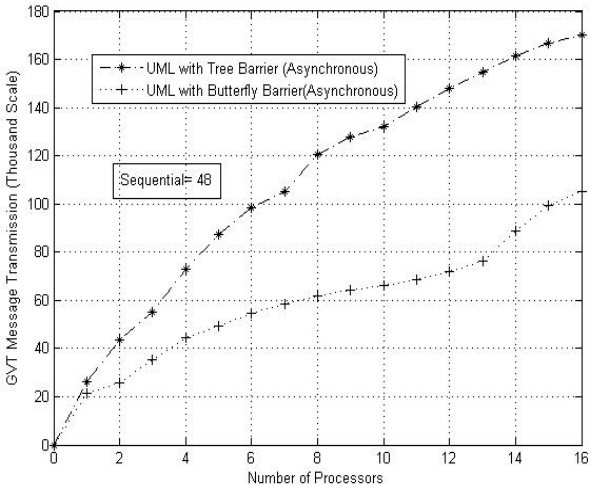

Figure 9: GVT message transmission for PHOLD across 16 processors with a fixed message size population of 16 messages per LP on a total of 256 LPs.

seen for the same reason discussed for Fig. 6.

The simulation results of event processing rate and GVT messages with variable size of message population for 16 processors on PHOLD benchmark are shown in Figs. 8 and 9, respectively. For these two simulation results, the same configuration parameters were used as we discussed for Figs. 5 and 6, except that we used all 16 processors with comparatively large message population. For Fig. 8, the number of remote event messages increased linearly with the number of processors, except that changing from 8 processors to 16 processors caused a sudden stability in the remote event processing. The number of GVT messages increased linearly too in Fig. 9, except that a wide difference can be found between tree and butterfly barriers from 8 processors to 16 processors. Finally, for the sake of computing the speedup, we have run the experiments for selective 4, 8, and 12 processors with respect to the memory buffer requirements. In Fig. 10, it was observed that the achievable speedup using the

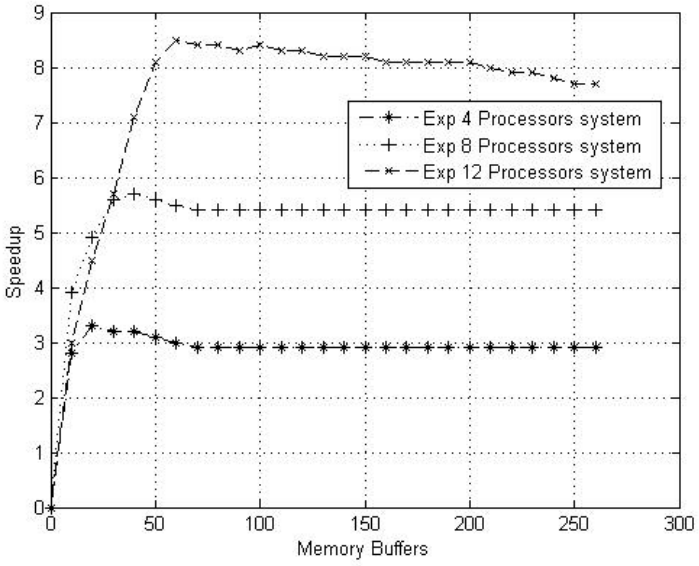

Figure 10: Effect of unacknowledged message list (UML) scheme on the speedup with respect to 4, 8, and 12 processor systems in tree and butterfly barriers.

proposed UML approach for all cases (4, 8, and 12 processors) are stable and linear with respect to the increase in the memory buffers. As we increase the number of processors in the system, it increases both total remote event messages and the relative speedup for each case.

6. CONCLUSION

This paper presented the implementation of synchronous barriers with the optimistic TMA. This approach is quite new since it combines two different families of algorithms (conservative and optimistic) to go for the same task of synchronization. We started our discussion from the optimistic algorithm in general and Time Wrap and Samadi's algorithms in particular. We also presented an analysis to show why an optimistic algorithm must have the capability to deal with some common problems like rollback, reclaiming memory, transient messages, and simultaneous reporting. Finally, we presented a new UML scheme that solves the transient message problem. To support the implementation of the proposed scheme, two algorithms are presented. Both our theoretical analysis and simulation results suggest that the tree barrier performs well with the UML scheme than the pure optimistic algorithm in terms of the number of synchronization messages that need to be transmitted to compute the GVT value.

REFERENCES

[1] Fujimoto, R. (2003). Parallel simulation: distributed simulation system, *Proceedings of the 35th Winter Simulation Conference*, 124-134

[2] Fujimoto, R. (2000). *Parallel and Distributed Simulation Systems*, John Wiley and Sons

[3] Chandy, K.; Misra, J. (1979). A case study in the design and verification of distributed programs, *IEEE Transactions on Software Engineering,* Vol. 5, No. 5, 440-452

[4] Bryant, R. (1977). Simulation of packet communication architecture computer systems, *Technical Report: TR-188*, Massachusetts Institute of Technology, Cambridge, MA

[5] Jefferson, D. R. (1985). Virtual time, *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, 404-425

[6] Perumalla, K. (2006). Parallel and distributed simulation: traditional techniques and recent advances, *Proceedings of the 38th Winter Simulation Conference*, 84-95

[7] Jeschke, M.; Ewald, R.; Park, A.; Fujimoto, R.; Uhrmacher, A. (2008). A parallel and distributed discrete event approach for spatial cell-biological simulations, *Special Issue on the Quantitative Evaluation of Biological Systems,* Vol. 35, No. 4, 22-31

[8] Peschlow, P.; Martini, P. (2007). Efficient analysis of simultaneous events in distributed simulation, *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications*, 244-251

[9] Madl, G.; Dutt, N.; Abdelwahed, S. (2007). Performance estimation of distributed real-time embedded systems by discrete event simulations, *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software*, 183-192

[10] Liu, Q.; Wainer, G. (2008). Lightweight Time Warp - a novel protocol for parallel optimistic simulation of large-scale DEVS and Cell-DEVS models, *Proceedings of the 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, 131-138

[11] Roberts, D.; Simoni, D. (2007). A teragrid-enabled distributed discrete event agent-based epidemiological simulation, *Proceedings of the 39th Winter Simulation Conference*, 1551-1554

[12] Rizvi, S. S.; Elleithy, K. M.; Riasat, A. (2008). A new mathematical model for optimizing the performance of parallel and discrete event simulation systems, *Proceedings of the 2008 Spring Simulation Multi-Conference,* Article No.: 2

[13] Lees, M.; Logan, B.; Dan, C.; Oguara, T.; Theodoropoulos, G. (2006). Analysing the performance of optimistic synchronisation algorithms in simulations of multi-agent systems, *Proceedings of the 20th Workshop on Principles of Advanced and Distributed Simulation*, 37-44

[14] Mattern, F. (1993). Efficient algorithms for distributed snapshots and global virtual time approximations, *Journal of Parallel and Distributed Computing*, Vol. 18, No. 4, 423-434
[15] Bauer, D.; Yaun, G.; Carothers, C.; Kalyanaraman, S. (2005). Seven-O'clock: a new distributed GVT algorithm using network atomic operations, *19th Workshop on Principles of Advanced and Distributed Simulation*, 39-48
[16] Mattern, F.; Mehl, H.; Schoone, A.; Tel, G. (1991). Global virtual time approximation with distributed termination detection algorithms, *Technical Report: RUU-CS-91-32*, Department of Computer Science, University of Utrecht, The Netherlands
[17] Jefferson, D. R. (1990). Virtual time II: the cancelback protocol for storage management in distributed simulation, *Proceedings of the 9th Annual ACM Symposium on Principle of Distributed Computation*, 75-90
[18] Chen, G.; Szymanski, B. (2007). 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, No. 4, 423-436
[19] Samadi, B. (1985). *Distributed simulation, algorithms and performance analysis (load balancing, distributed processing)*, PhD Thesis, Computer Science Department, University of California, Los Angeles
[20] Bellenot, S. (1990). Global virtual time algorithms, *SCS Multi-Conference on Distributed Simulation*, 122-127
[21] Das, S.; Sarkar, F. (1995). A hypercube algorithm for GVT computation and its application in optimistic parallel simulation, *Proceedings of the 28th Annual Simulation Symposium*, 51-60
[22] Leye, S.; Uhrmacher, A.; Priami, C. (2008). A bounded-optimistic, parallel beta-binders simulator, *Proceedings of the 2008 12th IEEE/ACM International Symposium on Distributed Simulation and Real-Time Applications*, 139-148
[23] Chandy, K.; Sherman, R. (1989). Conditional event approach to distributed simulation, *Proceedings of Distributed Simulation Conference Distributed Simulation*, 93-99
[24] Fujimoto, R.; McLean, T.; Perumalla, K.; Tacic, I. (2000). Design of high performance RTI software, *4th Workshop on Parallel and Distributed Simulation and Real-Time Applications*, 89-96
[25] Fujimoto, R.; Hybinette, M. (1997). Computing global virtual time in shared-memory multiprocessors, *ACM Transactions on Modeling and Computer Simulation*, Vol. 7, No. 4, 425-446
[26] Xiao, Z.; Gomes, F.; Unger, B.; Cleary, J. (1995). A fast asynchronous GVT algorithm for shared memory multiprocessor architectures, *9th Workshop on Parallel and Distributed Simulation*, 203-208
[27] Choe, M.; Tropper, C. (1998). An efficient GVT computation using snapshots, *Proceedings of Computer Simulation Methods and Applications*, 33-43
[28] Perumalla, K.; Fujimoto, R. (2001). Virtual time synchronization over unreliable network transport, *15th Workshop on Parallel and Distributed Simulation*, 129-136
[29] Srinivasan, S.; Reynolds, P. (1993). Non-interfering GVT computation via asynchronous global reductions, *Proceedings of the 25th Winter Simulation Conference*, 740-749
[30] Baldwin, R.; Chung, M.; Chung, Y. (1991). Overlapping window algorithm for computing GVT in Time Warp, *11th International Conference on Distributed Computing Systems*, 534-541
[31] Souza, L. M. D.; Fan, X.; Wilsey, P. A. (1994). pGVT: an algorithm for accurate GVT estimation, *8th Workshop on Parallel and Distributed Simulation*, 102-109
[32] Fujimoto, R. (1990). Performance of Time Warp under synthetic workloads, *Proceedings of the SCS Multi-Conference on Distributed Simulation*, Vol. 22, No. 1, 23-28
[33] Wang, J.; Tropper, C. (2007). Optimizing time warp simulation with reinforcement learning techniques, *Proceedings of the 39th Winter Simulation Conference*, 577-584
[34] Perumalla, K. (2007). Scaling Time Warp based discrete event execution to 10^4 processors on a Blue Gene supercomputer, *Proceedings of the 4th International Conference on Computing frontiers*, 69-76
[35] Chen, G.; Szymanski, B. (2005). DSIM: scaling time warp to 1,033 processors, *Proceedings of the 37th Winter Simulation Conference*, 346-355