Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

CAUSAL ORDER BASED TIME WARP: A TRADEOFF OF OPTIMISM

Yi Zeng
Wentong Cai
Stephen J. Turner

School of Computer Engineering
Nanyang Technological University
Singapore 639798 SINGAPORE

ABSTRACT

The optimistic synchronization paradigm, Time Warp, allows logical processes to advance aggressively. In the circumstances where the violation of the local causality constraint (*LCC*) is prone to occurring, this optimism may introduce substantial rollbacks and, as a consequence, significant overhead in recovering from erroneous computations. In this paper, a new approach, *COBTW*, is proposed, where the happen before relation is employed to capture the potential violations of *LCC* and causal order is applied to regulate the advancement of logical processes. Due to the difference between causal order and time-stamp order, there are discrepancies between them. Solutions to remove the discrepancies are proposed. Experiments conducted in a cluster and an emulated *WAN* suggest that *COBTW* reduces rollbacks caused by violations of *LCC* and empirically results in better performance, in comparison with the Time Warp protocol.

1 INTRODUCTION

Parallel and Distributed Simulation (*PADS*) requires that the advancement of each logical process (*LP*) ultimately obeys the rule known as the local causality constraint (*LCC*), i.e., events are processed in non-decreasing time-stamp order (*TSO*). To achieve this goal, a considerable number of synchronization algorithms have been proposed and applied, and they roughly fall into two categories, namely, conservative synchronization and optimistic synchronization. Conservative synchronization strictly prohibits any violations of *LCC* using well defined protocols (Bryant 1984, Chandy and Misra 1979). An *LP* is allowed to process a certain future event and advance its simulation time only if it is safe to do so. Optimistic synchronization, specifically Time Warp (Jefferson 1985), risks potential violations of *LCC* by allowing *LPs* to process future events in an aggressive way. Once a violation really happens, it is detected at runtime and recovery mechanisms are employed accordingly to cancel wrong computations to guarantee *LCC*.

A "fair" quantitative performance comparison between these two synchronization mechanisms is generally hard because of a high degree of interweaving of influencing factors, which are introduced not only by synchronization and underlying communication models, but also by specific applications (Ferscha 1996, Fujimoto 1999). Conceptually, the performance of conservative synchronization is highly determined by each *LP*'s lookahead. Because of rigid adherence to *LCC*, conservatively synchronized *LPs* probably suffer from over pessimism, which means a potential loss of parallelism in processing non-causally-related events. The performance of optimistic synchronization relies on balanced local virtual time (*LVT*) advancement among *LPs*. By exploiting the maximum parallelism among *LPs*, *LPs* in Time Warp simulation may, on the other hand, suffer from over optimism, which means a noteworthy portion of unwanted rollbacks and flooding of anti-messages.

The happen before relation (Lamport 1978), denoted by $\rightarrow$, is a fundamental relationship among events in distributed systems, which has been widely studied for decades. A message-passing distributed system is viewed as a set of N sequential processes $P_1, P_2, \ldots, P_N$. These processes do not share common memory and have no global clock. The behavior of process $P_i, 1 \leq i \leq N$, is modelled as a sequential occurrence of local events, denoted by E_i. The sequence E_i contains three types of events, namely, changing P_i's local state, sending and receiving messages. Let E denote the set of all events in the distributed system. The relation $\rightarrow \subseteq E \times E$ is the smallest transitive relation satisfying: (i) If $a, b \in E_i$ and a occurs before b, then $a \rightarrow b$; (ii) If $a \in E_i$ is the sending event of a message and $b \in E_j$ is the corresponding receiving event, then $a \rightarrow b$. If neither $a \rightarrow b$, nor $b \rightarrow a$, then a and b are concurrent events, denoted by $a \parallel b$.

The happen before relation can be captured using weak clocks, e.g., Lamport time (Lamport 1978) and strong clocks,

e.g., causal history, vector time and matrix clock (Raynal and Singhal 1995, Schwarz and Mattern 1994). Moreover, strong clocks have been utilized to implement causal order (*CO*) (Cai, Lee, and Zhou 2002, Prakash, Raynal, and Singhal 1997, Raynal, Schiper, and Toueg 1991, Schiper, Eggli, and Sandoz 1989), which means for any two messages m_1 and m_2 that are sent to the same destination P_i, if $S_{m_1} \rightarrow S_{m_2}$ (S_m and D_m denote the events of sending and processing of message m respectively), m_1 must be processed (delivered) at P_i before m_2, i.e., $D_{m_1} \rightarrow D_{m_2}$ must be guaranteed at P_i. Note that the happen before relation is also applicable to messages. $m_1 \rightarrow m_2$ means $S_{m_1} \rightarrow S_{m_2}$.

The happen before relation has been employed for different purposes in Time Warp simulation. In solving the simultaneous events problems (Fujimoto 1999), the happen before relation can be used to break the tie to prevent potential infinite rollbacks from happening and to guarantee a repeatable simulation sequence (Rönngren and Liljenstam 1999). To address inefficiencies in Time Warp's rollback mechanism, the happen before relation is used to detect dependencies in cascading and inter-related events and help fast cancellation, thereby saving a huge amount of time spent in futile computations (Chetlur and Wilsey 2001), e.g., to determine an event which is to be eventually cancelled at an early stage and avoid useless computational efforts.

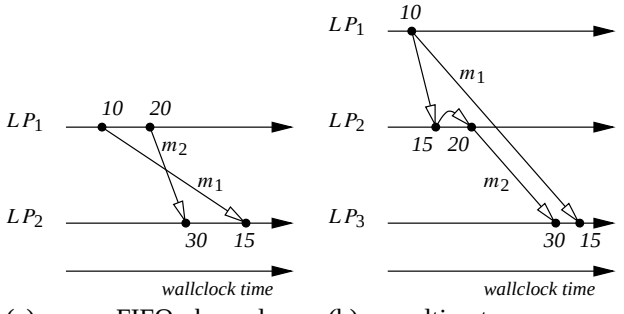
Although the happen before relation can help reduce the chances and the cost of rollbacks, it is only used in (Chetlur and Wilsey 2001) to eliminate intermediate causes, namely, rollbacks caused by receipt of anti-messages. The root cause of rollbacks, namely, the chances that straggler messages are received, remains untouched. In the presence of non-FIFO channels or multicast communication among *LPs*, our experiments show that the chances of receiving straggler messages are even higher. In this paper, we step further to incorporate *CO* into Time Warp. With the adherence to *CO*, messages delayed and disordered by the channel can be fully detected. Based on this knowledge, potential rollbacks caused by out of order processing of causally related messages can be eliminated.

The remainder of this paper is organized as follows: Section 2 describes the motivation for our approach. Section 3 elaborates causal order based Time Warp (*COBTW*) in detail. A variant of Mattern's global virtual time (*GVT*) computation algorithm (Mattern 1993) based on *CO* is also proposed. Experiment results and a comparison between Time Warp and *COBTW* are discussed in Section 4 and Section 5. We reach our conclusion in Section 6.

2 MOTIVATION

In a Time Warp simulation, the processing of messages is always carried out in *TSO* of the messages that have been received but have not been processed. On one hand, this aggressive approach exploits the maximum parallelism among events, i.e., events scheduled by messages are executed with less constraints; on the other hand, it risks additional computational efforts in cancelling wrongly scheduled events. Once the cancellation becomes overwhelming, the so-called over optimism, the advancement of *LPs* in the simulation could be heavily hindered. It in turn harms the simulation's performance dramatically.

Figure 1(a) shows a scenario in which a straggler message is received through a non-FIFO channel. LP_1 processes events $e_{1,10}$ and $e_{1,20}$ and schedules events $e_{2,15}$ and $e_{2,30}$ by messages m_1 and m_2 respectively (To simply the representation, $e_{i,t}$ denotes the event scheduled at LP_i with time-stamp t). Messages m_1 and m_2 have the happen before relation, i.e., $m_1 \rightarrow m_2$, nevertheless they arrive at LP_2 out of order. When LP_2 receives m_1 and processes $e_{2,15}$ scheduled by m_1, it realizes m_1 is a straggler message because it has already advanced its *LVT* to 30 as the result of processing an earlier received event $e_{2,30}$ scheduled by m_2. Figure 1(b) shows another scenario in which a straggler message is received in a multicast enabled simulation. LP_1 multicasts a message m_1 which schedules events $e_{2,15}$ and $e_{3,15}$ at LP_2 and LP_3 respectively. Note that it is assumed here that a multicast message always schedules events with identical time-stamps. As a result of processing event $e_{2,15}$, LP_2 schedules an internal event $e_{2,20}$ which subsequently schedules another event $e_{3,30}$ and sends it to LP_3 by message m_2. It is clear that $m_1 \rightarrow m_2$, but they arrive at LP_3 out of order due to different channels through which they traverse. Similar to the previous scenario, when LP_3 receives m_1 and processes $e_{3,15}$ scheduled by m_1, it detects m_1 is a straggler message because it has already advanced its *LVT* to 30 as the result of processing $e_{3,30}$ scheduled by m_2.



(a) non-FIFO channel (b) multicast

Figure 1: Straggler Messages in Different Environments

Contemporary Time Warp has a straight and simple approach in processing incoming messages. It has no preventive measures to detect potential straggler messages as shown in Figure 1 to prevent rollbacks from happening. In a distributed simulation, especially one where *LPs* communicate via non-FIFO channels with noteworthy delay and

jitter or where multicast messages are common, the chances of the above mentioned straggler messages are expected to be high. Once rollbacks occur, due to the complexity of the communication pattern, the computational efforts of cancellation are also expected to be significant.

With the help of *CO*, a number of straggler messages could be avoided, thus the number of rollbacks could be reduced. Specifically in Figure 1(a)/(b), when message m_2 is received, LP_2/LP_3 detects that another message, i.e., m_1 in the figure, that schedules an event with smaller time-stamp is still on its way. Therefore, m_2 is delayed until m_1 is received and processed.

Note that *COBTW* does not change the ultimate processing order of events, namely, *TSO*. *CO* can be viewed as a way to regulate *LPs*' optimism by means of checking events' causalities. A violation of *CO* indicates a potential violation of *LCC*, during which the specific *LP* must stop its advancement and wait for the late arrived straggler message(s). Conceptually, there exist discrepancies between *CO* and *TSO*. This issue is addressed in Section 3.3.

3 COBTW

3.1 System Model

Similar to the model of a distributed system described in Section 1, *COBTW* is assumed to be running on top of loosely coupled *LPs*. A simulation is composed of N sequential event driven *LPs*, denoted by $LP_1, LP_2, \ldots, LP_N$. These *LPs* do not share memory and operate asynchronously in parallel. Interactions among *LPs* are modelled by exchanging application specific basic messages (Mattern 1993) carrying time-stamped events and protocol messages carrying specific protocol-oriented data, e.g., for *GVT* computation, through corresponding transmission channels. These channels are assumed reliable, but FIFO is not required, i.e., messages may not arrive at their destination *LP* in the order that they were sent and may suffer from arbitrary non-zero delays. Each *LP* exhibits a lookahead, denoted by $L_i, L_i \geq 0, 1 \leq i \leq N$, which means LP_i at simulation time T_i will not schedule new events earlier than $T_i + L_i$. Lookahead is application dependent and could have a zero value.

An *LP*'s internal structure is similar to the architecture of an optimistic logical process depicted in (Ferscha 1996). Each *LP* maintains three queues, namely, the input queue (*IQ*), the output queue (*OQ*) and the state queue (*SQ*). Messages (Without explicit note, message means basic message afterwards) received from the incoming channel are enqueued in the *IQ* and ordered by the time-stamps of events carried by them. The *LP* continuously processes unprocessed events from the *IQ* and can schedule new events by outgoing messages. Apart from being sent through the outgoing channel, the outgoing messages are also duplicated and enqueued in the *OQ*. Meanwhile, the changes in the *LP*'s state are enqueued in the *SQ*. Upon the receipt of a straggler message or an anti-message, the *LP* is involved in a three-step rollback action, namely, recovering back to an old state by loading the appropriate one from the *SQ*, removing erroneous computations by sending anti-messages of those to be cancelled in the *OQ*, and restarting the processing of messages in the *IQ* from the point where the *LP* rolls back.

Diagrammatically, the advancement of *LPs* can be drawn as a tree (Nicol and Liu 1996) as shown in Figure 2. Each *LP* starts as a single branch. A new branch ramifies at the point of each rollback. The extent of each branch indicates the distance the *LP* advances before being rolled back. The black dots denote the events that occur in the simulation. The dashed branches represent the computations which have already been cancelled. The solid branches, with the arrows at their right ends, define the effective paths of the advancement (EPA_i) along which *LPs* evolve. Note that in a simulation, LP_i exactly has one *EPA*, denoted by EPA_i.

Observation 3.1. When a simulation terminates, events on each *LP*'s *EPA* define the effective computations.

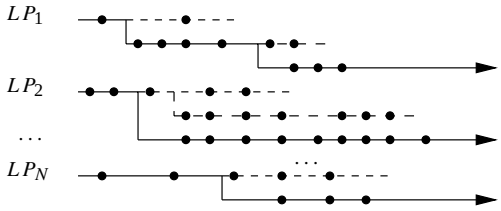


Figure 2: The Advancement of LPs

3.2 Processing Messages in CO

To simplify the explanation of the *COBTW* mechanism, the straightforward implementation (Raynal, Schiper, and Toueg 1991) of *CO* is adopted and only the handling of external messages, i.e., basic and protocol messages that were not destined to the sender *LP* itself, is discussed. Internal messages are sent, received and processed in the same way as Time Warp.

The clock maintained by LP_i has four components, namely, T_i, VT_i, $DELIV_i$ and $SENT_i$. T_i is a scalar value that denotes LP_i's *LVT*. VT_i is an N-tuple where $VT_i[i]$ denotes the number of external messages that LP_i has sent and $VT_i[j], i \neq j$ denotes LP_i's knowledge of another LP_j's $VT_j[j]$. $DELIV_i$ is an N-tuple where $DELIV_i[j], i \neq j$ represents the number of external messages sent from LP_j and processed by LP_i. $SENT_i$ is an $N \times N$ matrix and $SENT_i[j][k]$ shows LP_i's knowledge of the number of external messages sent from LP_j to LP_k (not necessarily processed). Similarly, the clock carried by an external message also has three components, namely, T, VT and $SENT$.

T is the time-stamp at which the carried event is scheduled to be processed. VT and $SENT$ are the snapshots of the sending LP's VT and $SENT$ at the time the message is sent out. Among these components, $DELIV$ and $SENT$ are necessary for the implementation of *CO*. VT is essential to determine the happen before relation between any two external messages (Schwarz and Mattern 1994).

The updating of LP_i's clock is performed at the instant LP_i sends and processes external messages, which are shown in Figure 3 and Figure 4 respectively, where sup denotes the componentwise maximum operation.

```
SEND(M,T,DestLPs) BEGIN
   M.SENT=SENT_i;
   FORALL LP_j ∈ DestLPs DO
      SENT_i[i][j]++;
   VT_i[i]++;
   M.VT=VT_i;
   M.T=T_i;
   send M through channel;
END
```

Figure 3: Send an External Message by LP_i

```
PROCESS(M,LP_src) BEGIN
   WAIT(∀j, DELIV_i[j] ≥ M.SENT[j][i]);
   T_i=M.T;
   DELIV_i[src]++;
   SENT_i[src][i]++;
   SENT_i=sup{SENT_i, M.SENT};
   VT_i[i]++;
   VT_i=sup{VT_i, M.VT};
   process event e carried by M;
END
```

Figure 4: Process an External Message by LP_i

3.3 Discrepancies between CO and TSO

It is apparent that the elements of VT_i increase monotonically and the sending order of messages determines their happen before relation, which subsequently defines their receiving order. In other words, if external messages m_1 and m_2 are from the same LP and m_1 was sent before m_2, then $m_1 \rightarrow m_2$, which means m_1 must be processed before m_2 to be coherent with *CO*. This principle is almost consistent with paradigms in Time Warp simulations, which are shown as Observation 3.2.

Observation 3.2. Assume that there are no optimizations considered. Then:

- An anti-message is always sent after and required to be processed after its corresponding positive message.
- A positive message sent again after cancelling its previous sending is always required to be processed after the latter.

However, there are still discrepancies due to the fact that *TSO* requires time-stamped messages to be processed in the order of their time-stamps, not necessarily in their sending sequence.

Figure 5(a) shows a scenario sometimes found in simulations. LP_1 processes events $e_{1,10}$ and $e_{1,20}$ and schedules events $e_{2,30}$ and $e_{2,25}$ by messages m_1 and m_2 respectively. The difference, compared with Figure 1(a), is that LP_1 schedules events in a non-monotonically ascending way, i.e., m_2 is sent after m_1 but schedules an event with smaller time-stamp. Under this circumstance, *CO* inevitably fails because m_2 is supposed to be processed before m_1 according to *TSO* but forcibly delayed after m_1 due to *CO*. This kind of failure can exist even if messages are sent to different *LPs*. In Figure 5(b), LP_1 processes event $e_{1,10}$ and sends LP_3 message m_1 with event $e_{3,30}$. And LP_2 processes $e_{2,16}$ scheduled by m_2 and schedules an internal event $e_{2,20}$ which subsequently sends LP_3 message m_3 with event $e_{3,24}$. It is obvious that $m_1 \rightarrow m_3$. But unfortunately, they cannot be processed in *CO* at LP_3 for the same reason.

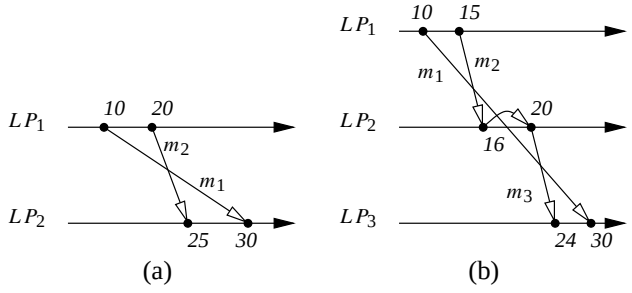


Figure 5: CO Conflicts with TSO

Recalling Observation 3.1, solutions to remove the discrepancies are governed by Observation 3.3 given below.

Observation 3.3. *CO* is said to be consistent with *TSO* iff when a simulation terminates, for all i, $1 \leq i \leq N$, the *CO* of the external messages processed on EPA_i is coherent with the *TSO* of the events scheduled by those messages. That is, for any two events e_{i,t_1} and e_{i,t_2} on EPA_i, scheduled by external messages m_1 and m_2 respectively, if $m_1 \rightarrow m_2$, it must hold that $t_1 \leq t_2$.

Observation 3.3 conceptually defines an approach to remove the discrepancies between *CO* and *TSO* by guaranteeing the coherence among the external messages and the scheduled events which perform effective computations. However, it is empirically hard because before a simulation terminates, these messages and events are generally unknown. An alternative approach is by guaranteeing the coherence among all the external messages and the scheduled events at any time in a simulation. This can be done

locally at each *LP* by ensuring no discrepancies among the outgoing messages and the events it generates. Before an *LP* sends an external message m_2 carrying an event with time-stamp t, it always checks the *OQ*. If there exists an outgoing message m_1 that carries a larger time-stamped event, m_1 must first be cancelled by sending its anti-message and re-sent after the sending of m_2. Specifically in Figure 5(a), when LP_1 processes $e_{1,20}$, LP_1 cancels m_1, sends m_2 and re-sends m_1 again to guarantee the consistency. Similarly in Figure 5(b), when LP_1 processes $e_{1,15}$, LP_1 cancels m_1, sends m_2 and re-sends m_1 again. Because now $m_1 \parallel m_3$, the discrepancy is removed.

The correctness of the above approach can be proved by the transitiveness of *CO* and one of the basic characteristics of Time Warp, i.e., an *LP* always schedules events in its future. Suppose there are two events e_{i,t_1} and e_{i,t_2} scheduled at LP_i by messages m_1 and m_2, where $m_1 \rightarrow m_2$. It holds that $t_1 \leq t_2$ according to the above mentioned sending scheme, because along a chain of intermediate causally related messages between m_1 and m_2, the time-stamps of scheduled events are always guaranteed to be increased monotonically.

The drawback of the above approach is the introduction of additional rollbacks brought about by the removal of discrepancies between *CO* and *TSO*. Compared with the same paradigms in Time Warp simulation, the approach incurs the additional sending of m_1 and its anti-message in both Figure 5(a) and (b). It seems paradoxical to the ultimate goal of the introduction of *CO*, removing unwanted rollbacks. However, experiments in Section 4 show that a substantial reduction of rollbacks still can be obtained.

3.4 CO based GVT Approximation

GVT is computed as the minimal time-stamp of all unprocessed messages in a simulation at a certain time. *GVT* is essential for the periodical memory reclamation, known as fossil collection, from *LPs*' working queues. Two major problems, namely, the transient message problem and the simultaneous reporting problem, make the *GVT* computation naturally challenging (Fujimoto 1999).

Mattern's *GVT* approximation algorithm is based on the concept of a consistent cut, which divides the simulation into past and future parts and guarantees that there are no messages sent from the future of the sending *LP* to the past of the receiving *LP*. After the first consistent cut C_1 is constructed, the second one C_2 is constructed in a way that ensures there are no messages sent prior to C_1 that still have not been received, generally resorting to a distributed termination algorithm.

In the presence of *CO*, a simplified *GVT* approximation algorithm that works similarly to Mattern's is shown in Figure 6. A protocol message, namely, *GVT* message (GM) which is sent and received in the same way as basic messages in the simulation, is introduced for the approximation. The approximation involves two phases. In the forwarding phase, LP_1 initially sends a GM to LP_2 and then LP_2 immediately forwards it to LP_3. This procedure continues and this phase ends when the GM arrives at LP_N. In the backward phase, the GM is sent in the opposite way to that in the forwarding phase and finally reaches LP_1. The forwarding path and the backward path effectively define the consistent cuts C_1 and C_2 respectively. Thus, the computation of *GVT* along C_2 can be done in the same way as that in Mattern's *GVT* approximation algorithm. Compared with Mattern's *GVT* approximation algorithm, the major simplifications in the above approach are in the construction of C_1 and C_2. Neither color messages nor a distributed termination algorithm is required. Due to *CO*, any messages, e.g., m_1, sent prior to C_1 are guaranteed to be received before C_2.

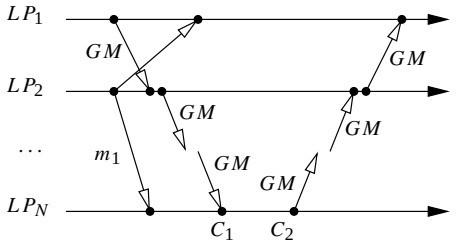


Figure 6: CO based GVT Approximation

4 EXPERIMENTS AND RESULTS

Our test-bed is a beowulf cluster. All the nine PCs (16 Intel Pentium III processors in total) are installed with Linux Redhat6.2 and interconnected through $100M$ ethernet. Simulation software, based on the implementation of the WARPED Time Warp simulation kernel (Wilsey 2000), was developed in C++ to conduct the comparison between Time Warp and *COBTW*. The kernel of the software is mainly composed of three parts (see Figure 7), a collection of basic abstract classes which define the architecture of the Time Warp simulation and two categories of derived classes which extend the definition and implement the Time Warp kernel and the *COBTW* kernel respectively. These two kernels encapsulate the differences between the mechanisms and provide similar APIs. Therefore the experiment application can be executed on either kernel with minimum modifications.

To be general, the parallel hold (*PHOLD*, Fujimoto 1990) model was adopted to artificially represent the simulations in the real world. Five out of the six model parameters specified in Fujimoto (1990) were configured. The number of *LPs* was fixed to 16 so that each *LP* was mapped to a single processor. To test *LPs*' different behaviors under different workloads, the message population, i.e., the number of event scheduling threads, was varied. Each experiment

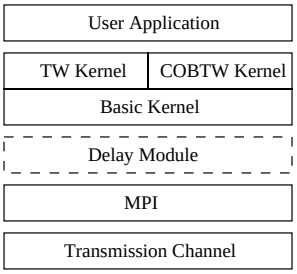


Figure 7: Layout of the Software

was repeated with the message population of 2, 4, 8 and 16 respectively. The time-stamp increment was dynamically set to a random value uniformly distributed in the range of $[0, 10]$. Note that zero increment is allowed in the simulation. The event processing time (computation grain) was set to a random value uniformly distributed in the range of $[1ms, 10ms]$. Because all the *LPs* are assumed to be equally dispersed and fully interconnected, the movement function was simply defined as a uniformly distributed random function. When an *LP* processes an event, any subsequently scheduled event has equal chance to be forwarded to any other *LP* (including the sender itself). To avoid the exponential increase of messages in the case of multicast (this could overload the simulation quickly), it is assumed that only one of the scheduled multicast events can subsequently schedule other events.

There are many criteria in evaluating the performance of Time Warp simulations, among which the number of rollbacks (*NRB*) is of great importance. Intuitively, *NRB* reflects the frequency the *LP* rewinds to its previous states and is proportional to the amount of computations being cancelled with the assumption of the aggressive cancellation strategy. Two sets of experiments were carried out in the cluster and emulated wide area network (*WAN*) respectively to evaluate the rollbacks of Time Warp and *COBTW* in different environments. The normalized speedups achieved by *COBTW* in comparison with the Time Warp are also summarized.

Figure 8 and Figure 9 show the results of the first set of experiments running in the beowulf cluster. *LPs* mapped on the cluster communicate through high speed channels. Moreover, because both Time Warp and *COBTW* kernels are built on MPI (MPIForum 2002), the transmission between any two *LPs* is actually in FIFO order. Figure 8 shows the *LPs'* average rollback behavior with only unicast enabled. Figure 9 depicts the rollback behavior with the configuration that the processing of any event has 0.3 probability to schedule multicast events (the number of destinations of each multicast event is fixed at two). It can be seen that *COBTW* could exhibit slightly higher *NRB* than Time Warp when the message population is low. This is due to the introduction of additional rollbacks (recall the discussion in

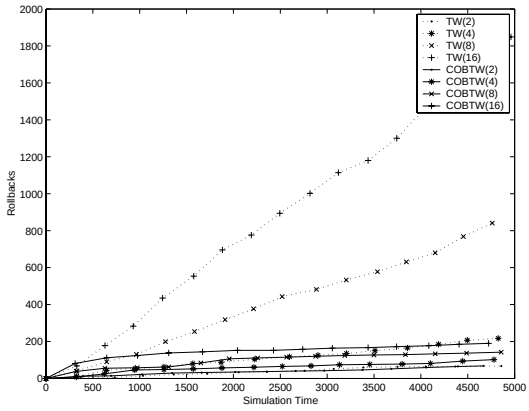


Figure 8: Rollback Comparison of PHOLD Running on Cluster (unicast)

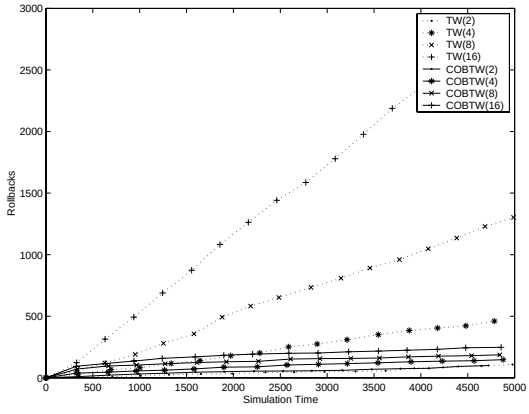


Figure 9: Rollback Comparison of PHOLD Running on Cluster (multicast)

Section 3.3) in maintaining the consistency between *CO* and *TSO*. As the message population increases, the advantage to prevent causal violations becomes more obvious, which offsets the disadvantage of the above mentioned additional rollbacks.

Figure 10 and Figure 11 show the results of the second set of experiments running in an emulated *WAN*. To emulate the latency of *WAN* (normally in the order of milliseconds Bal et al. 1998) and the non-FIFO channels, a delay module, shown as the dashed box in Figure 7, resides in the kernels and the latency suffered by any outgoing message was artificially set to a random value uniformly distributed in the range of $[50ms, 150ms]$. Because of the non-FIFO property, it can be seen that for experiments running on the Time Warp kernel, *NRB* in both unicast and multicast is eight times more than that in the cluster environment. However, because of the ability to prevent many causal violations caused by non-FIFO channels and

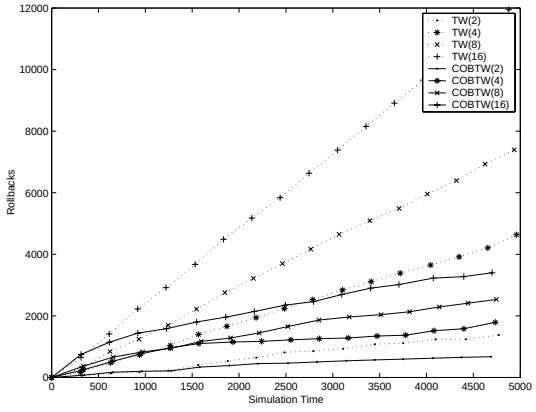


Figure 10: Rollback Comparison of PHOLD Running on Emulated WAN (unicast)

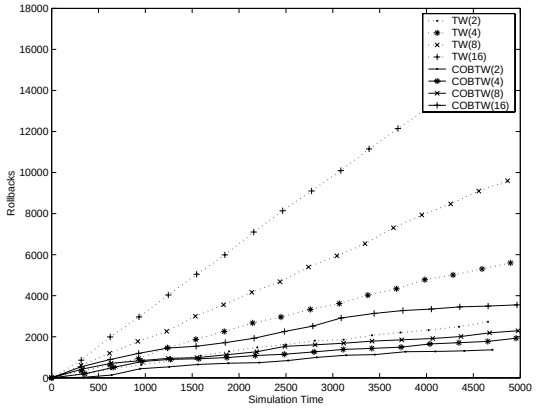


Figure 11: Rollback Comparison of PHOLD Running on Emulated WAN (multicast)

multicast messages, *COBTW* outperformed Time Warp in all cases and kept *NRB* at a relatively low level.

Figure 12 summarizes the normalized speedup, which is defined by the ratio of the execution time of the Time Warp to that of *COBTW*. In either environment, *COBTW* has similar performance to that of Time Warp when the message population is two. As the message population increases, *COBTW* yields better performance proportionally.

5 DISCUSSION

A simple *CO* delivery algorithm (Raynal, Schiper, and Toueg 1991) is currently used in *COBTW*. Compared with Time Warp, external messages in *COBTW* piggyback much more information ($O(N^2)$) to capture the happen before relation and guarantee *CO* among messages. This introduces considerable communication overhead in Time Warp simulations. By using the causal barrier, which captures the immediate causal predecessors of the sent message, instead of the

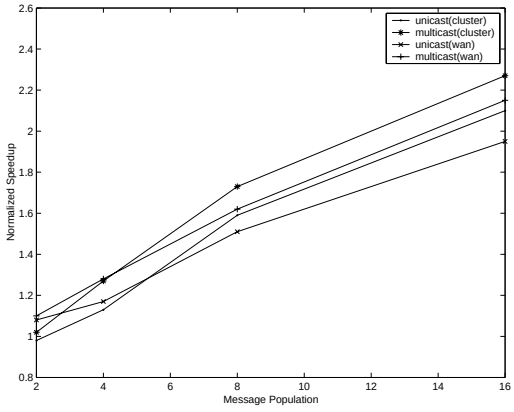


Figure 12: Speedup

$N \times N$ matrix $SENT$, the optimal approaches proposed in (Cai, Lee, and Zhou 2002, Prakash, Raynal, and Singhal 1997) significantly reduce the amount of piggybacked information and therefore, can be employed to minimize the communication traffic.

Recalling the approach to removing the discrepancies between *CO* and *TSO* (discussed in Section 3.3), some additional rollbacks are introduced and reflected in the result shown in Figure 8. By taking advantage of *LPs*' lookaheads, the above approach can be further improved to remove the additional rollbacks using the buffer mechanism shown in Figure 13. Suppose that LP_i processes events e_{i,t_1} and e_{i,t_2} $(t_1 < t_2)$ and schedules events $e_{j,t_1+\Delta t_1}$ and $e_{j,t_2+\Delta t_2}$ $(t_1 + \Delta t_1 > t_2 + \Delta t_2)$ by m_1 and m_2 respectively. With the promise of L_i, it holds that $\Delta t_1 \geq L_i$ and $\Delta t_2 \geq L_i$. Therefore, LP_i can delay (buffer) the sending of m_1 and m_2 by scheduling itself two internal events $e_{i,t_2+\Delta t_2-L_i}$ and $e_{i,t_1+\Delta t_1-L_i}$. Because $t_2 + \Delta t_2 - L_i < t_1 + \Delta t_1 - L_i$, m_2 is actually sent before m_1. Thus, the happen before relation between m_1 and m_2 are effectively guaranteed to be consistent with their time-stamps.

So far, the cancellations of messages are done in the traditional way in *COBTW*, i.e., one anti-message exactly cancels one positive message. Because of the uncertainty about the exact events to be eventually cancelled at the moment an *LP* rolls back, the cancellation arising from the receipt of a straggler message probably introduces multiple unnecessary rollbacks at *LPs*. With the knowledge of the

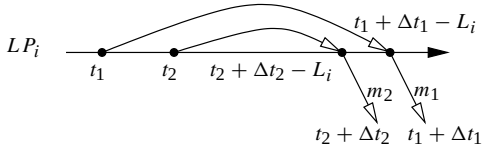


Figure 13: Buffer Outgoing Messages

happen before relation, the exact set of events to be cancelled at all *LPs* could be fully determined. This means that one rollback is enough for each *LP* to cancel the wrong computations caused by a straggler message, hence, providing a promising improvement from the original cancellation strategy. Based on a similar approach to that proposed in Chetlur and Wilsey (2001), relevant modifications are expected to be carried out to take advantage of *CO* in *COBTW*.

6 CONCLUSION

The happen before relation captures basic but important relations among messages (events) in distributed systems. Moreover, processing of messages in *CO* is required in various distributed applications. The happen before relation is a partial order, which means some, but not necessarily all, pairs of messages (events) can be ordered. If a pair of messages (events) cannot be ordered, they are said to be concurrent and can be processed in any sequence. Different from *CO*, discrete event simulation relies on the concept of simulation time and requires that events must be processed in *TSO*. *TSO* can be viewed as a total ordering scheme in which any two events can be ordered upon the two-tuple $< i, t >$, where i is the identification number of the *LP* which processes the event and t is the event's time-stamp.

In this paper, *CO* is introduced into Time Warp simulations to regulate the optimism of the *LPs*. An asynchronous *GVT* algorithm based on *CO* is also proposed. Conceptually, *LPs* could not process messages as optimistically as before. With the guarantee of the consistency between *CO* and *TSO*, a portion of violations of *LCC* can be detected and unsafe events are delayed. Although *CO* cannot eventually ensure the safeness of an event because any subsequently received concurrent events may have smaller time-stamps, substantial rollbacks and cancellations still can be prevented. Our experiments show that *COBTW* has a better result of *NRB* and performance than Time Warp in circumstances where causality violations are prevailing.

REFERENCES

Bal, H., A. Plaat, M. Bakker, P. Dozy, and R. Hofman. 1998, April. Optimizing parallel applications for wide-area clusters. In *International Parallel Processing Symposium*, 784–790.

Bryant, R. E. 1984. A switch level model simulator for MOS digital systems. *IEEE Transactions on Computers* C-33 (2): 160–177.

Cai, W., B. S. Lee, and J. Zhou. 2002. Causal order delivery in multicast environment: An improved algorithm. *Journal of Parallel and Distributed Computing* 62 (1): 111–131.

Chandy, K. M., and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* SE-5 (5): 440–452.

Chetlur, M., and P. A. Wilsey. 2001, May. Causality representation and cancellation mechanism in time warp simulations. In *Workshop on Parallel and Distributed Simulation*, 165–172.

Ferscha, A. 1996. *Handbook of parallel and distributed computing*, Chapter Parallel and Distributed Simulation of Discrete Event Systems, 1003–1041. McGraw-Hill.

Fujimoto, R. M. 1990, January. Performance of time warp under synthetic workloads. In *Proc. Multiconf. Distributed Simulation*, Volume 22, 23–28.

Fujimoto, R. M. 1999. *Parallel and distributed simulation systems*. Wiley Book Series on Parallel and Distributed Computing. New York, NY 10158, USA: Wiley.

Jefferson, D. R. 1985, July. Virtual time. *ACM Transactions on Programming Languages and Systems* 7 (3): 404–425.

Lamport, L. 1978, July. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM* 21 (7): 558–565.

Mattern, F. 1993. Efficient algorithms for distributed snapshots and global virtual time approximation. *Journal of Parallel and Distributed Computing* 18 (4): 423–434.

MPIForum 2002. The Message Passing Interface (MPI) standard. <http://www.mpi-forum.org/>.

Nicol, D. M., and X. Liu. 1996. The Dark Side of Risk (What your mother never told you about Time Warp). Technical Report TR96-298, Dartmouth College.

Prakash, R., M. Raynal, and M. Singhal. 1997, March. An adaptive causal ordering algorithm suited to mobile computing environments. *Journal of Parallel and Distributed Computing* 41:190–204.

Raynal, M., A. Schiper, and S. Toueg. 1991. The causal ordering abstraction and a simple way to implement it. *Information Processing Letters* 39:343–350.

Raynal, M., and M. Singhal. 1995, March. Logical time: A way to capture causality in distributed systems. Technical Report RR-2472, INRIA - Rennes.

Rönngren, R., and M. Liljenstam. 1999, May. On event ordering in parallel discrete event simulation. In *Workshop on Parallel and Distributed Simulation*, 38–45.

Schiper, A., J. Eggli, and A. Sandoz. 1989. A new algorithm to implement causal ordering. *Proc. Workshop on Distributed Algorithms* LNCS Vol. 392:219–232.

Schwarz, R., and F. Mattern. 1994. Detecting causal relationships in distributed computations: In search of the holy grail. *Distributed Computing* 7 (3): 149–174.

Wilsey, P. A. 2000. The WARPED Time Warp simulation kernel. <http://www.ece.uc.edu/~paw/warped/>.

AUTHOR BIOGRAPHIES

YI ZENG is a Ph.D student in School of Computer Engineering at Nanyang Technological University. He received his BSc and Msc in Computer Science from Zhejiang University, P.R.C. His current research topic is on parallel and distributed simulation systems.

WENTONG CAI is an Associate Professor with School of Computer Engineering at Nanyang Technological University (NTU), Singapore, and the head of Computer Science Division. He received his B.Sc. in Computer Science from Nankai University (P. R. China) and Ph.D., also in Computer Science, from University of Exeter (UK). He was a Postdoctoral Research Fellow at Queen's University (Canada) before joining NTU in Feb 1993. Dr. Cai has been actively involved in the research in Parallel and Distributed Computing for more than ten years and has published more than 100 research papers in this area. His current research interests include: Parallel & Distributed Simulation, Cluster and Grid Computing.

STEPHEN JOHN TURNER joined Nanyang Technological University (Singapore) in 1999 and is currently an Associate Professor in the School of Computer Engineering and Director of the Parallel and Distributed Computing Centre. Previously, he was a Senior Lecturer in Computer Science at Exeter University (UK). He received his MA in Mathematics and Computer Science from Cambridge University (UK) and his MSc and PhD in Computer Science from Manchester University (UK). His current research interests include: parallel and distributed simulation, distributed virtual environments, parallel algorithms and languages, grid computing and multi-agent systems.