# THE LOCAL TIME WARP APPROACH TO PARALLEL SIMULATION

Hassan Rajaei, Rassul Ayani, and Lars-Erik Thorelli

Royal Institute of Technology
Department of Teleinformatics
Electrum 204, S-16440 Kista, Stockholm, Sweden
E-mail: {rajaei, rassul, le}@it.kth.se

## ABSTRACT

The two main approaches to parallel discrete event simulation – conservative and optimistic – are likely to encounter some limitations when the size and complexity of the simulation system increases. For such large scale simulations, the conservative approach appears to be limited by blocking overhead and sensitivity to lookahead, whereas the optimistic approach may become prone to cascading rollbacks, state saving overhead, and demands for larger memory space. These drawbacks restrict the synchronization schemes based on each of the two approaches from scaling up. A combined approach may resolve these limitations, while preserving and utilizing potential advantages of each method. However, the schemes proposed so far integrate the two views at the same level, i.e. local to a logical process, and hence may not be able to fully solve the problems. In this paper we propose the Local Time Warp method for parallel discrete-event simulation and present a novel synchronization scheme for it called HCTW. The new scheme hierarchically combines a Conservative Time Window algorithm with Time Warp and aims at reducing cascade rollbacks, sensitivity to lookahead, and the scalability problems. Local Time Warp is believed to be suitable for parallel machines equipped with thousands of processors and thus an appropriate candidate for simulation of large and complex systems.

## 1. INTRODUCTION

Parallel Discrete Event Simulation (PDES) is in general synchronized by one of the two approaches known as conservative and optimistic respectively (Fujimoto 1990b). Performance studies (e.g. Fujimoto 1988; Fujimoto 1990a; Lubachevsky et al 1991; Reynolds et al 1989) show that both of the approaches are susceptible to some limitations. These studies indicate that the conservative method endures when the application exhibits poor *lookahead* in such case it may perform worse than sequential simulation. Accordingly, the optimistic approach becomes exposed to state saving and processing overhead especially when the application imposes excessive rollbacks to the simulation system. In particular when the problem size and the number of processors become large, the risk for explosive cascading rollbacks increases. This situations occurs mainly by processes that rapidly advance far in future virtual time. Cascading rollbacks dramatically deteriorate performance and prohibit the simulation to scale. Moreover, the system may become unstable and need an excessive amount of memory for saving the state variables (Fujimoto 1990; Jefferson 1990; Lubachevsky 1990). To overcome this problem and let the system run on the available memory, Jefferson (1990) strongly recommends incorporating the cancelback protocol in the optimistic scheme. However, the performance degrades considerably when limited memory is available. The synchronization and blocking overhead of the conservative schemes also limit scalability. Furthermore, conservative methods provide only static configuration and thus may not support applications which exploit dynamic structures. Despite these problems, there are interesting advantages for both approaches.

The optimistic approach has the ability of exploiting maximum parallelism, not being sensitive to lookahead, and supporting dynamic structures. The conservative methods generally require less memory and are considered to be able to more efficiently produce deterministic simulation results. Consequently, several attempts have been made to combine the two methods in order to provide more efficient parallel simulation schemes (Arvind and Smart 1992; Davis and Ramachandran 1992, Lubachevsky et al 1989, McAffer 1990, Mehl 1991; Parakash and Subramanian 1992; Turner and Xu 1992). The majority of these schemes add optimism to a conservative algorithm such that rollback for a logical process (LP) is limited either locally to the LP or by an upper-bound that restricts advancement of local virtual time of the LP. Furthermore, all these approaches combine the two views at the same level of LPs and thus they carry to some extent the problems associated with both views. Most notable is the scalability problem which seems to remain unsolved for the existing combined methods (Lubachevsky 1990). Our method specifically aims at addressing this issue and in contrast to the other schemes it hierarchically combines the two synchronization views.

In this paper we propose the *Local Time Warp* (LTW) approach to parallel simulation and present a novel synchronization scheme for it called HCTW. The scheme hierarchically combines the Conservative Time Windows algorithm (CTW) (Ayani and Rajaei 1992) and Time Warp (Jefferson 1985). The simulated system consists of a number of subsystems each of which comprises numerous components. Each subsystem is considered as a cluster and its components as logical processes. At the lower level, Time Warp is used locally for LPs of the clusters in order to exploit parallelism and maintain flexibility. To control cascade rollbacks CTW is used at the higher level. This layered synchronization method preserves the flexibility of the conventional Time Warp – which is referred to Global Time Warp (GTW) in this paper – while preventing the unwanted cascading rollbacks.

The rest of this paper is organized as follows: Section 2 describes the related work and gives an overview of the current hybrid techniques. In section 3 we present the Local Time Warp model and in Section 4 its synchronization scheme, HCTW. Section 5 discusses dynamic configuration. Section 6 gives

some concluding remarks and issues which need further investigation.

## 2. RELATED WORK

Clustering LPs has been supported by some parallel simulation languages such as Sim++ (Baezner et al 1990) and Maisie (Bagrodia and Liao 1990). However, these languages use clustering techniques mainly as a tool for assigning LPs to specific processors and allowing the LPs of a cluster to share some variables. Both intra- and intercluster synchronization are achieved by a single protocol. In contrast, our scheme differentiates between the two such that intracluster activities are synchronized optimistically whereas the intercluster synchronization is done conservatively. Furthermore, intracluster activities can be simulated in parallel both with respect to the cluster and to intercluster activities. In the rest of this section we focus on hybrid synchronization schemes and give an overview of techniques applied by the hybrid protocols proposed so far.

The issue of integrating the two approaches of PDES, although not new, has received more attention in recent years since the limitations of both methods have been more explored. The blocking problem and the sensitivity to lookahead in the conservative protocols, and cascading rollback and state saving problems in the optimistic schemes are the key limitation factors for the respective approaches. Therefore, an intelligent combination of the conservative and the optimistic approaches may be more powerful than either type in pure form (Lubachevsky et al 1991). One solution is to regulate somehow the degree of *risk* and *aggressiveness*, in Reynolds' (1988) terminology, of a protocol. If the degrees of aggressiveness and risk for a logical process $LP_i$ are denoted respectively by $A_i$ and $R_i$ then by regulating $0 \leq A_i \leq \infty$ and $0 \leq R_i \leq \infty$ a conservative, optimistic, or hybrid scheme can be obtained. There are three general methods for integrating the two synchronization approaches: (1) adding optimism into a conservative protocol, (2) limiting optimism, (3) switching seamlessly between optimistic and conservative schemes.

### 2.1 Adding Optimism into Conservative Protocols

To avoid blocking, a conservative protocol may use *speculative* computation such that whenever an LP is to be blocked, it optimistically simulates the first event from its event-list. Two possibilities exists: (1) The degree of aggressiveness is unbounded but the degree of risk is set to zero. (2) The degree of risk is relaxed from zero but the degree of aggressiveness is bounded to a certain limit which implicitly implies a bound on the degree of risk. In the first case the hybrid protocol is still conservative and the result of speculative computation is kept locally until it is certain that no other event will preempt it. If an error is detected while the results are kept, a local rollback will restore the correct order. This approach is adapted in SRADS with local rollback (Dickens and Reynolds 1990), Speculative Simulation (Mehl 1991), and Breathing Time Buckets (Steinman 1992). In the second case the hybrid protocol approaches an optimistic one from a conservative direction. The result is sent out but when an error is detected a limited number of rollbacks will restore the correct order. Filtered Rollback (Lubachevsky et al 1989) falls into this category. It is a combination of Time Warp and the bounded lag algorithm such that the upper-bound of the lag, i.e. the lower bound after which causality error may occur, is relaxed to a certain value which is pre-defined and determined by the user. Lubachevsky et al (1989) theoretically proved that filtered rollback under certain assumptions is scalable. However, Lubachevsky (1990, pp 195) suggests avoiding Time Warp for certain applications since "Check-pointing rollback with global recovery is asymptotically non-scalable. This means that the time to simulate a larger system on a proportionally larger parallel processor grows asymptotically very quickly".

The main disadvantage of adding optimism into conservative protocols at the LP level is that the modified schemes still cannot support dynamic configurations which frequently is needed in simulation of large and complex systems. The scalability of this approach remains questionable. The local rollback method (Dickens and Reynolds 1990, Mehl 1991), although resolving blocking by using speculative computation, may not exploit potential parallelism since the results are kept by each LP until becoming secured. This is improved by Breathing Time Buckets (Steinman 1992) to a group of LPs residing in one node.

### 2.2 Limiting Optimism

In the previous section a conservative protocol approached an optimistic one through relaxing the degree of aggressiveness from zero. From the other direction, an optimistic scheme can be made less optimistic and approach conservative protocols if the degree of risk is bounded (from infinite) to a certain ceiling.

To reduce cascade rollbacks, rollback distance, and state saving overhead of the optimistic protocols, it is possible to prevent the local virtual time of LPs from advancing into the far future. In Wolf (Madisetti et al 1988), whenever a rollback occurs, special messages are broadcasted in order to quickly stop the erroneous computation. Moving Time Window (MTW) (Sokol et al 1988) uses a window boundary to limit the optimistic computation such that a window with pre-defined size is pulled through the virtual time. Bounded Time Warp (BTW) (Turner and Xu 1992) divides the simulation duration time into a number of equally sized intervals. BTW to some extent is similar to MTW but instead of pulling the windows concurrently with the event processing, all events inside the current interval are simulated before the next interval is started. In MIMDIX (Madisetti et al 1992) additional processes called Genie processes are used at regular intervals to probabilistically determine whether to *resynchronize* the LPs in order to slow down the progress rate of their virtual time advancement. If it is decided to do so, then a SYNC message with a timestamp slightly greater than the Global Virtual Time (GVT) is broadcast to all LPs to force them to rollback to the SYNC time. All events with timestamp greater than SYNC are deleted. Some messages are undone to enable restarting of the computation. Unified Distributed Simulation,UDS, (McAffer 1990) uses the degrees of aggressiveness and risk and provides a sliding window mechanism for both sending and receiving messages. Messages with timestamps outside the boundaries of the windows are not sent or received but rather the corresponding LPs are blocked, resulting in a possible deadlock. Reiher and Jefferson (1989) propose two throttling methods to limit rollback: *window-based throttling* and *penalty-based throttling*. The first method explicitly prevents events from executing in the far future. The second method identifies objects whose work has to be more frequently undone and lets them be executed less often. The experimental data presented in (Reiher and Jefferson 1989)

indicates that throttling methods may not show significant improvement in performance.

The primary drawback of limiting optimism at the LP level is to find an appropriate boundary to significantly improve performance. For large scale simulation, it would be a difficult task to predict a good enough boundary to improve performance and not degrade it. MIMDIX (Madisetti et al 1992) uses probabilistically determined intervals for controlling the progress rate of virtual time. To do so, probabilistic knowledge about the behaviour of the system under simulation is needed. If the interval for resynchronization becomes short, LPs are more often forced to synchronize themselves to GVT and hence behave more conservatively. On the other hand, if the interval becomes too large then resynchronization may have no significance. Deleting messages after each SYNC may also become a source of overhead but it seems to reduce the memory limitation of Time Warp.

### 2.3 Switching Between Optimism and Conservatism

It would be desirable to dynamically determine the degree of aggressiveness or risk. In Composite ELSA (Arvind and Smart 1992) a node can switch between conservative and optimistic mode by providing extra information. The information is used to determine whether an event is certain or guessed. Both local and global rollbacks are allowed and there is no notion of GVT. In Adaptive Time Warp, ATW, (Ball and Hoyt 1990) a tuneable *Blocking Window* (BW) is defined such that a processor is blocked for a BW period whenever it experiences an abnormally high number of rollbacks. The BW value is periodically updated during simulation. If BW = 0 (i.e. no blocking period) then the scheme turns to pure Time Warp and when BW = $\infty$ (i.e. block until become certain) the scheme turns to a pure conservative protocol. In CO-OP (Conservative-Optimistic, Steinman 1992) an LP does not always select the first message of its event-list. If some condition is not satisfied then the LP is temporarily blocked waiting for less error-prone messages.

Switching seamlessly between the two approaches is an attractive method especially when the behaviour of the application changes dynamically. Nevertheless, this method requires extra information to be stored, processed, and cross-examined. This can become costly and degrade performance, especially when the problem size and the number of processors increase. The performance of Composite ELSA reported by Arvind and Smart (1992) confirms this conclusion in that the performance dramatically degrades when increasing the number of processors.

## 3. THE LOCAL TIME WARP MODEL

This paper proposes a new combined method for PDES, Local Time Warp (LTW), that preserves the dynamic behaviour of the optimistic approach while limiting cascading rollbacks. The simulation system is decomposed into a number of subsystems interacting through messages. Each subsystem is further decomposed and its components are modeled as LPs according to the Virtual Time paradigm (Jefferson 1985). The subsystems encapsuling LPs are called *clusters*. It is assumed that each cluster encapsulates more than one LP. Further, it is also assumed that the partitioning task is assisted by the user. LPs of a cluster can directly pass messages to each other. Consequently, the *Cluster Internal Network* (CIN) has a dynamic configuration and there is no need to establish links. In contrast, clusters are interconnected through links constituting an Inter-Cluster *Network* (ICN). Nevertheless, clusters and ICN can be reconfigured dynamically under simulation (described in Section 5). Figure 1 illustrates the architecture of LTW with its two level partitioned simulation model.

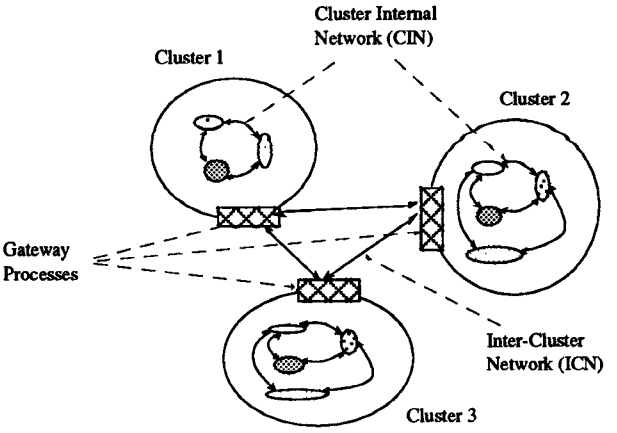


**Figure 1:** Abstract view of the Local Time Warp model. For simplicity only one GP per cluster is shown.

Each cluster (Figure 2) has two *Gateway Processes* (GPs) which are in charge of communication of timestamped messages between clusters in non-decreasing timestamp order. Each LP is uniquely identified throughout the system by an identifier composed of the address of the cluster in which the LP is residing and the internal identifier of the LP. Throughout the simulation system, any LP can send messages to any other LP, assuming the address of the receiver LP is known. When a message is sent, the address of the receiving LP is checked to determine whether the message is internal or external. All external messages of a cluster are automatically directed to its *Output Gateway Process* (OGP) which is in charge of correct message delivery to the destination cluster. Message are sent to the *Input Gateway Process* (IGP) of the receiving cluster to be finally delivered to the designated LP. Messages sent to other clusters should consume non-zero service time at their source LP. Many applications, including communication networks and computer systems, satisfy this constraint.

Each cluster has a *Cluster Virtual Time* (CVT) which resembles GVT in the Global Time Warp method with the difference that CVT is local to the cluster. The virtual time of the two gateway processes of a cluster are called *Input Virtual Time* (IVT) and *Output Virtual Time* (OVT).

**Definition 1:** Let C be the set of all clusters $i$ in the system and $n_i$ be the number of LPs in cluster $i$. Then:

$$\forall_{i \in C}\ CVT_i = \min \{ IVT_i, \min \{ LVT_{i,j} \mid j = 1, 2, \ldots, n_i \} \} \quad (1)$$

where $LVT_{i,j}$ is the *Local Virtual Time* of $LP_{i,j}$ (i.e., LP number $j$ in cluster number $i$ ).

In the above definition it is assumed that messages of a cluster are delivered in zero time within the cluster and the causality effect is immediate. Further, the underlying communication system will preserve the order of message sending. As a result, no messages will be in transit (likewise is assumed in (Jefferson 1990)). This definition can easily be adapted for the case when the above assumption is not true and some messages possibly are in transit when $CVT_i$ is computed.

In this case, a more general algorithm such as those presented in (Bellenot 1990; Lin and Lazowska 1990) can be applied for computing $CVT_i$.

**Rule 1:** For any cluster *i*, the input virtual time, $IVT_i$, is non-decreasing time measure.

**Rule 2:** The output gateway of a cluster *i* can send out a message only when $CVT_i$ becomes greater than or equal to the timestamp of that message, i.e. the message becomes *committed.*

**Definition 2:** Let $CVT_i^k$ denote the *k*th ($k$ = 0, 1, 2, ... ) computation of $CVT_i$ and $t_m$ the timestamp of message *m* at $OGP_i$ ($t_m \le CVT_i^{k+1}$ ). Then the output virtual time of $OGP_i$ is defined as:

$$OVT_i = \begin{cases} CVT_i^k & \text{-before computing } k{+}1\text{:th } CVT_i \text{ and sending out any message} \\ t_m & \text{-after computing } k{+}1\text{:th } CVT_i \text{ and sending out message } m \\ CVT_i^{k+1} & \text{-after computing } k{+}1\text{:th } CVT_i \text{ and sending out all committed messages} \end{cases}$$

**Corollary 1** (from Definition 1) :

$$\forall_{i \in C}\, IVT_i \ge CVT_i \qquad (2)$$

Since $IVT_i$ is a lower bound on timestamps of incoming messages to cluster *i*, this excludes the possibility of arrival of an input message with time less than cluster virtual time.

**Corollary 2** (from Definition 2) : $OVT_i$ is non-decreasing.

**Lemma 1:** Let $t(m_{i,j})$ be the timestamp of a message that is sent from $LP_{i,j}$ to $OGP_i$. Then

$$t(m_{i,j}) \ge OVT_i \qquad (3)$$

**Proof:** Follows directly from Definition 2 since no message can be generated earlier than $CVT_i$. ♦

**Lemma 2:** For any cluster *i* the following relation holds:

$$IVT_i \ge CVT_i \ge OVT_i \qquad (4)$$

**Proof:** $CVT_i$ is non-decreasing, thus Definition 2 implies $OVT_i \le CVT_i$. The result follows from (2). ♦

Relation 4 is fundamental to the Local Time Warp model and dictates to some extent how its synchronization scheme should be organized.

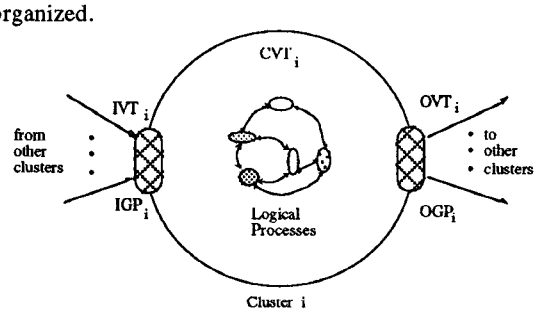


**Figure 2:** Components of cluster *i*.

## 4. SYNCHRONIZATION OF LOCAL TIME WARP

Synchronization of Local Time Warp is achieved at two levels: *logical process level* and *cluster level.* At the logical process level, LPs are synchronized according to the Time Warp protocol. At the cluster level, relation 4 is enforced by a conservative algorithm to guarantee that no rollback occurring in a cluster propagates and triggers chain reactions into another cluster. This is achieved by not allowing any message ever to arrive at the input gateway of a cluster with timestamp less than the time of that gateway. In this section we propose a novel scheme called HCTW to synchronize the LTW model. The scheme adapts Time Warp at LP level and uses a modified version of the CTW algorithm (Ayani and Rajaei 1992) at the cluster level.

### 4.1 Logical Process Level Synchronization

The logical processes of a cluster simulate their local events in the same way as the Global Time Warp. The LPs communicate messages resulting in possible rollbacks. The global virtual time is replaced by the cluster virtual time and fossil collection is performed accordingly. All messages for destinations external to a cluster are automatically passed to its output gateway instead of being directly sent to their destination LPs. The messages are kept in the gateway process until $CVT_i$ becomes greater than or equal to their timestamps. Until then, the messages are subject to possible rollback and hence are not sent out. Consequently, messages at the output gateway are either safe or unsafe. Safe messages fall in a time window bounded by $OVT_i$ and $CVT_i$ (Figure 3) and cannot be rolled back. Unsafe messages can, however, be rolled back and in the worst case back to $CVT_i$. The cluster virtual time is advanced by computing its new value at regular intervals or according to an alternative scheme, e.g. in each iteration of the HCTW algorithm. At the input gateway, the HCTW algorithm identifies a time window over messages sent from the predecessor clusters. The boundaries of the input window are respectively the current and new values of the input virtual time. Messages of the input window are delivered to their destination logical processes resulting in possible rollbacks. However, the rollbacks are limited to the cluster and to the lower bound of $CVT_i$, since any predecessor output gateway should also send out only safe messages. When the committed messages of the output gateway are delivered, $OVT_i$ is advanced to $CVT_i$.

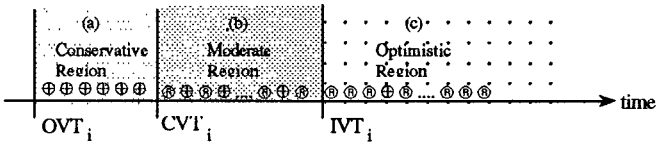


**Figure 3:** Operational regions of a cluster: region (a) is conservative, region (b) is moderate, and region (c) is optimistic.

In principle, the operational zone of a cluster can be divided into three regions (Figure 3): *conservative, moderate,* and *optimistic.* In the conservative region, no event is rolled back and hence can be committed. In the moderate region and no event-message can arrive from IGP with a timestamp less than $IVT_i$ (according to Rule 2). Consequently, in this region only a *moderate* number of events, activated indirectly or independently of IGP, are *risky* and subject to possible rollback. In the optimistic region, the probability of rollback to occur by new stragglers arriving from IGP is rather considerable. A few LPs may operate most of their time in the optimistic region. LTW limits the impact of such LPs to the cluster they are residing in and thus no other clusters are affected. Furthermore, it is possible to introduce an optional ceiling for the optimistic region of each cluster to further limit rapid progress of local virtual time of the LPs. Although it is

possible to integrate the cancelback protocol (Jefferson 1990, Lin 1992) at this level, we believe this is unnecessary due to the fact that LTW considerably limits cascaded rollbacks. Furthermore, because of the far smaller number of LPs in a cluster than in the global system CVT becomes computationally more cost effective than the global counterpart, GVT. Therefore, it is possible to more frequently compute CVT and hence keep the length of the state queue, i.e. the queue holding saved states of the LP, under control. Moreover, in the case of local bottleneck the simple optional ceiling mentioned above can further limit the local rollback distance, hence avoiding the overhead of the cancelback protocol.

### 4.2 Cluster Level Synchronization

The gateway processes of a cluster are controlled by a modified version of the Conservative Time Window algorithm called the HCTW algorithm (Figure 4). For each $IGP_i$, the algorithm identifies an input time window $IW_i = [L_i, U_i]$ where $L_i$ = current value of $IVT_i$ and $U_i$ = next value of $IVT_i$. Messages inside the window are delivered to their destination. Likewise, boundaries of the output time window, $OW_i$, will be determined with the difference that the HCTW algorithm indirectly determines the upper-bound of the window which is equal to the new value of $CVT_i$. As a result, at the beginning (and at the end) of each HCTW iteration, $OVT_i$ becomes equal to $CVT_i$. The following parameters are defined for the algorithm presented in Figure 4: Let C be the set of all clusters $i$ in the system, $Pred_i$ be the set of all immediate predecessors of $IGP_i$, $Suc_i$ be the set of all immediate successors of $OGP_i$, $IW_i$ be the time window for $IGP_i$, $\delta_i$ be the minimum time distance from any LP in cluster $i$ to any other LP in its immediate successor clusters, and

$$\tau_i = \begin{cases} t(e_i) & \text{time of the first message in } IGP_i \\ \infty & \text{if the input queue is empty} \end{cases}$$

$$\sigma_i = \begin{cases} t(e_i) & \text{time of the first message in } OGP_i \\ OVT_i & \text{if the output queue is empty} \end{cases}$$

To preserve the conservative constraint for all input windows $IW_i$, all $OGP_j$ ($j \in Pred_i$) should send their earliest time $\sigma_j$ to their immediate successors $IGP_i$. Then the upper-bound of window $IW_i$ is computed as: $U_i = \min \{ \sigma_j \mid j \in Pred_i \}$ If $U_i - \tau_i < 0$ then $IW_i = \emptyset$ (i.e. the window is empty) else $IGP_i$ can send all event messages in $IW_i$ to LPs of the cluster and sets $IVT'_i = U_i$.

In the algorithm of Figure 4, all statements starting with $\forall_{i \in C}$ and those separated by || (lines 5, 8, and 13) can be executed in parallel whereas a few sequences are sequentialized by barriers (lines 9 and 15). In the distribution phase both the input and output gateway processes of a cluster $i$ are assigned to the same processor to avoid more barriers. The $IVT_i$ clock is advanced to $U_i$ (line 20) regardless of the condition whether $IW_i$ is empty or contains messages. When the new value of $IVT_i$ is known, then it becomes possible to compute $CVT_i$ and the boundary of the output window $OW_i$. All $IVT_i$ ($i \in C$) will progress if their predecessors $OGP_j$ ($j \in Pred_i$) advance their clock. The latter is also true since the virtual time of each output gateway progresses with its cluster virtual time, $CVT_j$.

```
1. Initialization stage: initialize event lists and variables
2. repeat
3. 1)   Nomination Phase:
4.      ∀i∈C IGPi : IWi = [τi, ∞];
5.   || ∀i∈C OGPi : OWi = [σi, ∞];
6. 2)   Adjustment Phase:
7. L:   ∀i∈C IGPi : send τi + δi to OGPi
8.   || ∀i∈C OGPi : send σi to IGPj  (j ∈ Suci)
9.      Barrier
10.        ∀i∈C IGPi :
11.           Ui = min { Ui, min { σj | j ∈ Predi } } ;
12.           If Ui < τi then begin IWi = Ø; τi = Ui; end
13.     || ∀i∈C OGPi :
14.           if τi + δi < σi then σi = τi + δi
15.     Barrier
16.        if any changes to any σi or τi then goto L
17. 3)  Distribution Phase:
18.        ∀i∈C IGPi :
19.           If IWi ≠ Ø then deliver messages of IWi to LPs;
20.           IVTi = Ui;  /* update IVTi */
21.           CVTi = min { Ui, {LVTi,j | j = 1, 2, ..., ni } }
22.        ∀i∈C OGPi :
23.           deliver messages (if any) of OWi = [OVTi, CVTi];
24.           OVTi = CVTi;
25. Until (End_of_Simulation)
```

**Figure 4:** The HCTW algorithm which synchronizes the parallel activities of Local Time Warp at the cluster level. Notice that in the adjustment phase some flags can be used to prevent unnecessary message sending and recomputation. This detail, however, is intentionally left to the implementation level.

The Local Time Warp model is sufficiently general to be synchronized by other protocols at cluster level. Nevertheless, it is our belief that HCTW can efficiently synchronize the activity of the Local Time Warp approach for the following reasons: (1) The output gateway exploits a time window over messages to be sent out, and hence it is more natural to use a window-based algorithm. (2) Several studies (Ayani and Rajaei 1992; Dickens and Reynolds 1990; Lubachevsky 1989; Nicol 1991) indicate that synchronous window-based conservative algorithms present an important class of parallel simulation schemes which in some cases outperform other conservative methods. (3) Synchronous window-based algorithms are powerful enough to incorporate extra features such as a general termination algorithm Sanjeevan and Abrams 1991). (4) The HCTW algorithm can accommodate both SIMD, MIMD, and a combination of SIMD-MIMD architectures.

The HCTW algorithm of Figure 4 can be improved. Instead of having processors blocked at barrier at line 9 until all of them arrive, the barrier can be replaced by a **wait-until** construct. In this case an $IGP_i$ process waits until it receives $\sigma_j$ from all its predecessor $OGP_j$. Likewise, an $OGP_i$ will wait until it receives $\tau_i + \delta_i$ from its $IGP_i$. Having two gateway processes per cluster,

although increases the degree of parallelism in the nomination and adjustment phases, increases the overhead of sending messages from IGPs to OGPs as well as the overhead of context switching between GPs. If the two gateway processes are merged into one GP while keeping the Local Time Warp model intact (i.e. having the two $IVT_i$ and $OVT_i$ clocks and the two $IW_i$ and $OW_i$ windows), then these overheads may decrease trading off the degree of parallelism. Figure 5 illustrates the above two modifications. The algorithms of Figures 4 and 5 seem to favour shared memory multiprocessors mainly due to the usage of barrier. However, there are techniques that make the algorithms suitable also for computers distributed over a network, e.g. distributed shared memory, distributed barrier (Davis and Ramachandran 1992), or a simple co-ordinator process to integrate the functionality needed for a distributed barrier.

1. **Initialization stage**: initialize event lists and variables
2. **repeat**
3. **1) Nomination Phase:**
4. $\forall_{i \in C}\, GP_i$ : $IW_i = [\tau_i, \infty]$; $OW_i = [\sigma_i, \infty]$;
5. **2) Adjustment Phase:**
6. L: $\forall_{i \in C}\, GP_i$ :
7. **if** $\tau_i + \delta_i < \sigma_i$ **then** $\sigma_i = \tau_i + \delta_i$ ;
8. send $\sigma_i$ to $\forall\, GP_j$ $(j \in Suc_i)$ ;
9. **wait until** received $\{\sigma_j \mid j \in Pred_i\}$;
10. $U_i = \min\{ U_i, \min\{ \sigma_j \mid j \in Pred_i \}\}$ ;
11. **If** $U_i < \tau_i$ **then begin** $IW_i = \emptyset$; $\tau_i = U_i$; **end**
12. **Barrier**
13. **if** any changes to any $\tau_i$ **then goto** L
14. **3) Distribution Phase:**
15. $\forall_{i \in C}\, GP_i$ :
16. **If** $IW_i \neq \emptyset$; **then** deliver emessages of $IW_i$ to LPs;
17. $IVT_i = U_i$;
18. $CVT_i = \min\{ U_i, \{LVT_{i,j} \mid j = 1, 2, \ldots, n_i\}\}$;
19. deliver messages (if any) of $OW_i = [OVT_i, CVT_i]$;
20. $OVT_i = CVT_i$;
21. **Until (End_of_Simulation)**

**Figure 5:** The improved HCTW algorithm. The two gateway processes are merged into one, $GP_i$, while the LTW model is preserved.

### 4.3 Special Cases

In the Global Time Warp protocol the receiving time of each message should be greater than its sending virtual time (Jefferson 1985, Rule 1). Some implementations of Time Warp relax this rule. In LTW, this constraint is also relaxed for local messages in a cluster. However, the LPs should not send only such messages and must also have messages that consume non-zero positive service time. For external messages, as mentioned previously, the receive virtual time should be greater than the send virtual time. More formally:

**Assumption 1:** The service time of an $LP_{i,j}$ for an internal message cannot be zero for the entire simulation period and for an external message should not be zero at any time.

**Assumption 2:** There is no LP in the simulation problem such that its local virtual time never advances.

Assumptions 1 and 2 cannot prevent the possibility of zero cycle. As an example consider two LPs of a cluster sending messages to each other consuming zero service time. The same two LPs may occasionally send messages to other LPs while consuming non-zero service time for such messages. Consequently relation (5) is satisfied but the zero cycle remains. Therefore, the following condition, which is referred to as *predictability property* (Misra 1986), should hold:

**Assumption 3:** Let $\Gamma_i$ be the set of $n$ logical processes in cluster $i$ building a cycle of $<LP^1, LP^2, \ldots, LP^n, LP^1>$ such that $LP^k$ sends messages to its successor $LP^{(k+1) \bmod n}$ in the cycle (the successor of $LP^n$ is $LP^1$). In such a cycle, there is at least one process $LP^j$ such that whenever it sends a message $m$ to its successor in the cycle, the timestamp $t$ of that message is greater than the time at which the message is sent:

$$\exists\, LP^j \in \Gamma_i \mid m(LP^j, LP^{(j+1) \bmod n}, LVT^j, t) \text{ and } t > LVT^j \quad (5)$$
$$j = 1, 2, \ldots, n$$

### 4.4 Relaxing Non-zero Time Distance Constraint

Until now it has been assumed that there was a positive non-zero time distance $\delta_i$ from cluster $i$ to its immediate successors. Although this constraint improves performance, for some application it may not be needed. In the absence of such a time distance, clusters of the simulation system in the worst case progress sequentially. Nevertheless, this does not necessarily imply that the LPs of other clusters cannot continue simulating their local events. One problem which may arise, however, is that the probability of rollback inside clusters increases. Since the progress rate of $CVT_i$ may become less than the case with non-zero time distance, the length of state queue may increase. A global zero cycle can also arise in the simulation system. To prevent such a cycle, the predictability property of assumption 3 is generalized as follows:

**Assumption 4:** Let $\Psi$ be the set of $n$ logical processes in the simulation system building a cycle of $<LP^1, LP^2, \ldots, LP^n, LP^1>$ over clusters boundaries such that $LP^k$ sends messages to its successor $LP^{(k+1) \bmod n}$ in the cycle. In such a cycle, there is at least one process $LP^j$ such that whenever it sends a message $m$ to its successor in the cycle, the timestamp $t$ of that message is greater than the time at which the message is sent:

$$\exists\, LP^j \in \Psi \mid m(LP^j, LP^{(j+1) \bmod n}, LVT^j, t) \text{ and } t > LVT^j \quad (6)$$
$$j = 1, 2, \ldots, n$$

Assumption 4 should hold for the entire simulation system and over cluster boundaries. However, for the sake of generality the following rule is derived from assumption 4 to comprise cycles of clusters (Figure 6):

**Assumption 5:** Let $\Sigma$ be the set of $n$ clusters $C^i$ in the simulation system constructing a cycle of $<C^1, C^2, \ldots, C^n, C^1>$ such that $C^i$ sends messages to its successor $C^{(i+1) \bmod n}$ in the cycle (successor of $C^n$ is $C^1$). In such a cycle, there is at least one cluster $C^k$ such that whenever a message $m$ is passed through its gateways, the timestamp of that message is increased by at least one of the LPs of that cluster:

$$\exists\, C^k \in \Sigma \mid OVT^k(m) > IVT^k(m) \quad (7)$$
$$k = 1, 2, \ldots, n$$

where $IVT^k(m)$ and $OVT^k(m)$ are respectively the virtual time for a message $m$ at the input and output gateways of cluster $k$.

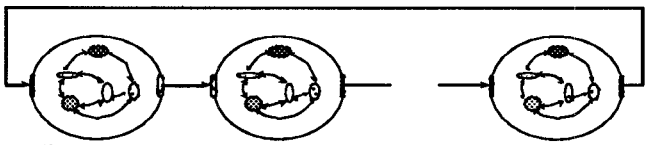

**Figure 6:** Interconnection of clusters may construct a loop

## 5. DYNAMIC CONFIGURATION

One important property of the Global Time Warp is the support for dynamic configuration which is highly desirable in simulation of complex applications. The same is preferred for LTW. LPs inside a cluster can dynamically be created and terminated as in GTW. The gateway processes should handle message arrivals for terminated LPs. Most likely this level of flexibility will suffice for many applications which may need dynamic changes only inside their subsystems. However, for some applications such as communication networks or computer systems it is advantageous to provide dynamic changes even at cluster level. Such applications can utilize this property to model, e.g. partial network crashes or subsystem break-down. The primary difficulty is how to incorporate the changes at a level which is synchronized by a conservative protocol using links. This section presents a method for HCTW that facilitates creation and termination of clusters during simulation. A similar technique, however, may be used for changing the structure of network of LPs when using, e.g., the Conservative Time Windows algorithm.

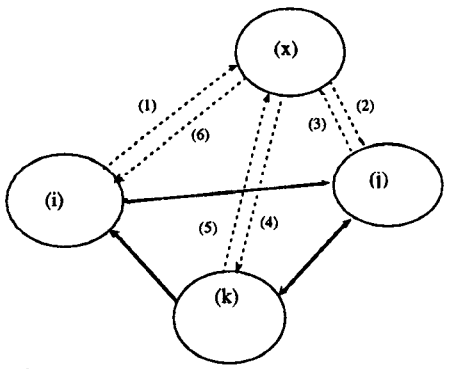


**Figure 7:** Dynamic creation (termination) of a cluster $x$ in Local Time Warp.

### 5.1 Dynamic Creation of Clusters

A cluster can be created by a request sent either from another cluster, or from the outside world. The dominant problem is how to link the newly created cluster to the existing ones in ICN, while the conservative constraint at the cluster level is maintained. We can describe the mechanism through an example. Consider Figure 7 where a cluster $x$ is newly created by a request from cluster $i$ and is to be coupled to ICN by a set of requested links shown by dashed lines. The figure illustrates three types of links: *forward*, *reverse*, and *intermediary*. In Figure 7, link 1 and 2 are intermediary, whereas 3 and 5 are reverse, 4 and 6 are forward.

(1) Forward link: is established from $x$ to an arbitrary cluster $k$ by sending a *request-to-link* to $k$. The request message has a timestamp $t(req)$, a higher priority over event-messages, and may not follow the time order constraint. A request-to-link is granted immediately if $t(req) \geq IVT_k$. Otherwise the request is delayed with $\Delta = IVT_k - t(req)$ to enforce relation 4 and maintain the conservative property at the cluster level. In both cases the link can be established as soon as the request arrives, but the response message will have a time of $t$ for the first case and $t + \Delta$ for the second case notifying the source of request when the new link can be used. In case of a delay, the virtual time of the output gateway, $OVT_x$, either should be adjusted to accommodate the change or special measures should be taken to enforce the delay.

(2) Reverse link: links an arbitrary cluster $k$ to the newly created $x$. A *request-for-link* is sent from cluster $x$ to cluster $k$. At arrival of such a request if $t(req) \leq OVT_k$ then the link is immediately established, otherwise a delay of $\Delta = t(req) - OVT_k$ will be added to the response. As in forward linking, a reverse link can be established as soon as the request arrives. The response time denotes whether the request-for-link was accepted immediately or a delay was imposed.

(3) Intermediary links: connect the newly created cluster $x$, from the requesting source $i$ to its immediate successor $j$ through $x$, i.e. links 1 and 2. This type is a special case of both forward and reverse links where the arbitrary clusters are replaced by pairs of directly linked ones. The new links can be established immediately after creation of the requested cluster at simulation time $t$. All three clocks of the new cluster are initialized to $t$, i.e. $IVT_x = CVT_x = OVT_x = t$, to enforce relation (4) for clusters $x$, $i$, and $j$.

### 5.2 Dynamic Termination of Clusters

Clusters can be terminated by two methods: (1) a termination condition is detected by the cluster after CVT is computed; (2) a termination request is received. In the first case the gateway processes of the cluster inform their respective predecessor and successor clusters of the condition and then may close all related links. The output links are closed when all messages with timestamp less than CVT are delivered. In the second case, a request for termination may arrive at either input or output gateway of a cluster corresponding to an external or internal request. If the time of the request is $t$, the output gateway will discard all messages with timestamps greater than $t$. In addition, the gateway informs its immediate successors about the termination by sending a *link-closed* message at time $t$. Likewise is done for the input gateway and all predecessors are informed. The predecessor gateways discard all messages that should be sent to cluster $x$ and have timestamp greater than $t$. Finally, the cluster will terminate when its $CVT_x$ reaches $t$. In this scheme the legitimacy of the termination request is left to the application program.

## 6. CONCLUSIONS

We proposed the *Local Time Warp* approach to PDES which combines the optimistic method with the conservative one at two different levels. In contrast to the Global Time Warp and previously proposed combined schemes, the LTW model hierarchically partitions the simulation system into a number of subsystems, each comprising numerous objects. The components of each subsystem are simulated by the Time Warp

protocol whereas the activities between the subsystems are synchronized by a window-based conservative protocol. Synchronization of the LTW model is achieved by a novel scheme called HCTW which hierarchically combines the Conservative Time Window algorithm with Time Warp. The LTW model is general enough to be synchronized by other methods. However, we believe that the HCTW scheme can efficiently handle the complexity of the combined approach.

There are several advantages in using the Local Time Warp approach for parallel simulation. The model prevents cascading rollbacks in the global system and does not allow a rollback occurring in one cluster to penetrate into others. The state saving overhead is reduced in many cases by bounding cascading rollbacks. Moreover, the computation of the Cluster Virtual Time, CVT, becomes less time consuming than its GVT counterpart. As a result, CVT can be more frequently computed for reclaiming memory of fossils hence reducing the amount of memory needed for state saving. Consequently, the Local Time Warp is unlikely to become unstable, while at the same time the dynamic behaviour of the Global Time Warp to a great extent is preserved.

The Local Time Warp approach to parallel simulation together with its synchronization scheme HCTW may be used for simulation of large and complex applications on computer systems with massive parallelism and hence supports scalability of the parallel simulation. This method appears to suit both SIMD, MIMD or a combined SIMD-MIMD architecture as well as a network of parallel computers. The LTW model is applicable to both shared and distributed memory multiprocessor systems. A prototype of the LTW model is currently operational and we are investigating its performance. In addition, further investigations are needed for issues such as the scalability of the model as well as incorporating higher levels of clusters and dedicating each one to a parallel machine distributed on a network of computers.

## ACKNOWLEDGEMENTS

This work was partly supported by the Swedish National Board for Industrial and Technical Development (NUTEK) under contract # 90-01773.

## REFERENCES

Arvind D.K., and Smart C.R., 1992. "Hierarchical Parallel Discrete Event Simulation in Composite ELSA", in *Proc. of 6th Workshop on Parallel and Distributes Simulation*, 147-156.

Ayani R. and Rajaei H. 1992. "Parallel Simulation Using Conservative Time Window", in *Proc. of the Winter Simulation Conference*, (Dec), 709-717

Baezner D., Lomow G., Unger W. 1990. "Sim++: The Transition to Distributed Simulation", *Proc. of SCS90, Conf. on Dist-Sim, San Diego*, (Jan), 211-218.

Bagrodia R. and Liao W. 1990. "Maisie: A Language and Optimizing Environment for Distributed Simulation", *In Proc. of SCS90, Conf. on Dist-Sim, San Diego*, (Jan), 205 - 210.

Ball D. and Hoyt S., 1990. "The Adaptive Time Warp Concurrency Control Algorithm", in *Proc. of SCS90 Multiconference on Distributed Simulation*, 174-177.

Bellenot, Steven, 1990. "Global Virtual Time Algorithms", in *Proc. of SCS Multiconference on Distributed Simulation*, 122-127.

Davis M.H., and U. Ramachandran, 1992. "A Distributed Hardware Barrier in an Optical Bus-Based Distributed Shared Memory Multiprocessor", in Proc. of International Conference on Parallel Processing, I-228-231.

Dickens, P.M., and Reynolds P.F. 1990. "SRADS with Local Rollback", in *Proc. of the SCS Multiconference on Distributed Simulation 22, 1*, 161-164.

Dickens P., and Reynolds P., 1991. "A Performance Model for Parallel Simulation", in *Proc. of the 1991 Winter Simulation Conference*, 618-626.

Fujimoto, Richard. 1988. "Lookahead in Parallel Discrete Event Simulation", in *Proc. of International Conference on Parallel Processing*,.

Fujimoto, Richard. 1990. "Performance of Time Warp under Synthetic Workloads", in *Proc. of the SCS Multiconference on Distributed Simulation, San Diego*, 23-28.

Fujimoto, Richard. 1990. "Parallel Discrete-Event Simulation", *Comm. of ACM vol. 33, No. 10*, 30-53.

Fujimoto, Richard. 1990. "Optimistic Approaches to Parallel Discrete Event Simulation", *Trans. of the Society for Computer Simulation, Vol. 7, No. 2*, 153-191.

Jefferson D.R. 1985. "Virtual Time", *ACM Trans. on Programming Language and System, 7, 3* 404-425.

Jefferson, David, 1990. "Virtual Time II: Storage Management in Distribute Simulation", in *Proc. of the 9th Annual ACM Symp on Principles of Distributed Computing*, 75-90.

Lin Y-B. and Lazowska E., 1990. "Determining the Global Virtual Time in a Distributed Simulation", in *Proc. of International Conference on Parallel Processing* , III:201-209.

Lin, Yi-Bing, 1992. "Memory Management Algorithm for Optimistic Parallel Simulation", *in Proc. of the 6th Workshop on Parallel and Distributed Simulation*, 43-52

Lubachevsky, Boris. 1989. "Efficient Distributed Event-Driven Simulation of Multiple Loop Networks", *Communication of the ACM*, 32: 111-123.

Lubachevsky, B.D., Shwartz, A., and Weiss A. 1989. "Rollback Sometimes Works ... if Filtered", in *Proc. of 1989 Winter Simulation Conference*, 630-639.

Lubachevsky, Boris 1990. "Simulating Colliding Rigid Disks in Parallel Using Bounded Lag Without Time Warp", in *Proc. of SCS Multiconference on Distribute Simulation*, 194-202.

Lubachevsky B., Weiss A, Shwartz A, 1991. "An Analysis of Rollback-Based Simulation", *ACM Trans. on Modeling and Computer Simulation, Vol. 1, No. 2*, 154-193.

Madisetti, V., Walrand J., and Messerschmit D. 1988. "Wolf: A Rollback Algorithm for Optimistic Distributed Simulation Systems", in *Proc. of Winter Simulation Conference*, 296-305.

Madisetti V., Hardaker D., and Fujimoto R. 1992. "The MIMDIX Operating System for Parallel Simulation", in *Proc. of the 6th Workshop on Parallel and Distributed Simulation*, 65-74.

McAffer, Jeff. 1990. "A Unified Distributed Simulation System", in *Proc. of the 1990 Winter Simulation Conference*, 415-422.

Mehl, Horst. 1991. "Speed-Up of Conservative Distributed Discrete Event Simulation Method by Speculative Computing ", in *Proc. of SCS Multi-Conference on Advances in Parallel and Distributed Simulation*, 163-166.

Misra, Jayadev. 1986. "Distributed Discrete-Event Simulation", *ACM Computing Surveys, Vol. 18, No. 1*, 39-65.

Nicol, D.M.1991. "Performance Bounds on Parallel Self-Initiating Discrete-Event Simulation", *ACM Trans. on Modeling and Computer Simulation, Vol. 1. No. 1*, 24-50.

Parakash A. and Subramanian R., 1992. "An Efficient Optimistic Distributed Simulation Scheme based on Conditional Knowledge", in *Proc. of 6th Workshop on Parallel and Distributes Simulation*, 85-94.

Reiher P., and Jefferson D., 1989. "Limitation of Optimism in the Time Warp Operating System", in *Proc. of 1989 Winter Simulation conference*, 765-769.

Reynolds, Paul. 1988. "A Spectrum of Options for Parallel Simulation", in *Proc. of the 1988 Winter Simulation conference*, 325-332.

Reynolds P., Weight C., and Fidler R. 1989. "Comparative Analyses of Parallel Simulation Protocols", in *Proc. of the 1989 Winter Simulation Conference*, 671-678.

Sanjeevan V., and Abrams M. 1991. "The Cost of Terminating Synchronous Parallel Discrete-Event Simulation", in *Proc. of the 1991 Winter Simulation Conference*, 642-651.

Sokol L.M., Briscoe D.P., and Wieland A.P. 1988. "MTW: A Strategy for Scheduling Discrete Simulation events for Concurrent Execution", in *Proc. of SCS Conf. on Distributed Simulation*, 34-42.

Steinman, Jeff. 1992. "SPEEDES: A Unified Approach to Parallel Simulation", in *Proc. of the 6th Workshop on Parallel and Distributed Simulation*, 75-83.

Turner S.J, and Xu M.Q., 1992. "Performance Evaluation of the Bounded Time Warp Algorithm", in *Proc. of 6th Workshop on Parallel and Distributes Simulation*, 117-126.