Adaptive algorithms vs. Time Warp: An analytical comparison

Sudhir Srinivasan
Paul F. Reynolds, Jr.

Computer Science Report No. CS-95-20
April 6, 1995

Adaptive algorithms vs. Time Warp: An analytical comparison

Sudhir Srinivasan Paul F. Reynolds, Jr.
Department of Computer Science
Olsson Hall, University of Virginia
Charlottesville, VA 22903.
Phone: (804) 982-2200 Fax: (804) 982-2214
Email: `{ss7a|pfr}@uvacs.cs.Virginia.EDU`

Abstract

Adaptive synchronization algorithms have been proposed to improve upon purely conservative and purely optimisitic algorithms. Experimental studies have have indeed provided encouraging results. In the spirit of previous analyses, we present the first known analytical comparison of adaptively optimisitic algorithms with the Time Warp protocol. We define a class of adaptive protocols, the asynchronous adaptive waiting protocols (AAWP's) and identify several practical protocols that belong to this class. We show that Time Warp can outperform an AAWP arbitrarily. We describe NPSI adaptive protocols, a sub-class of AAWP's, and specify a member of this sub-class, the Elastic Time Algorithm. We show that this algorithm can outperform Time Warp arbitrarily.

Keywords: Modeling methodology, parallel simulation, adaptive protocols, limited optimism, Time Warp, Elastic Time Algorithm, reduction networks.

Submitted to: *1995 Winter Simulation Conference*

1 Introduction

Historically, research in synchronization algorithms (protocols) for parallel discrete event simulation (PDES) has followed two tracks: conservative and optimistic [Fuji90]. A significant result of this research is that neither approach seems to be universally efficient. As defined in [Reyn88], *adaptive* protocols are those that modify their behavior dynamically in response to changes in the state of the simulation. Several adaptive protocols have been proposed and implemented recently, with encouraging results. However, there are no analytical studies comparing the performance of adaptive protocols with that of traditional protocols. While experiments have shown that adaptively optimistic protocols improve on the performance of the purely optimistic Time Warp protocol [Jeff85] in general, it is interesting to question whether they can perform worse than Time Warp. We present a comparison of a general class of adaptive protocols called the asynchronous adaptive waiting protocols (AAWP's) with Time Warp in the vein of the worst-case comparison between the Chandy-Misra protocol [ChMi79] and Time Warp in [LiMi90]. We show that it is possible for Time Warp and AAWP's to outperform each other arbitrarily. Thus, while intuition suggests that adaptive protocols should enhance performance in general, our analysis indicates that they must be designed carefully since incorrect adaptive decisions can lead to arbitrarily worse performance than Time Warp.

We assume familiarity with the common approach to PDES [Fuji90], namely the partitioning of a discrete event simulation into components called logical processes (LP's). Each LP is itself a sequential discrete event simulator. The LP's must execute events, whether generated internally or scheduled by other LP's, without violating causality constraints (effectively). Typically, this is the responsibility of a *protocol*.

A *conservative* protocol is one in which an LP executes an event only after determining that it is safe to do so (i.e. no other event with a smaller timestamp will be scheduled later). Thus, it is possible for an LP to remain blocked for some period of time while there is insufficient information to proceed with the next scheduled event. An *optimistic* algorithm takes the opposite approach in that LP's execute events without the guarantee of safety and "repair" their execution if and when an error is detected, typically using a checkpoint and rollback approach. In [Reyn88], *aggressiveness* is defined to be the property by which an LP processes events conditionally (i.e. without a guarantee of safety) and *risk* is defined as passing messages that are based on aggressive or inaccurate processing. Conservative protocols are non-aggressive and without risk while optimistic protocols are completely aggressive and with risk. Therefore, these two classes of protocols represent the extremities of a spectrum of possibilities.

Both conservative and optimistic strategies have their pros and cons. The main advantage of conservative protocols is that they do not have any checkpointing and rollback overheads. However, processes may be blocked due to insufficient information when in fact it would be safe for them to proceed. This artificial blocking [Reyn82] introduces lost-opportunity cost [SrRe94]. On the other hand, optimistic protocols do not incur lost-opportunity costs since processes never block. The main hazard of optimistic protocols is that the checkpointing and rollback costs may degrade performance severely. The gap between these two extremes may be bridged either by adding optimism to conservative protocols or by limiting the optimism of purely optimistic protocols. The inherently dynamic nature of simulations [NiRe90] suggests that regardless of the approach to combine the conservative and optimistic strategies, the hybrid scheme must also be dynamic. *Adaptive protocols* are those that change the bindings of one or more of their design variables dynamically [Reyn88]. Adaptive protocols that modify their aggressiveness and risk (collectively called *optimism*) dynamically seem the most likely to perform well consistently. Henceforth, we will refer to such protocols simply as adaptive protocols.

Several protocols have been proposed that either limit optimism in Time Warp or add optimism to conservative protocols [SrRe94]. Most of these have been shown to improve the performance of the protocols from which which they are derived. Since all of these results are experimental, it is interesting and important to question whether adaptive protocols can perform worse than traditional protocols. Our analysis shows they can.

In the next section we define the AAWP class of adaptive protocols and identify several members of this class. We then show by example that Time Warp can outperform an AAWP arbitrarily. Next, we describe a framework for a family of adaptive protocols called NPSI adaptive protocols (which are also AAWP's). Finally, we describe a specific NPSI adaptive protocol, ETA, based on this framework and show that ETA can outperform Time Warp by a factor proportional to the amount of logical time simulated.

2 Assumptions

The reader is referred to [Fuji90] for details of Time Warp and related concepts. We assume the following for all protocols in this paper:

- Each LP is located on its own processor.
- The protocols employ aggressive cancellation and aggressive rollback.

The following assumptions define *asynchronous adaptive waiting protocols* (AAWP's), the class of adaptive protocols to which our analysis applies. These protocols are based on Time Warp and control optimism by introducing delays between event executions:

i) The simulation loop of an LP is as follows:

```
While not done
   Process event
   Wait for time δ >= 0
   Rollback if necessary
   Process received messages
   Save state
   Collect fossils
endwhile
```

Note, we do not make any assumptions about the particulars of the AAWP, such as criteria for deciding if an LP should wait on a given iteration or how long it should wait. We only assume that such waiting occurs between event executions, if at all. An LP aborts its waiting if it receives a message that will cause it to roll back. A separate delay scheme may be used to control risk.

ii) The waiting at each LP is asynchronous with respect to the waiting at other LP's.

iii) The actions of an LP that are relevant to the analysis are: event execution, rollback and adaptive waiting. We ignore overheads such as state saving, receiving messages, global virtual time (GVT) computation and fossil collection for simplicity; the analysis can be extended to include these as well.

iv) The AAWP does not increase the capability of the LP's to guess computation. An increase in the guessing power of an LP could compensate for any deleterious effects of the adaptive control. Since an LP's guessing capability depends entirely on the application being simulated, it is reasonable to assume the AAWP cannot increase this capability.

v) Since adaptive waiting is expected to reduce rollback costs, the depth (and therefore cost) of each rollback is assumed to be bounded by a constant for the AAWP. If the rollback costs are not bounded for the AAWP, our claim that an AAWP can perform arbitrarily worse than Time Warp can be shown trivially.

vi) Since the delay time δ can be controlled directly by the AAWP, it is assumed to be bounded by a constant. Once again, if this were not true, our claim that an AAWP can perform arbitrarily worse than Time Warp can be shown trivially.

Several protocols that have been proposed and implemented belong to the class of AAWP's. In the penalty based throttling scheme of [ReJe89], an LP that has been rolling back excessively is made to block for some period of time. This decision is made independently of other LP's. Similarly, in Adaptive Time Warp [BaHo90], an LP may decide to block after executing an event based on local history and statistical estimation. In [Madi93], LP's estimate each others' logical clock values and block if their clock value differs largely from that of another LP. In [HaTr94], a real-time blocking window is computed each time an LP executes an event and the LP blocks for an amount of time equal to this window (which may be zero). Similarly, in [FeTr94], an LP blocks probabilistically for some amount of time after each event execution. The new class of adaptive protocols we have described in [SrRe94], which we call NPSI adaptive protocols, also satisfy the AAWP assumptions. Windowing algorithms in which the windows are computed individually for different LP's (such as Unified Distributed Simulation system [McAf90] and Breathing Time Warp [Stei93]) are AAWP's as well since the LP's wait when they reach the ceilings of their independent windows. Note, global windowing algorithms do not fit the AAWP model since the global window forces all LP's to synchronize before any of them can proceed. Protocols in which optimism is limited based on spatial clustering [Gima89, RaAT93] also satisfy our assumptions, since waiting may be required at cluster boundaries where events are exchanged conservatively and the clusters operate asynchronously.

3 Time Warp outperforms AAWP's

We show by example that an AAWP can take arbitrarily longer than Time Warp to complete a simulation. The intuition behind the example is this: on the one hand, it is possible for Time Warp simulations to execute very efficiently, with few rollbacks; on the other hand, it is also possible for a Time Warp simulation to generate many false events and consequent rollbacks which can degrade its performance severely [LuWS91]. Errors in adaptive decisions regarding when and how long to wait can cause a Time Warp execution to move from the former category to the latter. In the example

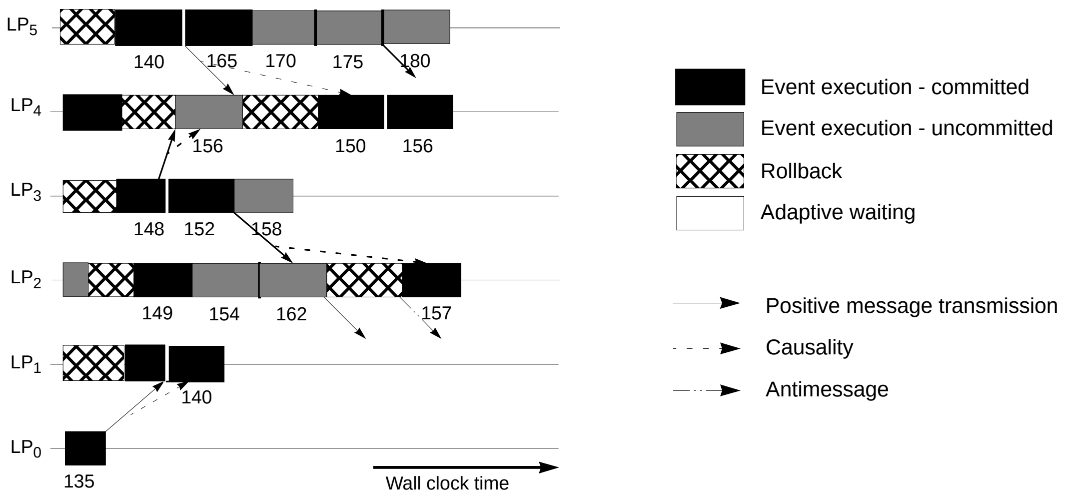


Figure 1 - Time Warp execution

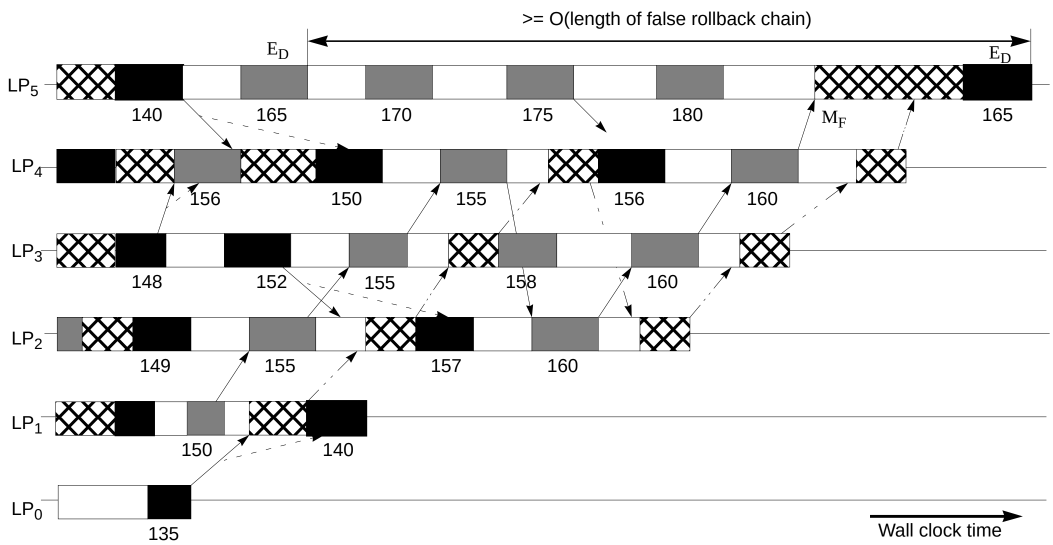


Figure 2 - False rollback chain with cycle due to incorrect waiting

here, we show a situation where the AAWP induces a false rollback chain that delays the committing of an event (relative to the Time Warp execution) by at least an amount of time proportional to the length of the rollback chain. By arguing that this rollback chain can be arbitrarily long, we show that the committing of an event can be delayed arbitrarily.

Consider the Time Warp execution shown in Figure 1. The x-axis denotes advance of wall-clock time while the y-axis denotes the different LP's. The numbers below the events are their respective timestamps. A dashed arrow indicates the causal dependence of an event on a message (i.e. the event at the head of the arrow was scheduled by the arrival of the message at the tail of the arrow). The two important events to note in this execution are: (i) the event with timestamp 140 at LP_1, which is causally dependent on a message from LP_0 that arrives just in time to be executed by LP_1, and (ii) the event with timestamp 165 at LP_5, which is the one whose committing execution will be delayed due to an erroneous waiting decision.

Figure 2 shows an execution of the same simulation as in Figure 1 using an AAWP. For simplicity, we assume the initial conditions for the two executions are the same except for one difference: both LP_0 and LP_1 wait for some time at the beginning of the portion of the execution depicted. Due to what will turn out to be an error in the decision process, LP_0 delays longer than LP_1. As a result, LP_1 is ready to execute an event before the message from LP_0 arrives and causes the event with timestamp 140 to be scheduled. LP_1 executes its next scheduled event (with timestamp 150) and sends a message to LP_2. Since we assume aggressive rollback, this event is a *false* one as it is executed out of order. When the message from LP_0 arrives later, this false event is rolled back and an antimessage is sent to LP_2. However, the message sent to LP_2 by the false event has already intiated a chain of false events. The antimessage starts a chain of rollbacks and antimessages that follows close behind the chain of false events. Regarding these two chains, we observe the following:

i) It is possible that the rollback chain catches up with the false event chain immediately and thus terminates both chains. However, the case that is relevant to this discussion is the one shown in Figure 2 where the rollback chain does not catch up with the event chain until the latter reaches LP_5 and initiates an unnecessary rollback. This scenario is feasible [LuWS91].

ii) The chains by themselves may be harmless - the problem arises from the fact that the final false message that arrives at LP_5 (marked M_F in Figure 2) has timestamp 160, which is smaller than 165, the timestamp of the second event at LP_5 (marked E_D in Figure 2). Thus, M_F rolls back the first execution of E_D, *even though that execution was correct and could have been committed.* E_D is re-executed after the false rollback completes. Therefore, the committing execution E_D is delayed by at least an amount of time proportional to the length of the false rollback chain.

iii) Each of the two chains has a cycle in it, involving LP_2, LP_3 and LP_4. While we have shown only one iteration of this cycle, it is possible to have an arbitrary number of iterations, i.e. the chains could revisit these LP's an arbitrary number of times before reaching LP_5.

iv) For the false event chain to delay the committing execution of E_D, the timestamp of M_F must be smaller than that of E_D. Since we assume a correct implementation of the underlying Time Warp protocol and the application, it follows that logical time must advance eventually as we traverse the false event chain. Therefore, an arbitrarily long chain would require that the timestamp of E_D also be arbitrarily large (we disregard pathological situations such as when the false chain makes diminishing advances in logical time such that the timestamp of M_F is bounded; such situations reflect a flaw in the implementation - the physical system being simulated cannot make diminishing progress in real time!). The larger the timestamp of E_D, the less likely that its first execution is on the critical path of the simulation, since other LP's are farther behind in logical time. Thus, while it is possible for the false chains to be arbitrarily long, the probability that these chains are damaging to the simulation decreases with the length of the chain. However, the probability is not zero - we can imagine a simulation where E_D marks the transition to a new phase of simulated time, i.e., the entire simulation makes a "jump" in logical time to the next phase of activity. If so, the delaying of E_D could cause the entire phase transition to be delayed.

v) It is possible to have very long false chains which do not span much logical time, as shown in Figure 2. Here, logical time does not increase in an iteration of the cycle in the false chain; it increases only across iterations. Thus, the timestamp of M_F is small but the chains are long.

vi) Finally, even if the lengths of the chains are bounded (by some means), it is possible to have an arbitrary number of instances of such chains in the course of a single simulation run.

In summary, we have shown by example that the committing execution of an event may be delayed (relative to the Time Warp execution) by an arbitrary amount of time due to false events and rollbacks created by errors in the waiting decision.

3.1 Lazy cancellation

Since the events in the rollback chain are false events (i.e. they should not have been generated), they will have to be rolled back even under lazy cancellation. Thus, the example holds under lazy cancellation as well. In fact, the problem is aggravated to a certain extent with lazy cancellation because the initial antimessage generated by the rollback of the event with timestamp 150 at LP_1 will be delayed until LP_1 crosses time 150 after executing the event with timestamp 140. Thus, the physical time lag between the two chains may be longer, allowing the false event chain to propagate farther.

3.2 Lazy rollback

The commitment of the first execution of E_D in the Time Warp execution implies that the event with timestamp 150 at LP_1 *must* be a false one even with lazy rollback. Thus, the two chains will be generated in this case also. However, E_D may not be rolled back when

M_F arrives. This can happen due to one of the following: (i) M_F has no impact on the state in which E_D was executed, or (ii) the rollback chain arrives at LP_5 while LP_5 executes the event scheduled by M_F so that when the execution is complete (or is preempted, assuming event preemption), it is cancelled immediately and the operating state for E_D is not affected. However, even if the final execution of E_D is not delayed, the advance of GVT* beyond 165 will be delayed by at least the amount of time it takes for the chains to disappear. Since GVT is the commitment horizon for events [Jeff85], the committing of E_D can be delayed arbitrarily even with lazy rollback.

3.3 Pragmatic issues

We have identified a general class of adaptive protocols, the AAWP's, and shown that it is possible for Time Warp to outperform them by an arbitrary amount. Since several practical protocols belong to the AAWP class, this counter-intuitive result suggests that AAWP's must be designed with care. Correspondingly, while we have developed several very effective NPSI adaptive protocols [SrRe94], we have also discovered some that perform no better than Time Warp and some that perform worse.

It is possible to modify protocols so as to avoid the scenario described earlier. However, it is not clear whether doing so will be beneficial in the general case. Moreover, since there may be other scenarios similar to the one we have described, establishing a property of AAWP's that avoids this scenario cannot guarantee that Time Warp will not outperform AAWP's by more than a constant factor.

4 NPSI adaptive protocols

We describe a design framework for a new class of adaptive protocols. This framework will be used in section 5.3 to define a specific adaptive protocol that outperforms Time Warp by a factor proportional to the length of the simulation run. We call this new class *near-perfect state information* (NPSI) adaptive protocols because these protocols assume the availability of near-perfect information at each LP about the state of the system, at little or no cost to the simulation. The adaptive waiting decisions of LP's are based on this NPSI. In practice, such information can be disseminated using a high-speed reduction network [PaRe93] at almost no cost to the simulation. Our studies [SrRe93] have shown that such a network can disseminate critical information to the LP's at latencies

* *Global virtual time* is the minimum of all logical clocks and timestamps of any messages in transit [Jeff85].

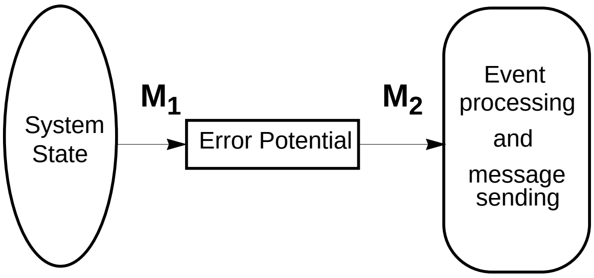


Figure 3 - Framework for NPSI adaptive protocols

that are two or three orders of magnitude smaller than typical event execution times (i.e microseconds versus milliseconds). We have designed and implemented NPSI protocols over a prototype reduction network with very encouraging results. A more detailed discussion on the rationale behind NPSI protocols, their design and performance can be found in [SrRe94].

There are two phases in the design of NPSI adaptive protocols:

- identifying the information that must be collected dynamically and on which the decision to limit optimism is to be based
- designing the mechanism that translates the collected information into control over an LP's aggressiveness and risk

The framework depicted in Figure 3 separates these phases by introducing a quantity we call *error potential* (EP_i), associated with each LP_i. EP_i is an estimate of the need for LP_i to dcrease its optimism. The mapping M_1 translates the relevant NPSI to a value of EP_i. The NPSI adaptive protocol keeps EP_i up-to-date for each LP_i as the simulation progresses, by evaluating M_1 at high frequency using state information it receives from the feedback system. M_2 dynamically reflects new values of EP_i in the event execution and communication rates. Different NPSI adaptive protocols may be constructed by designing the mappings M_1 and M_2. Note, these mappings only specify if an LP should wait and how long it should wait; the general structure of NPSI protocols conforms to the AAWP model in section 2.

5 AAWP's outperform Time Warp

We show by example that Time Warp can take arbitrarily longer than an AAWP to complete a simulation. Our approach is to describe the execution of a simulation using both Time Warp and a specific AAWP. We show that Time Warp takes an amount of time that is quadratic in the amount of logical time simulated while the AAWP takes linear time. Since the difference in completion times is not bounded by a

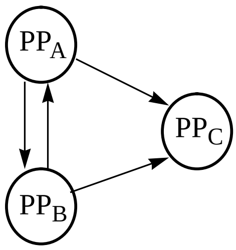


Figure 4 - Physical system for echoing

constant for a given simulation, the AAWP outperforms Time Warp arbitrarily.

5.1 Physical system

The system we consider for simulation was described previously in [LuWS89] as an example of “echoing” in Time Warp. We refer to it as EchoSystem. It consists of three physical processes (PP’s), A, B and C with the communication topology shown in Figure 4. Upon receiving a message from PP_A, PP_B processes it and sends a message to PP_A and vice-versa. If no message is received from the other, both of them prepare a message to send to PP_C. If a message arrives when one is being created or sent to PP_C, that sending is aborted and the new message is processed. Sending and receiving of messages takes no time. Processing a message from PP_A (PP_B) takes *u* real time units on PP_B (PP_A). Preparation of a message to PP_C takes 2*u* real time units. Suppose at time 0 PP_A receives the first message from PP_B. Then it may be verified that the only message traffic that occurs in this system is between PP_A and PP_B at intervals of *u* real time units. The idle periods between intervals are insufficient to build a message to PP_C. Note, real time in the physical system corresponds to logical time in the simulator.

For the simulator, we assume the following: processing a message between PP_A and PP_B takes one unit of wall-clock time (including sending the follow-on message), preparing a message to PP_C takes one time unit and sending of an antimessage also takes one time unit. LP’s advance their simulation clocks to the timestamp of the next event *after* executing that event. We assume this to simplify the proofs - the theorems can be proven even if it is assumed that logical clocks are advanced before commencing event execution. LP_C is assumed to perform its work fast enough so that its actions are irrelevant to the discussions and proofs. σ_X denotes the logical clock value of LP_X.

5.2 Time Warp execution

The Time Warp execution of this simulation is shown in Figure 5. The x-axis represents wall-clock time. The numbers (in multiples of *u*) at the junctions of the LP time lines and the unit time intervals indicate the logical time to which the LP’s have simulated (i.e. the logical clock value of the LP). A solid arrow indicates a message transmission while a dashed arrow indicates an antimessage transmission. The bold lines at various points on the time lines of LP_A and LP_B indicate rollback. From the picture, the echoing is evident immediately in the fact that the two LP’s roll back alternately, with increasing amplitudes.

Theorem 1: A Time Warp execution of EchoSystem takes *n(n+1)/2* wall-clock time units to simulate *nu* units of logical time.

Proof: The theorem is proved by induction on the amount of wall-clock time required for GVT to advance from logical time *(n-1)u* to *nu*.

Induction hypothesis: It takes *n* units of wall-clock time for GVT to advance from logical time *(n-1)u* to *nu*.

Base case: *n* = 1: From Figure 5 we see that in the first wall-clock time interval (i.e. wall-clock time [0:1)) LP_A advances σ_A to *u* and LP_B advances σ_B to *2u*. Thus, GVT has advanced from 0 to *u* in one wall-clock time unit and the hypothesis holds.

Induction step: Assume the hypothesis holds for the logical time window [*(n-1)u*:*nu*). Recall that the message traffic in the physical system consists only of the single message being exchanged by LP_A and LP_B. Thus, in any correct simulation of this system, GVT advances by *u* logical time units each time an LP receives this message, processes it and sends it back. During this process, the actions of other LP’s cannot affect GVT. Without loss of generality, assume LP_A takes *n* wall-clock time units to advance GVT from *(n-1)u* to *nu* (by induction hypothesis). In these *n* wall-clock time units, LP_B will send *n* messages to LP_C. Thus, at the end of [*(n-1)u*:*nu*), LP_A has just sent a message to LP_B with timestamp *nu* and LP_B has sent *n* false messages to LP_C. Therefore, LP_B takes *n* wall-clock time units to send the *n* antimessages to LP_C and one wall-clock time unit to process the message, advance σ_B to *(n+1)u* and send the message back to LP_A with timestamp *(n+1)u*. At this point, GVT will have advanced to logical time *(n+1)u*, requiring *n+1* wall-clock time units to do so. Thus the total wall-clock

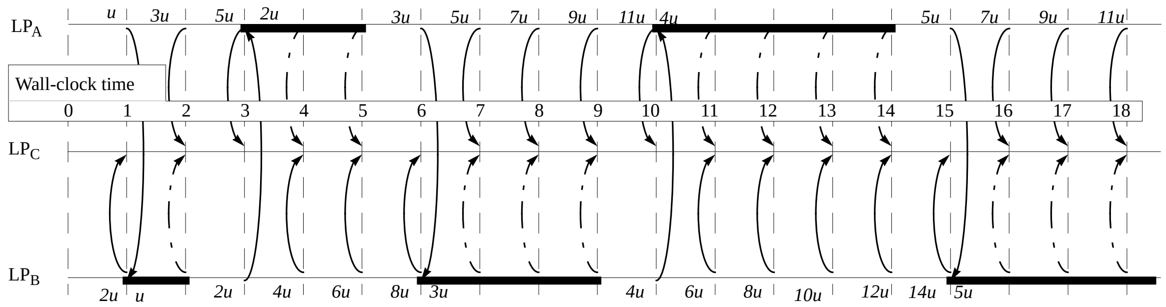


Figure 5 - Echoing in Time Warp

time required to simulate up to logical time *nu* is $\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$. ∎

We have thus shown that Time Warp takes $O(n^2)$ wall-clock time to simulate the specified physical system, where *n* is a measure of the logical time span of the simulation run.

5.3 AAWP execution

We describe a specific NPSI adaptive protocol based on the design framework described in section 4. The error potential (EP_i) of each LP_i is given by the following M_1:

$$M_1 : EP_i = \sigma_i - GVT_i$$

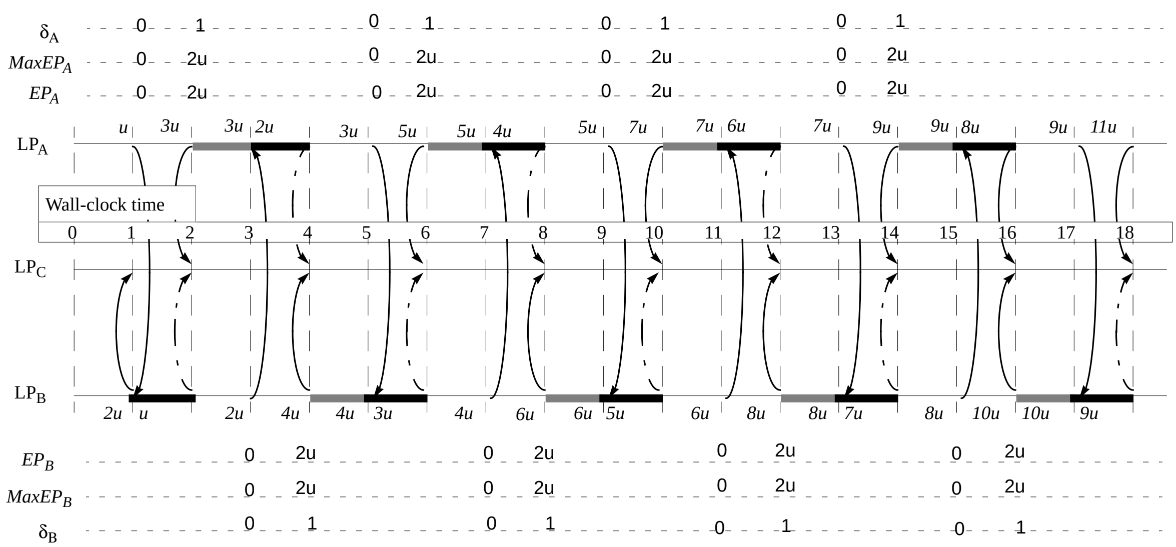
where GVT_i is the value of GVT made available to LP_i by the feedback system. M_2 is a function that maps EP_i into a wall-clock time delay, δ_i, given by:

$$M_2 : \delta_i = \begin{cases} \frac{EP_i}{MaxEP_i} & EP_i > 0 \\ 0 & EP_i = 0 \end{cases}$$

where $MaxEP_i$ is the maximum value of EP_i observed thus far. After executing an event, LP_i re-computes δ_i and waits for δ_i units of wall-clock time before proceeding to the next event. If a message is received during a wait period that will cause a rollback, the LP aborts the waiting and proceeds to roll back. Since this algorithm is a variant of the *elastic time algorithm* (ETA) described in [SrRe94], we will refer to it by the same name.

Figure 6 shows the execution of the simulation using the protocol described above. The shaded lines represent waiting due to the adaptive protocol. It is evident from the diagram that the echoing observed under Time Warp (Figure 5) has been avoided since each rollback is of only unit length.

Figure 6 - Avoiding echoing with adaptive aggressiveness



The discussion in section 4 justifies the following assumption:

NPSI Assumption: A change in an LP's logical clock value is reflected in the values of GVT visible to the different LP's in a fraction of the time it takes for an LP to execute an event.

For example, we may assume this latency is equal to 0.1 wall-clock time units since an LP takes one wall-clock time unit to execute an event.

THEOREM 2: An execution of EchoSystem using ETA takes $2n$-1 wall-clock time units to simulate nu units of logical time.

PROOF: The proof consists of induction on the amount of wall-clock time required for GVT to advance from logical time $(n-1)u$ to nu.

Induction hypothesis: For n = 2, 3, ... (i) it takes 2 units of wall-clock time for GVT to advance from logical time $(n-1)u$ to nu, (ii) $MaxEP_A <= 2u$, and (iii) $MaxEP_B <= 2u$.

Base case: $n = 2$: Referring to Figure 6, at wall-clock time 1 LP_A is at logical time u and has sent a message to LP_B while LP_B is at logical time $2u$ and has sent a false message to LP_C. Thus GVT = u at wall-clock time 1. The message from LP_A causes LP_B to roll back to logical time u and send an antimessage to LP_C, requiring one wall-clock time unit. In the wall-clock time interval [2:3), LP_B processes the message, advances σ_B to $2u$ and sends a message with timestamp $2u$ back to LP_A. In the wall-clock time period [1:2), LP_A advances σ_A to $3u$, builds a message and sends it to LP_C. Since LP_B rolls σ_B back to u just after wall-clock time 1, by the NPSI assumption, at wall-clock time 2 we have $\sigma_A = 3u$, $GVT_A = u$ and therefore $EP_A = 2u$. Since $MaxEP_A$ was 0 initially, $MaxEP_A = 2u$, giving $\delta_A = 1$. Thus LP_A waits during the wall-clock time period [2:3). Consequently, at wall-clock time 3, GVT = $2u$, $MaxEP_A = 2u$ and $MaxEP_B = 0$.

Induction step: Assume the hypothesis holds for the logical time window $[(n-1)u:nu)$. As in the proof of Theorem 1, the argument is made that in any correct simulation of the specified physical system, (i) GVT advances by u logical time units each time an LP receives a message, processes it and sends it back and (ii) the actions of other LP's during this period cannot influence this GVT advance. Without loss of generality, assume LP_A takes 2 wall-clock time units (by hypothesis), say $[i-1:i)$ and $[i:i+1)$, to advance GVT from $(n-1)u$ to nu and send a message with timestamp nu to LP_B. In $[i-1:i)$, LP_B advances σ_B from $(n-1)u$ to $(n+1)u$ and sends a message to LP_C. Since LP_A sent a message with timestamp nu to LP_B at the end of $[i:i+1)$, σ_A must be $(n-1)u$ at the end of $[i-1:i)$. Further, LP_A received a message from LP_B at the beginning of $[i-1:i)$ (because LP_A advances GVT). This implies LP_A could not have built and sent a message to LP_C during $[i-1:i)$. Therefore, LP_A must have rolled back during $[i-1:i)$. This means σ_A must have been rolled back to $(n-1)u$ at the beginning of $[i-1:i)$. By the NPSI assumption, at the end of $[i-1:i)$ $GVT_B = (n-1)u$. Thus, $\sigma_B = (n+1)u$, $EP_B = 2u$, $MaxEP_B = 2u$ (by hypothesis) and $\delta_B = 1$. Consequently, LP_B waits during $[i:i+1)$. At the end of $[i:i+1)$, GVT = nu, LP_A has sent a message with timestamp nu to LP_B and LP_B has sent one false message to LP_C. The message from LP_A at the end of $[i:i+1)$ causes LP_B to roll back and send one antimessage to LP_C (requiring one wall-clock time unit). LP_B uses another wall-clock time unit to process the message, advance σ_B to $(n+1)u$ and send a message with timestamp $(n+1)u$ to LP_A. Thus, after two wall-clock time units, GVT is advanced from nu to $(n+1)u$. Since LP_B controls the advance of GVT in these two wall-clock time units, $EP_B = 0$ and consequently, $MaxEP_B = 2u$. The actions of LP_A in this period are exactly the same as those of LP_B in the interval $[(i-1)u:(i+1)u)$ described earlier. It follows that $MaxEP_A$ also equals $2u$.

The total wall-clock time required to simulate nu units of logical time is therefore given by the sum of the wall-clock times required to simulate the windows $[0:u)$ and $[u:nu)$:

$$1 + \sum_{i=2}^{n} 2 = 2 \cdot n - 1 \qquad \blacksquare$$

We have thus shown that ETA takes $O(n)$ wall-clock time to simulate EchoSystem, where n is a measure of the logical time span of the simulation run. The corresponding completion time with Time Warp is $O(n^2)$. Since the difference in completion times is not bounded by a constant for a given simulation, the adaptive protocol outperforms Time Warp arbitrarily.

5.4 Pragmatic issues

We have described a specific AAWP, the elastic time algorithm (ETA) and shown that it can outperform Time Warp arbitrarily. While the simulation for which this phenomenon is observed is somewhat contrived, ETA is not. In [SrRe94] we describe an experiment on a four-processor Time Warp implementation using a workload very similar to EchoSystem. The experimental set-up included a prototype high-speed reduction network which was used to provide near-perfect state information to implement ETA. In conformance with our analysis here, we observed that the speedup of ETA over Time Warp increased with the logical time span of the simulation, i.e., the larger the maximum simulated time, the larger the speedup.

6 Summary

The lack of consistent performance with the two traditional approaches to synchronization in parallel discrete event simulations (conservative and optimistic) has led to a number of hybrid approaches. Many of these have indeed demonstrated better performance under test cases. An adaptive protocol, one that modifies itself in response to changes in the simulation, appears to be the most likely to perform well with a wide range of simulations. However, there have been no analytical studies comparing the performance of adaptive protocols with that of traditional protocols. We present the first known analytical comparison of adaptive protocols with Time Warp. We demonstrate that it is possible for Time Warp to arbitrarily outperform a class of adaptive protocols we call asynchronous adaptive wait protocols (AAWP's). Protocols in this class control aggressiveness and risk by introducing independently controlled delays at the LP's. This class is general enough to include many practical protocols. Conversely, we describe a member of a new class of adaptive protocols called NPSI adaptive protocols (which are a subset of AAWP's), and present an example in which this protocol outperforms Time Warp arbitrarily. Thus, while adaptive limiting of optimism appears to enhance performance in practice, our study shows that care must be taken in the design of AAWP's since incorrect adaptive decisions can lead to arbitrarily worse performance than Time Warp.

Acknowledgments

We are grateful to Bronis de Supinski for proof reading the paper and for his insightful comments. We thank Craig Williams for her suggestions. This work was supported by Mystech, Inc. (Academic Affiliates Program).

References

[BaHo90] Ball, D. and Hoyt, S., “The adaptive Time-Warp concurrency control algorithm”, *Proceedings of the SCS Multiconference on Distributed Simulation*, January 1990, 174-177.

[ChMi79] “Distributed Simulation: A case study in the design and verification of distributed programs”, *IEEE Transactions on Software Engineering*, Vol. SE-5, No. 5 (September 1979), 440-452.

[FeTr94] Ferscha, A. and Tripathi, S.K., “Parallel and distributed simulation of discrete event systems”, University of Maryland at College Park Technical Report number CS-TR-3336, August 1994.

[Fuji90] Fujimoto, R.M., “Parallel discrete event simulation”, *Communications of the ACM*, Vol. 33, No. 10, October 1990, 30-53.

[Gima89] Gimarc, R.L., “Distributed simulation using hierarchical rollback”, *Proceedings of the 1989 Winter Simulation Conference*, 1989, 621-629.

[HaTr94] Hamnes, D.O. and Tripathi, A., “Evaluation of a local adaptive protocol for distributed discrete event simulation”, *Proceedings of the 1994 International Conference on Parallel Processing*, August 1994, Vol. III, 127-134.

[Jeff85] Jefferson, D.R., “Virtual time”, *ACM Transactions on Programming Languages and Systems*, Vol. 7, No. 3, July 1985, 404-425.

[LiMi90] Lipton, R.J. and Mizell, D.W., “Time Warp vs. Chandy-Misra: A worst-case comparison”, *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, January 1990, 137-143.

[LuWS89] Lubachevsky, B., Weiss, A. and Shwartz, A., “Rollback sometimes works . . . if filtered”, *Proceedings of the 1989 Winter Simulation Conference*, December 1989, 630-639.

[LuWS91] Lubachevsky, B., Weiss, A. and Shwartz, A., “An analysis of rollback-based simulation”, *ACM Transactions on Modeling and Computer Simulation*, Vol. 1, No. 2, April 1991, 154-193.

[Madi93] Madisetti, V.K., “Randomized algorithms for self-synchronization”, Private communication, 1993.

[McAf90] McAffer, J., “A unified distributed simulation system”, *Proceedings of the 1990 Winter Simulation Conference*, 1990, 415-422.

[NiRe90] Nicol, D.M. and Reynolds, P.F., Jr., “Optimal dynamic remapping of parallel computations”, *IEEE Transactions on Computers*, Vol. 39, No. 2, February 1990.

[PaRe93] Pancerella, C.M. and Reynolds, P.F., Jr., “Disseminating critical target-specific synchronization information in parallel discrete event simulations”, *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, May 1993, 52-59.

[RaAT93] Rajaei, H., Ayani, R. and Thorelli, L-E., “The Local Time Warp approach to parallel simulation”, *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, May 1993, 119-126.

[ReJe89] Reiher, P.L. and Jefferson, D., “Limitation of optimism in the Time Warp operating system”, *Proceedings of the 1989 Winter Simulation Conference*, December 1989, 765-770.

[Reyn82] Reynolds, P.F., Jr., “A shared resource algorithm for distributed simulation”, *Proceedings of the 9th Annual Symposium on Computer Architecture*, April 1982.

[Reyn88] Reynolds, P.F., Jr., “A spectrum of options for parallel simulation”, *Proceedings of the 1988 Winter Simulation Conference*, December 1988, 325-332.

[SrRe93] Srinivasan, S. and Reynolds, P.F., Jr., “Non-interfering GVT computation via asynchronous global reductions”, *Proceedings of the 1993 Winter Simulation Conference*, December 1993, 740-749 (available by anonymous FTP from ftp.cs.virginia.edu/pub/techreports/CS-93-17.ps.Z).

[SrRe94] Srinivasan, S. and Reynolds, P.F., Jr., “NPSI adaptive synchronization algorithms for PDES”, Technical report number CS-94-44, Department of Computer Science, University of Virginia, November 8, 1994 (submitted to the 1995 Winter Simulation Conference - available by anonymous FTP from : ftp.cs.virginia.edu/pub/techreports/CS-94-44.ps.Z).

[Stei93] Steinman, J.S., “Breathing Time Warp”, *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, May 1993, 109-118.