

Elastic Time

SUDHIR SRINIVASAN

Mystech Associates, Inc.

PAUL F. REYNOLDS, JR.

University of Virginia

We introduce a new class of synchronization protocols for parallel discrete event simulation, those based on *near-perfect state information* (NPSI). NPSI protocols are adaptive, dynamically controlling the rate at which processes constituting a parallel simulation proceed, with the goal of completing a simulation efficiently. We show by analysis that a class of adaptive protocols (that includes NPSI and several others) can both arbitrarily outperform and be arbitrarily outperformed by the Time Warp synchronization protocol. This mixed result both substantiates the promising results we and other adaptive protocol designers have observed, and cautions those who might assume that any adaptive protocol will always be better than any nonadaptive one. We establish in an experimental study that a particular NPSI protocol, the *Elastic Time Algorithm*, outperforms Time Warp, both temporally and spatially, on every workload tested. Although significant options remain with respect to the design of ETA, the work presented here establishes the class of NPSI protocols as a very promising approach.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures—*multiple-instruction-stream, multiple-data-stream processors*; C.4 [**Computer Systems Organization**]: Performance of Systems; D.1.3 [**Programming Techniques**]: Concurrent Programming—*parallel programming*; D.4.1 [**Operating Systems**]: Process Management—*synchronization*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*discrete event, Monte Carlo, parallel*

General Terms: Algorithms, Documentation, Experimentation, Performance

Additional Key Words and Phrases: Adaptive protocols, aggressiveness, near-perfect state information, optimistic protocols, risk

1. INTRODUCTION

Discrete-event simulations have proven to be difficult to parallelize, despite significant amounts of potential parallelism. Studies indicate parallel discrete event simulation (PDES) is a viable technology. (Fujimoto [1993] provides an excellent survey.) However, performance is usually dependent

Authors' addresses: S. Srinivasan, Mystech Associates, Inc., 5205 Leesburg Pike, Suite 1200, Falls Church, VA 22041; P. F. Reynolds, Jr., Department of Computer Science, University of Virginia, Olsson Hall, Charlottesville, VA 22903; email: reynolds@Virginia.edu.

Permission to make digital/hard copy of part or all of this work for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage, the copyright notice, the title of the publication, and its date appear, and notice is given that copying is by permission of the ACM, Inc. To copy otherwise, to republish, to post on servers, or to redistribute to lists, requires prior specific permission and/or a fee.

© 1998 ACM 1049-3301/98/0400-0103 \$5.00

on characteristics of simulated applications (even particular instances). Thus, a consistently efficient approach has proven to be elusive for nearly 20 years [Nicol and Heidelberger 1995]. The bulk of synchronization protocols have failed to be consistently efficient because they do not adapt to the dynamic, unpredictable nature of the synchronization requirements of parallel simulations [Nicol and Reynolds 1990; Fujimoto 1990]. To this end, *adaptive synchronization protocols*—those that change the bindings of one or more of their design variables dynamically [Reynolds 1988]—offer a promising approach. (See discussion in ORSA [1993].) The primary contributions of this article are to define and establish a new class of adaptive synchronization protocols, the NPSI adaptive synchronization protocols, and demonstrate (both theoretically and empirically) their potential to be consistently efficient.

The common approach to parallelizing discrete event simulations is to model the physical system as a collection of separate physical entities called physical processes (PPs). PPs interact with each other using messages. In the parallel discrete event simulator, each PP is simulated by a logical process (LP) which is essentially a sequential discrete event simulator with its own logical clock and events list and the state variables that describe the PP it simulates. The interactions among PPs are captured via the exchange of timestamped messages between LPs over logical channels; there is no shared state. Since the messages exchanged by PPs usually result in some activity at receiving PPs, receipt of a timestamped message at an LP usually schedules one or more events or affects the state of the LP receiving the message. The central problem in PDES is one of synchronization: to determine efficiently when an LP should simulate a scheduled event, allowing for the arrival of events from other LPs.

Many synchronization protocols have been proposed [Fujimoto 1990] to solve the synchronization problem. These can be classified broadly as nonaggressive and aggressive, based on the strategy for simulating events. With a nonaggressive (or conservative) protocol, an LP executes an event only after determining that the event is safe; that is, execution of that event will not result in an error. With an aggressive protocol, LPs execute events without the guarantee of freedom from errors; in the event an error does occur, the common approach is to recover from the error using a rollback mechanism. Most aggressive protocols also include risk, the property by which results of aggressive processing are propagated to other LPs [Reynolds 1988]. Aggressiveness and risk are collectively referred to as *optimism*.

We propose a new class of PDES protocols, the NPSI adaptive protocols, characterized by the use of near-perfect state information (NPSI) to control optimism adaptively. NPSI protocols operate by computing an error potential (EP) based on NPSI and translating the EP into control over aggressiveness and risk. The rate of progress in simulated time of an LP is controlled by its EP, which is typically a function of the states of other LPs. EP can be thought of as an elastic force that sometimes restricts and sometimes allows the progress of the LP in simulated time, hence, *elastic*

time. We demonstrate that an adaptive approach based on low cost near-perfect state information has significant potential to yield a consistently efficient synchronization protocol.

We propose the Elastic Time Algorithm (ETA), a promising NPSI protocol. ETA uses temporal information (logical clock, unreceived message time, and next event time) to compute EP and it controls optimism by introducing delays proportional to EP between event executions. The delays are asynchronous and computed independently for each LP. We call this the *microadaptive* approach, as opposed to the *macroadaptive* approach where adaptiveness occurs in the form of relatively infrequent changes to system parameters based on system monitoring. A performance study of ETA shows it to be highly efficient over a wide range of workloads and scalable up to moderately sized parallel systems, at least.

2. RELATED WORK

The optimistic approach is widely believed to be more likely to yield a consistently efficient protocol [Fujimoto 1990]. However, since uncontrolled optimism is prone to poor performance, an approach with controlled optimism is desirable. We categorize previous approaches based on controlling criteria.

2.1 Window Based

The earliest protocols applied a loosely synchronous approach. Between synchronizations, LPs execute only those events that have timestamps within an agreed-upon window. Since the difference between the logical clocks of any two LPs is bounded, it is possible to limit the lengths of rollback chains and therefore control the cascading of rollbacks. Several aggressive windowing algorithms have been proposed: Moving Time Windows [Sokol et al. 1988], Filtered Rollback [Lubachevsky et al. 1989], Window-Based Throttling [Reiher and Jefferson 1989], Unified Distributed Simulation System [McAffer 1990], Bounded Time Warp [Turner and Xu 1992], Aggressive Global Windowing Algorithm [Dickens 1993], and Breathing Time Warp [Steinman 1993]. Basic differences lie in the methods used to determine the time window. As with nonaggressive windowing protocols, the primary limitation of aggressive windowing protocols is the determination of the window size, which can affect performance significantly. Optimal size can depend on the nature of the application, and distribution of the computational load, among others.

2.2 Space Based

Here, optimism is limited by spatial boundaries rather than temporal boundaries. LPs are divided into clusters, each of which operates optimistically internally. Interaction among clusters is without risk (i.e., messages are exchanged only after it is determined that they are safe). Erroneous computation is limited to a cluster so that when a rollback occurs, the length of the rollback chain will generally be smaller. Examples of this

technique are Hierarchical Rollback [Gimarc 1989] and Local Time Warp [Rajaei et al. 1993]. Physical limits on the propagation of messages cannot guarantee the absence of severe cascading of rollbacks (e.g., see the CHASE workload in Section 7.6.1). Thus, although the lengths of rollback chains will be smaller in general, echoing [Lubachevsky 1989] is still possible.

In a special class of space-limited aggressive protocols each cluster is a single LP. Rollbacks are local to each LP and echoing and cascading rollbacks are avoided. Examples are SRADS with local rollback [Dickens et al. 1990], Speculative Computing [Mehl 1991], Breathing Time Buckets [Steinman 1991], and Risk-Free Time Warp [Bellenot 1993]. In this approach, without risk, the benefits of aggressive processing are limited to individual LPs. The space-based approach suffers from the pitfalls of both the conservative and the optimistic techniques: (i) risk-free interaction between clusters introduces the problems of conservative protocols: potential for deadlock, sensitivity to lookahead, and overheads of communicating lack-of-dependence information; and (ii) aggressive processing introduces state saving and rollback costs. Rollback costs are very low in the special case of one LP per cluster, but they can be very expensive in the more general case.

2.3 Penalty Based

Each LP's optimism is controlled by its recent behavior. Based (usually) on rollback behavior, some LPs are penalized (and block) whereas others are favored (and continue executing). Good performance requires accurate prediction of the future, based on observed (local) behavior. Four protocols have been proposed: Penalty-Based Throttling [Reiher and Jefferson 1989], where any LP that sends an antimessage is penalized; Adaptive Time Warp [Ball and Hoyt 1990], which penalizes an LP during a time it is processing forward for rolling back often in the recent past; Randomized Self-Synchronization [Madisetti 1993], where an LP estimates logical clock values of other LPs and based on the difference between these estimates and its own logical clock value, it either proceeds aggressively or suspends processing; and Composite ELSA [Arvind and Smart 1992], where the degrees of optimism and risk of LPs are controlled using speculation functions and local measures.

The penalty-based approach appears very promising. So far, it has been based either solely on local states or on estimates of nonlocal states. To achieve good performance, LPs must react dynamically to changing system state, which is best done by obtaining and using actual nonlocal state information, as in our NPSI approach.

2.4 Knowledge Based

This approach restricts the propagation of an incorrect computation (one leading to a primary rollback) as soon as an LP determines it to be incorrect. The LP broadcasts a message indicating the potentially erroneous nature of their execution to all affected LPs that consequently limit

their optimistic processing. Examples of this approach are: Wolf [Madisetti et al. 1988], which takes a conservative approach and stops all potentially incorrect computation, and Filter [Prakash and Subramanian 1991], which stops only those computations that are exactly known to be incorrect. The effectiveness of this technique is limited since it is reactive rather than proactive: by the time the primary rollback occurs, the incorrect computation may have proliferated so that rollbacks will be required to recover from the errors.

2.5 Probabilistic

MIMDIX [Madisetti et al. 1993] incorporates probabilistic synchronization. Here, a process called a genie probabilistically sends a synchronization message to all LPs periodically, causing them to synchronize to the timestamp of the message. Those LPs that are ahead of this timestamp are rolled back. By keeping the timestamp of the synchronization message close to GVT, all LPs can be kept temporally close to each other, thus reducing the risk of cascading rollbacks. Problems with this approach are that it does not restrict only incorrect computations, it requires an efficient, high priority broadcasting mechanism, and an effective probability for controlling the synchronization is very application dependent.

2.6 State Based

State-based protocols differ from those preceding in two significant ways: they are adaptive in that LPs continually adjust their optimism, and adaptive decisions are based on state information that, although available locally, is directly affected by the actions of other LPs. Two of these protocols [Hamnes and Tripathi 1994; Ferscha and Tripathi 1994] are similar in that they both utilize information about channels to decide when and for how long LPs should wait. Intuitively, it may be argued that the NPSI approach has an advantage over these protocols because LPs in NPSI protocols can receive information from more than immediate predecessors. A third protocol [Das and Fujimoto 1994] uses memory consumption as the basis for limiting optimism. It limits the memory consumption of LPs adaptively and, consequently, also limits their optimism. It is based on a memory management protocol such as Cancelback or Artificial Rollback [Lin 1992b]. We take the opposite approach in that NPSI adaptive protocols limit the optimism of LPs directly and, consequently, limit their memory consumption, thus eliminating the need for costly memory management schemes.

2.7 Commentary

A survey of adaptive protocols is also provided in Das [1996]. Several of these protocols have demonstrated better performance than Time Warp in certain cases. Controlling optimism can reduce rollback costs and improve performance. However, these protocols will have limited performance due to one or more of the following: (i) the criterion for controlling optimism

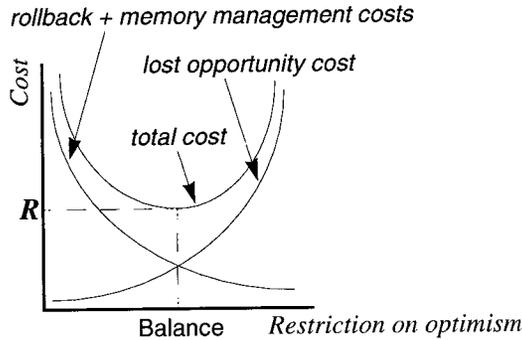


Fig. 1. Tradeoff introduced by controlling optimism.

(window size, cluster size, penalty thresholds, probabilities, etc.) is predetermined; (ii) the controlling decision is based solely on local history; or (iii) the LPs operate synchronously.

3. NEAR-PERFECT STATE INFORMATION

Optimistic protocols incur three time costs: state saving, rollback, and memory management. Limiting optimism introduces a fourth, lost opportunity cost, characterized by the potential loss in performance when an LP stops executing events or sending messages even though it is safe for it to continue. The time cost of state saving is a function of the size of the state that is modified by events and the frequency of state saving, rather than the level of optimism and thus can be ignored here. To obtain good performance, protocols that control optimism must minimize the cost function:

$$\begin{aligned} \text{total cost} = & \text{rollback cost} + \text{memory management cost} \\ & + \text{lost opportunity cost.} \end{aligned}$$

Although limiting optimism decreases the first two costs, it increases the third, leading to the tradeoff shown in Figure 1. The best possible performance for protocols with controlled optimism is characterized by $R = 0$, where R is the residual total cost when the balance is reached. $R = 0$ when controlled optimism eliminates both rollback and memory management costs without adding lost opportunity cost.

To attain good balance, protocols must identify incorrect computations and limit their propagation. Unfortunately, PDES synchronization requirements are dynamic, irregular, and data-dependent [Nicol and Reynolds 1990; Fujimoto 1990]. Typically, this is due to the fact that simulated systems have an information flow that is translated in the simulator into a causal chain of events among LPs. Since the propagation of such chains is based on probabilistic decisions and input parameters, it is impossible to determine the flows a priori except in special cases. Consequently, we believe the two key requirements for a consistently efficient protocol are

that it is adaptive and that it uses feedback from the simulation. Ideally, these requirements are satisfied by providing LPs with perfect state information, such that any relevant change in system state is visible instantaneously. Since perfect state information is impossible to achieve in practice, we explore the use of near-perfect state information. We assume the existence of a feedback system that operates asynchronously with respect to the LPs and provides them with NPSI at low cost. We describe the computational model for the feedback system next, and discuss how it may be realized in practice.

Reduction Model

In the reduction model of computation, each LP encodes the relevant parts of its state into a set of values, V_i^j , called an input state vector (ISV). These ISVs are processed by an NPSI Calculator (NPSIC) to produce output state vectors (OSV) for each LP as in the following.

Let $\mathbf{ISV}_1, \mathbf{ISV}_2, \dots, \mathbf{ISV}_n$ be the input state vectors submitted by the n LPs such that $\mathbf{ISV}_i = \langle V_i^1, V_i^2, \dots, V_i^m \rangle$, where m is the size of each LP's state vector. Let $\theta_k, 1 \leq k \leq m$ be binary associative operators. Then for each LP _{i} the NPSI Calculator computes $\mathbf{OSV}_i = \langle \text{TS}_i^1, \text{TS}_i^2, \dots, \text{TS}_i^m \rangle$, where $\text{TS}_i^k := \theta_k \langle V_i^k \rangle \forall l \in P_i, 1 \leq k \leq m$ and P_i is the predecessor set of LP _{i} .

The predecessor set of LP _{i} could be any subset of the n LPs. Typically, the subset is based on the communication graph defined by the LPs such that a directed edge exists from LP _{a} to LP _{b} if LP _{a} can send a message to LP _{b} . LP _{i} 's predecessor set includes LP _{j} if there exists a path from LP _{j} to LP _{i} in the communication graph. Since the predecessor set can be different for different LPs, the information computed by the NPSIC is called *target-specific information* [Pancerella and Reynolds 1993]. In the case where the communication graph is strongly connected (i.e., there exists a path between every ordered pair of nodes), target-specific information is equivalent to global information.

Since LPs and the NPSIC are mutually asynchronous, we use state vectors to force the interactions between them to be atomic: an LP must provide a complete ISV and must read a complete OSV. However, it is not required that every LP _{i} reads every OSV _{i} generated by the NPSIC. We have established elsewhere [Reynolds et al. 1993] that the atomicity criterion is sufficient to provide sequential consistency, a property that facilitates the design of provably correct synchronization algorithms using this model. The reduction model is sufficiently powerful to provide useful information such as global virtual time [Srinivasan and Reynolds 1993].

The reduction model can be realized in practice on a shared memory architecture by dedicating a set of processes to the task of computing OSVs using ISVs stored in shared memory that are updated by other processes participating in the simulation. Examples of similar computations can be found in Lin [1992a] and Fujimoto and Hybinette [1997]. On distributed memory architectures, the reduction model may be realized using a high-

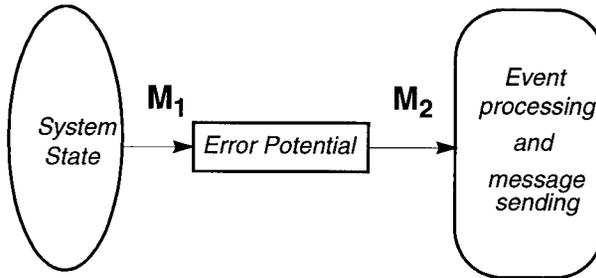


Fig. 2. General framework for adaptive protocols.

speed asynchronous reduction network such as the parallel reduction network (PRN) [Reynolds et al. 1993]. Our experience with the design, construction, and testing of the PRN suggests that a production version of such a network can operate at very high speeds (less than 20 nanoseconds per stage in the tree). These low latencies, combined with its pipelined tree structure make such a network scalable up to thousands of processors. We have used the PRN to implement and study ETA (Section 7.5). A global reduction network implements a special case of the general reduction model in which all OSVs are identical and are computed using information from all ISVs. Although a target-specific reduction network that implements the more general model has not been constructed yet, the results of Pancerella and Reynolds [1993] indicate its feasibility. Finally, we note many high-performance computers support reduction operations in software, and recently, we have implemented reductions efficiently on a Myrinet-based network.

4. NPSI ADAPTIVE PROTOCOLS

NPSI adaptive protocols are optimistic protocols that control the aggressiveness and risk of LPs dynamically using near-perfect state information. There are two phases in the design of NPSI protocols:

- identifying the state information on which the decision to control optimism is to be based; and
- designing the mechanism that translates this information into control over an LP's optimism.

There are many choices for each of these. To facilitate independent study of each, we decouple them by introducing an error potential (EP_i) associated with each LP_i , to control LP_i 's optimism. The framework we propose is shown in Figure 2. The NPSI adaptive protocol keeps each EP_i up to date as the simulation progresses, by evaluating M_1 at high frequency using state information it receives from the feedback system. Similarly, M_2 dynamically translates new values of EP_i into delays in the event execution and communication rates. Different protocols may be devised by changing M_1 and M_2 . Our goal is to identify mappings M_1 and M_2 that produce efficient adaptive protocols.

4.1 M_1 —Computing EP

EP_i indicates the likelihood of LP_i 's computation becoming incorrect in the near future. The key to consistently good performance is an M_1 that will predict with high accuracy whether an LP's computation will be rolled back. An inaccurate M_1 can produce a low EP when the computation is erroneous, resulting in higher rollback costs, or a high EP when the computation is correct, resulting in higher lost opportunity cost.

The information used in computing M_1 must be such that it can be obtained using the reduction model described in Section 3. One such reduced value is global virtual time (GVT) [Jefferson 1985]. To illustrate the nature of M_1 , we list some possible mappings.

- (1) $EP_i =$ number of local events executed with timestamps $>$ GVT. This mapping is based on two concepts: the cost of a rollback is generally proportional to the number of events rolled back; and no LP can roll back to a logical time smaller than GVT. Thus, the EP will be proportional to the maximum possible rollback depth. Unfortunately, this mapping does not capture the likelihood of an LP rolling back. For instance, consider two LPs, one at logical time 100 having executed 2 events with timestamps $>$ GVT and another at logical time 50 having executed 10 events with timestamps $>$ GVT. This mapping will restrict the latter even though the former is more likely to be rolled back. This reasoning suggests that logical time should be included in the mapping. Note, the mapping described here is similar to the Breathing Time Warp algorithm [Steinman 1993] and is an example of how our framework can be used to represent different protocols.
- (2) $EP_i =$ logical clock of $LP_i -$ GVT. The rationale is that if an LP is far ahead of others in logical time and may receive messages from them, it is quite likely to be rolled back. This mapping remedies the problem described in (1).
- (3) $EP_i = \eta_i -$ GVT where η_i is the time of the next scheduled event at LP_i . It is reasonable to expect this mapping to outperform mapping (2) since η_i is a predictor of LP_i 's next event whereas LP_i 's logical clock only describes the current state. Unfortunately, the LP with minimum η_i cannot determine the fact that its next event is safe when there are no messages in transit and therefore will continue to wait for an amount of time proportional to its EP_i , which can be arbitrarily long, even though it could proceed. Thus, this mapping can lead to substantial lost opportunity. This drawback is eliminated in the M_1 used in the Elastic Time Algorithm described in Section 6.1.

4.2 M_2 —Controlling Aggressiveness and Risk

M_2 is a function that translates a value of EP into control over the aggressiveness and risk of an LP. It must be designed such that higher values of EP_i result in less aggressiveness and/or risk at LP_i . A simple scheme is to establish a threshold such that whenever the value of EP

exceeds this threshold, event execution and communications are suspended. A more sophisticated scheme would reduce the event execution and communication rates gradually as EP increases. This deceleration can be achieved by inserting delays at appropriate points. M_2 would then be a function that maps EP to a wall-clock time delay.

An LP that sends messages to other LPs based on aggressive computation exhibits risk [Reynolds 1988]. Since risk contributes to rollback costs, some mechanism should be included to control it. This mechanism could be similar to the one used to control aggressiveness: an LP could either stop sending messages while EP exceeds a threshold or it could gradually decrease the rate at which it sends messages as EP increases. Another possibility is a common mechanism controlling both aggressiveness and risk. By controlling the risk of LPs, we can generate any hybrid scheme between the two extremes: an approach with unlimited risk such as Time Warp, and a risk-free approach such as SRADS with Local Rollback [Dickens and Reynolds 1990].

5. ADAPTIVE ALGORITHMS VERSUS TIME WARP: AN ANALYSIS

It is important to understand the potential benefits and shortcomings of adaptive algorithms. We define a class of adaptive protocols, the asynchronous adaptive waiting protocols (AAWPs) and identify several published protocols that belong to this class. We show that: Time Warp can outperform an AAWP arbitrarily; and an NPSI protocol we call ZETA (which is also an AAWP), can outperform Time Warp arbitrarily. This study serves specific purposes. (1) It compares the performance of a general class of adaptive protocols with that of Time Warp and shows that although adaptiveness can improve relative performance, it can also result in arbitrarily worse performance. Thus adaptive protocols must be designed with care. (2) It also demonstrates that NPSI protocols are capable of outperforming Time Warp arbitrarily. Along with the empirical study of NPSI algorithms described later, this demonstrates the significant potential of NPSI algorithms to be consistently efficient. Our analysis is analogous to a previous comparison of Time Warp with the Chandy–Misra protocol [Lipton and Mizell 1990].

5.1 Assumptions

In this section, we assume each LP is located on its own processor and protocols employ aggressive cancellation and aggressive rollback. The following assumptions define asynchronous adaptive waiting protocols, 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.

- (1) The simulation loop of an LP is as follows. No assumptions are made about details 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.

```

Adaptive event execution loop
while not done
  Process event
  Wait for time  $\delta \geq 0$ 
  Rollback if necessary
  Process received messages
  Save state
  Collect fossils
endwhile

```

- (2) The waiting at each LP is asynchronous with respect to the waiting at other LPs.
- (3) We focus on event execution, rollback, and adaptive waiting, ignoring overheads such as state saving, receiving messages, GVT computation, and fossil collection for simplicity.
- (4) The AAWP does not increase the capability of the LPs to guess computation. An increase in the guessing power of an LP could compensate for any detrimental effects of the adaptive control. Since an LP's guessing capability depends entirely on the application, it is reasonable to assume the AAWP cannot increase this capability.
- (5) 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 rollback costs are not bounded, it can be shown trivially that an AAWP can perform arbitrarily worse than Time Warp.
- (6) 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, it could be shown trivially that an AAWP could perform arbitrarily worse than Time Warp.

Practical examples of AAWPs include Reiher and Jefferson [1989], Ball and Hoyt [1990], Madiseti [1993], McAffer [1990], Steinman [1993], Hamnes and Tripathi [1994], Ferscha and Tripathi [1994], and, of course, the NPSI protocols proposed here. Note, global windowing algorithms do not fit the AAWP model since the waiting is synchronized across all LPs.

5.2 Time Warp Outperforms AAWPs

We show by example that an AAWP can take arbitrarily longer than Time Warp to complete a simulation. The intuition is: it is possible for a Time Warp simulation to execute very efficiently, with few rollbacks; however, it is also possible for a Time Warp simulation to generate many false events and consequent rollbacks that can degrade its performance severely. Errors in adaptive decisions regarding when and how long to wait can cause a Time Warp execution to move from the first category to the second. We show a situation where the AAWP induces a false rollback chain that delays the committing of an event by at least an amount of time propor-

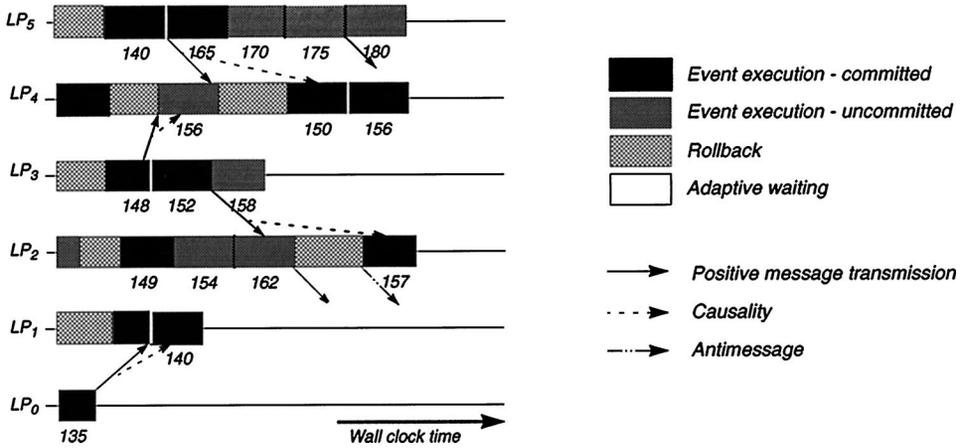


Fig. 3. Time Warp execution.

tional to the length of the rollback chain. By arguing that this rollback chain can be arbitrarily long, we establish that the committing of an event can be delayed arbitrarily.

Consider the Time Warp execution shown in Figure 3. Numbers below events are the events' 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). Two noteworthy events are: the event with timestamp 140 at LP₁, which is causally dependent on a message from LP₀ that arrives just in time to be executed by LP₁; and the event with timestamp 165 at LP₅, whose committing execution will be delayed due to an erroneous waiting decision.

Figure 4 shows an execution of the same simulation using an AAWP. We assume the same initial conditions, except, for the AAWP both LP₀ and LP₁ wait at the beginning of the execution. Due to an error in the decision process, LP₀ delays longer than LP₁. Thus, LP₁ is ready to execute an event before the message with timestamp 140 from LP₀ arrives. LP₁ executes its next event (with timestamp 150) and sends a message to LP₂. Since we assume aggressive rollback, this event is *false*: it is executed out of order. When the message from LP₀ arrives, the false event is rolled back and an antimessage is sent to LP₂. However, the message sent to LP₂ by the false event has initiated a chain of false events. The antimessage starts a chain of rollbacks and antimessages that follows close behind the chain of false events. The rollback chain could catch up with the false event chain and prevent it from reaching LP₅. However, it is also possible for the chains to persist [Lubachevsky et al. 1991], as shown in Figure 4. The central problem is that the false event chain results in a message M_F at LP₅ with timestamp 160, which is smaller than 165, the timestamp of event E_D at LP₅. 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

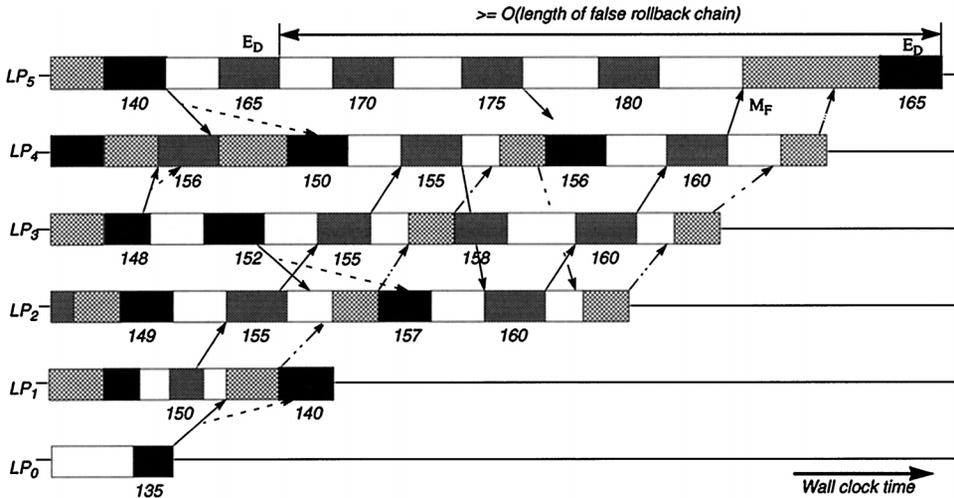


Fig. 4. False rollback chain with cycle due to incorrect waiting.

is delayed by at least an amount of time proportional to the length of the false rollback chain.

The two chains each have a cycle, involving LP_2 , LP_3 , and LP_4 . It is possible for the chains to revisit these LPs an arbitrary number of times before reaching LP_5 . 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 . Assuming correct implementations of both Time Warp and the application, logical time must advance as we traverse the false event chain. Therefore, an arbitrarily long chain would require that the timestamp of E_D be arbitrarily large. The larger the timestamp of E_D , the less likely its first execution is on the critical path of the simulation, since other LPs are further behind in logical time. Thus, the probability that a false chain is damaging to the simulation decreases with the length of the chain. However, the probability is not zero—consider a simulation where E_D marks the transition to a new phase of simulated time; that is, the simulation makes a “jump” in logical time to the next phase of activity. The delay of E_D could delay the entire phase transition. It is possible to have very long false chains that do not span much logical time. In Figure 4, logical time increases only across iterations and not inside any one iteration. Finally, even if the lengths of the chains are bounded, it is possible to have an arbitrary number of instances of such chains in a simulation (i.e., the number of instances is bounded only by the logical time span of a simulation run).

In summary, we have shown that the committing execution of an event can be delayed by an arbitrary amount of time due to false events and rollbacks created by errors in the waiting decision. In Srinivasan and Reynolds [1995] we show this analysis holds for systems employing lazy cancellation and lazy rollback as well.

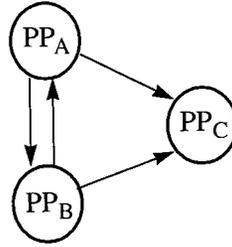


Fig. 5. EchoSystem.

5.3 AAWPs Outperform Time Warp

We show that Time Warp can take arbitrarily longer than an AAWP to complete a simulation; namely, Time Warp takes an amount of time that is quadratic in the amount of logical time simulated and the AAWP takes linear time. Since the difference in completion times is not bounded by a constant for a given simulation, the AAWP outperforms Time Warp arbitrarily.

5.3.1 Simulated System. The system we consider (henceforth called EchoSystem) was described in Lubachevsky et al. [1989]. It consists of three physical processes A, B, and C as shown in Figure 5. Upon receiving a message from PP_A , PP_B processes and returns it to PP_A and vice versa. If no message is received from the other, each prepares a message to send to PP_C . Arrival of a message terminates the sending of a message to PP_C (if any). Sending and receiving messages takes no time. PP_A (PP_B) takes u real time units to process a message from PP_B (PP_A). Preparation of a message to PP_C takes $2u$ real time units. If at time 0, PP_A sends the first message to PP_B , message traffic between PP_A and PP_B occurs at intervals of u real time units. Note, real time in the physical system corresponds to logical time in the simulator.

We assume the wall-clock time required for processing a message between PP_A and PP_B (including sending the follow-on message), preparing a message to PP_C , and sending an antimessage, is one time unit each. We assume LPs advance their simulation clocks to the timestamp of the next event after executing that event; our theorems are true for the before case as well. LP_C is assumed to perform its work fast enough that its actions are irrelevant to the analysis. σ_X denotes the logical clock value of LP_X .

5.4 Time Warp Execution

The Time Warp execution of EchoSystem is shown in Figure 6, from which the echoing is evident. The numbers (in multiples of u) along the LP time lines indicate the logical clock values of the LPs.

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

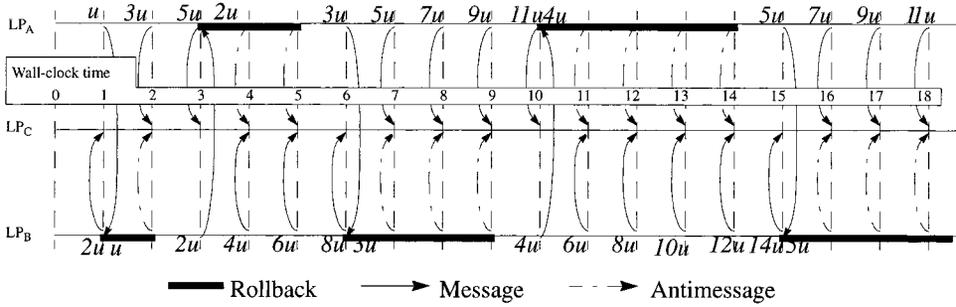


Fig. 6. Echoing in Time Warp.

A detailed proof of this theorem is provided in Srinivasan and Reynolds [1995] by induction on the amount of wall-clock time required to advance logical time from $(n - 1)u$ to nu . The crux of the proof lies in the observation that in the time it takes an LP to roll back $(2t - 1)u$ logical time units (t units of wall-clock time) and then advance by u logical time units (1 unit of wall-clock time) the other LP would have advanced $(t + 1)2u$ units of logical time so that the difference in their logical clocks will now be $(2(t + 1) - 1)u$, thus increasing the depth of rollback ad infinitum.

5.4.1 AAWP Execution. Consider the following NPSI adaptive protocol, ZETA,¹ based on the framework of Section 4.

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

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

where σ_i is LP_i 's logical clock and $MaxEP_i$ is the maximum value of EP_i observed thus far. After executing an event, LP_i recomputes 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. Figure 7 shows the execution of the simulation using the protocol described previously. It is evident from the diagram that the echoing observed under Time Warp (Figure 6) has been avoided since each rollback is only of unit length.

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

In Srinivasan and Reynolds [1995], we prove, again by induction, that the amount of wall-clock time required to advance logical time from

¹As the name suggests, ZETA preceded ETA, the NPSI algorithm described in following sections.

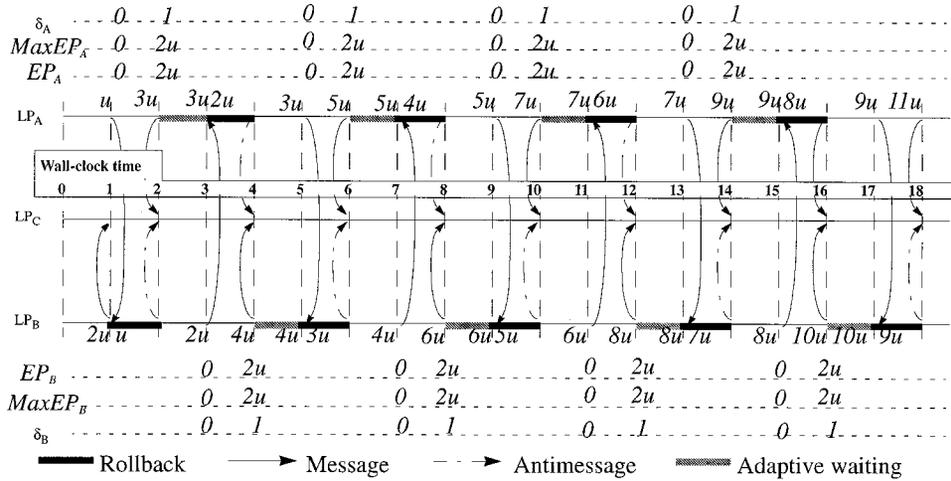


Fig. 7. Avoiding echoing with adaptive aggressiveness.

$(n - 1)u$ to nu in this case is two wall-clock time units. The intuition behind this proof is best understood by observing the base case ($t = 1$) of the Time Warp execution: although LP_B takes $t + 1 = 2$ wall-clock time units to roll back $(2t - 1)u = u$ logical time units and then move ahead by u logical time units, LP_A advances by $2u$ logical time units and then waits for $\delta_i = EP_i/MaxEP_i = 2u/2u = 1$ unit of wall-clock time. Thus, when the next message is sent by LP_B, LP_A is ahead only by u logical time units (compared to $3u$ with Time Warp) so that the next rollback is once again limited to a depth of u logical time units.

We have thus shown that ZETA 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 ratio of completion times is not bounded by a constant for a given simulation, the adaptive protocol outperforms Time Warp arbitrarily.

Although EchoSystem is somewhat contrived, ZETA is not. In Srinivasan and Reynolds [1995b], we present the results of simulating EchoSystem using an implementation of ZETA. In conformance with our analysis here, we have observed that the speedup of ZETA over Time Warp increases with the logical time span of the simulation.

5.5 Analysis Summary

We have established the theoretical potential for NPSI protocols to outperform Time Warp, in terms of completion time, by an arbitrary amount. This is very encouraging. However, the fact that Time Warp can also outperform NPSI protocols (and AAWPs in general) by an arbitrary amount dictates that caution must be exercised. In subsequent sections we explore and analyze experimentally a class of NPSI protocols that, by design, embody the encouraging aspects of the analysis just presented.

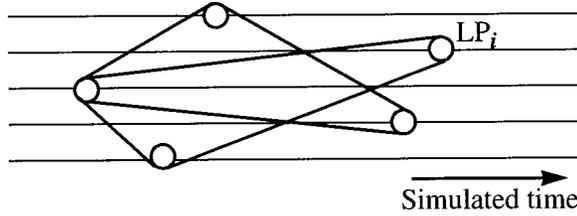


Fig. 8. Elastic Time.

6. ELASTIC TIME ALGORITHM

We present the Elastic Time Algorithm (ETA), an NPSI adaptive protocol based on the framework described in Section 4. Recall that the framework consists of two mappings M_1 and M_2 , and a value called the error potential. M_1 computes an error potential for each LP and M_2 uses it to control the LP's optimism. The following analogy provides an intuitive understanding of ETA: imagine LP_i and its predecessors as pins moving along the logical time line with an elastic band around them, as shown in Figure 8 (each LP has its own elastic band, considered in isolation from other bands). The farther LP_i moves away from its predecessors, the slower its progress due to the restraining pull of the elastic band tying it to its predecessors. When the LP farthest behind moves forward, the restraint on LP_i is reduced so that it may quicken its pace. Thus, the rate at which an LP advances in simulated time is controlled by the tension in the elastic band—hence elastic time. The tension in the elastic band corresponds to LP_i 's error potential.

6.1 M_1 —Computing EP_i

Define the following.

σ_i = logical clock of LP_i ; timestamp of the event either just executed or being executed currently.

v_i = minimum unreceived message time of LP_i (i.e., the smallest among the timestamps of all messages sent by LP_i that have not yet been received and/or incorporated into the events list of the receivers).

η_i = next event time of LP_i , defined as σ_i while LP_i executes an event; otherwise, the smallest among the timestamps of events scheduled to be executed at LP_i .

$\alpha_i = \text{MIN}(\eta_i, v_i)$.

$\alpha'_i = \text{MIN}_{LP_k \in \text{predecessor set of } LP_i}(\alpha_k)$.

M_1 is the function:

$$M_1 : EP_i = \text{MAX}\{\eta_i - \alpha'_i, 0\}.$$

Note, α'_i constitutes the state information obtained dynamically using a feedback system. This mapping is a variation of mapping (3) in Section 4.1: it uses α'_i to compute EP_i rather than GVT. We call α'_i the *minimum future*

time (MFT) since it is the minimum (over the predecessor set) of the next event times and the timestamps of any messages in transit. Thus α'_i can be thought of as representing the time of the next event relevant to LP_i (i.e., the next event that can affect LP_i). Note, v_i is included in computing α'_i since a message in transit may ultimately cause an event to be scheduled at LP_i . Since η_i is LP_i 's next event time, the difference between η_i and α'_i is an indicator of how far ahead (in logical time) LP_i 's next scheduled event is of the next event that can possibly affect LP_i . It is interesting to compare the definitions of target-specific virtual time (TSVT) [Pancerella and Reynolds 1993], the target-specific analogue of GVT, and α'_i :

$$\text{TSVT}_i = \text{MIN}_{LP_k}(\sigma_i, v_i) \forall LP_k \in \text{the predecessor set of } LP_i$$

$$\alpha'_i = \text{MIN}_{LP_k}(\eta_i, v_i) \forall LP_k \in \text{the predecessor set of } LP_i.$$

From the preceding definitions, we see that TSVT_i represents the current state (σ_i) of the simulation, whereas α'_i indicates the immediate future (η_i). Intuition suggests, and our experimental observations support this intuition, that α'_i will prove more effective in controlling optimism than TSVT_i or GVT. We describe an algorithm to maintain σ_i , η_i , and v_i in Srinivasan [1995].

Recall that mapping (3) of Section 4.1 has the drawback that the LP with the next safe event (i.e., the event that is guaranteed not to be rolled back) is unable to determine this fact. Potentially, this leads to arbitrarily long periods of unnecessary waiting. The use of α'_i instead of GVT in ETA solves this problem: if LP_i has the next safe event then $v_j \geq \eta_i$ and $\eta_j \geq \eta_i \forall LP_j$ predecessors of LP_i , which implies $\alpha'_i \geq \eta_i$. Correspondingly, ETA's M_1 sets EP_i to zero, effectively terminating any waiting in which LP_i may be engaged.

6.2 M_2 —Controlling Optimism

Given a value of EP_i computed by M_1 , M_2 is the linear function,

$$M_2: \delta_i = s \cdot EP_i,$$

where s is a scaling factor (discussed in Section 8). The event-processing loop of an LP using ETA (i.e., the event-processing loop of an LP using Time Warp modified to incorporate M_2) is shown in the following.

Event-processing loop of an LP using M_2

1. $\sigma_i = 0$; η_i = timestamp of first scheduled event
2. While simulation is not complete
3. $\sigma_i = \eta_i$; Process next event {schedule future events; send messages as required}
4. Compute new η_i
5. Start wait timer
6. do
7. Receive and buffer messages
8. Exit from loop if
9. there is a message that will cause rollback

10. or if the message has timestamp = $TSVT_i$
11. Process a message (if any) and recompute η_i
12. Update α'_i , EP_i and δ_i
13. Read wait timer
14. while wait timer value $< \delta_i$
15. If necessary, roll back and recompute η_i
16. Process messages and recompute η_i
17. Save state
18. Collect fossils
19. Endwhile

Note, the blocked state of LP_i is implemented as a loop (lines 5 to 14) with the following desirable properties.

- (1) EP_i and δ_i are updated (i.e., M_1 is computed) frequently in the blocked state.
- (2) The blocked state is not opaque in that while in this state, LP_i receives messages from its input channels (line 7) and checks for two kinds of messages (lines 8 to 10): (i) one that may cause it to roll back, and (ii) one with timestamp $TSVT_i$. In each case, an action is warranted immediately—a rollback in the former and processing the message with timestamp $TSVT_i$ (since there can be no messages with timestamps $< TSVT_i$) in the latter—and therefore, the waiting is aborted.
- (3) LPs process messages in the blocked state. This reduces lost opportunity cost in the following situation. Assume an incoming message at LP_i has a timestamp smaller than η_i . Since this message may schedule an event at LP_i , η_i may decrease in processing the message, thus reducing EP_i (which would have been abnormally high if the message had not been processed, resulting in lost opportunity cost). Also, processing messages as soon as possible reduces the time for which they are considered unreceived and thus accelerates the progress of α'_i .
- (4) Waiting is not memoryless. A wait timer is started once upon entry to the blocked state. While LP_i is blocked, η_i is constant and α'_i increases monotonically. Thus, δ_i decreases. LP_i interprets each value of δ_i as an estimate of the amount of time it should have waited since the start of the waiting period. When this value becomes smaller than the time actually waited, LP_i exits the blocked state.

Finally, note this M_2 provides direct control over an LP's aggressiveness only. Risk is controlled only to the extent that while in a blocked state, the LP does not send out messages. Clearly, it is possible (and perhaps desirable) to have a separate mechanism to limit risk.

6.3 Properties of ETA

ETA is adaptive because the aggressiveness and risk of LPs can change dynamically. ETA follows a *microadaptive* approach wherein the adaptiveness is fine-grained: it occurs after each event execution. In contrast, most existing protocols that limit optimism follow a *macroadaptive* approach wherein control settings are fixed for significant periods of time and

changed between periods. A microadaptive approach is inherently more capable of reacting efficiently to rapid changes in system behavior.

ETA is a hybrid of optimistic and conservative approaches, depending on the value of s . When $s = 0$, $\delta_i = 0$ irrespective of the value of EP_i , and each LP executes exactly one iteration of the wait-loop in Section 6.2 in each iteration of the main event execution loop. Thus, with $s = 0$, ETA is essentially identical to Time Warp. As s tends to infinity, δ_i becomes large even for small values of EP_i . However, as noted earlier, the LP (LP_i) with the next safe event has $EP_i = 0$, and consequently, $\delta_i = 0$. LP_i terminates its waiting, executes its next event, and then recomputes η_i . This new value of η_i is subsequently incorporated into the α s of LP_i 's successors so that the LP (LP_j) with the next event that is "almost" safe computes $EP_j = 0$, terminates its waiting, and executes its next event. Although it appears as if events are executed somewhat sequentially, the use of target-specific information to compute the error potential implies multiple LPs may execute events concurrently. In the preceding example, LP_i 's new next event may be safe and thus be executed concurrently with LP_j 's. Furthermore, we have described the operation of ETA under zero-lookahead [Fujimoto 1990] conditions: concurrency may be increased by incorporating lookahead into M_1 . By making the state saving frequency proportional to the rollback frequency, the state saving costs approach zero for very large values of s and, overall, ETA behaves similarly to a conservative protocol. ETA is similar to window-based protocols with two significant differences:

- it is completely asynchronous: there are no barrier synchronizations to negotiate windows;
- each LP's logical time window may be considered infinite but the event execution rate decreases as the LP moves farther away from the base of the window.

7. PERFORMANCE ANALYSIS

We describe a three-part performance study of ETA. The first part was conducted using an implementation of ETA on a prototype four-processor reduction network to provide NPSI, the second part focuses on demonstrating scalability using a simulation of systems with larger numbers of processors, and the third studies ETA on a larger number of processors simulating a real application.

7.1 Performance Criteria

From Section 3, a good measure of success for protocols that control optimism is how significantly they reduce R , the residual total cost at the balance point in Figure 1. Recall the cost function:

$$\begin{aligned} \text{total cost} = & \text{rollback cost} + \text{memory management cost} \\ & + \text{lost opportunity cost.} \end{aligned}$$

Memory management schemes are required in Time Warp because LPs consume excessive amounts of memory due to uncontrolled optimism. As demonstrated by results presented here, controlling optimism significantly reduces memory consumption so that the need for memory management schemes could be eliminated (except in certain special topologies involving fast source LPs). Consequently, we focus on rollback and lost opportunity costs; our protocols do not incorporate memory management. Thus, the cost function reduces to:

$$\textit{total cost} = \textit{rollback cost} + \textit{lost opportunity cost}.$$

A second measure of protocols that control optimism is their performance relative to protocols that do not control optimism. Time Warp is our reference protocol. ETA outperforms Time Warp over a wide range of workloads that includes those for which Time Warp performs well and those for which it does not.

7.2 Testing Environment

The first part of the performance study was conducted by implementing ETA on a cluster of four SPARC-2 workstations connected by Ethernet and a prototype parallel reduction network (PRN) designed and built in-house [Reynolds et al. 1993]. Each workstation communicates with its own auxiliary processor (AP) through dual-ported memory (DPRAM). The four APs are connected to the PRN, which computes and disseminates globally reduced values at very high speeds. The APs are dedicated processors intended to offload the task of sending values into the PRN and reading its global output from the workstations so as to keep interference with the simulation to a minimum. The workstations communicate relevant state changes to the APs through the DPRAM. The APs process this information and forward it to the PRN. Also, the APs read the globally reduced output of the PRN and propagate relevant parts of it to the workstations. Since the PRN produces globally reduced values, target-specific NPSI (Section 3) is emulated by software over the PRN.

7.3 Test Cases

A protocol should be tested extensively. Unfortunately, a standard set of performance benchmarks for PDES protocols does not exist. To facilitate broad testing of ETA on performance-critical properties (e.g., event execution times, message destination probabilities, load distribution, and message latencies), we developed a synthetic workload generator similar to the PHOLD model [Das and Fujimoto 1994]. Our generator supports *message-initiating* workloads (closed systems), *self-initiating* workloads (open systems), and combinations of the two. Configurable parameters include event execution time, average timestamp increment, state saving cost, communication topology and distributions, number of local events per message, and state saving and fossil collection frequencies. Time-consuming application-

Table I. Workloads

Name	Top	Msg/Self	Initial Pop.	Event Delay	Comment
MSG-BIAS	T_2	Msg	25	100 μs	Similar to a biased queueing network.
LGRAIN	T_1	Msg	25	1 ms	Represents workload where Time Warp is expected to perform well.
ECHO	T_3	Self + Msg	1 msg. @ LP_0	100 μs	Implementation of echoing example described in Lubachevsky et al. [1989]. LP_0 and LP_1 execute in self-initiating mode. After processing message-initiating event, LP_0 sends message to LP_1 . After processing message, LP_1 sends it back to LP_0 and cycle repeats. The causal message exchanged between LP_0 and LP_1 rolls back its receiver and while receiver is rolling back, sender simulates ahead, sending erroneous messages to LP_2 . Thus, LP_0 and LP_1 spend increasing amounts of time rolling back and progress of simulation diminishes rapidly. This workload is inherently unstable.
PHASES-MSG	T_1	Msg	25	Slow: 1 ms Fast: 100 μs	LPs undergo alternating phases of slow and fast execution. Slow phase consists of 10 events, with mean delay of 1 ms, and a fast phase consists of 100 events, with mean delay 100 μs .

specific actions such as event executions and state saving are simulated by busy loops; all other mechanisms such as rollback, restoring state, sending antimessages, and fossil collection and data structures such as saved state list and antimessage list are implemented in detail. Table I describes a suite of workloads for which performance results are presented in Section 7.5. The suite is a representative subset of the workloads for which performance was studied in Srinivasan [1995]. Figure 9 shows topologies used in the workloads. A torus was used because it appears often in the PDES literature and it provides sufficient connectivity to introduce rollback costs. A number associated with an arc is the probability that a generated message is sent along that

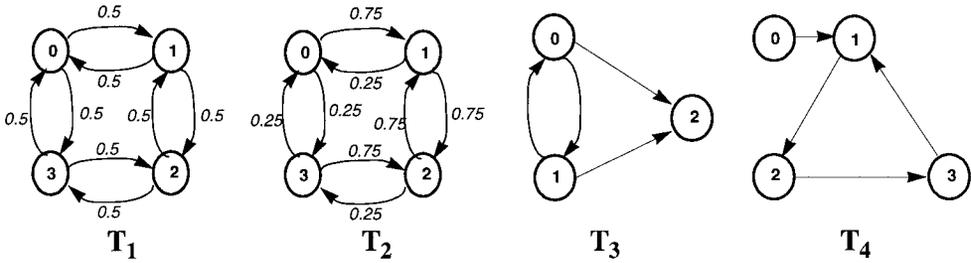


Fig. 9. Topologies for workloads.

arc. For all workloads, the mean state saving time is $25 \mu\text{s}$ and state saving and fossil collection are performed after each event.

7.4 Metrics

Recall the cost function of Section 7.1:

$$\text{total cost} = \text{rollback cost} + \text{lost opportunity cost}.$$

Rollback cost can be measured directly by the time an LP spends rolling back, including state restoration and sending of antimessages. Reduction in rollback cost (time) has the effect of reducing the completion time of the simulation, which reflects the reduction in total cost. On the other hand, lost opportunity cost introduced by limiting optimism tends to increase completion time. Thus, completion time, in conjunction with rollback time, can be used to obtain some indications about lost opportunity cost. Other metrics of rollback behavior are number of rollbacks and average depth of rollback (average number of events rolled back).

Memory in aggressive systems can be of three types:

- processed and committed: this is managed by fossil collection;
- processed and uncommitted: this is created due to uncontrolled aggressive processing and therefore can be managed by limiting optimism;
- unprocessed: this is created by *runaway LPs* (LPs consuming other LPs' message memory, unconstrained) and can be managed by a memory-based flow control mechanism.

Controlling optimism is expected to reduce these last two significantly. Processed and uncommitted memory is measured by the maximum and average sizes of the saved-state list (in terms of the number of entries in the state list). Since ETA does not incorporate memory-based flow control currently, we do not measure unprocessed memory.

The scaling factor s in M_2 translates the value of EP_i from logical time units to a delay in real-time. The range of EP_i is dependent on the logical time increments exhibited by the LPs and the rate at which they execute events, send messages, and so on. Since these factors differ considerably across applications, it is expected that the value of s that maximizes performance will differ across applications. Thus, s is the independent

variable in our performance tests. In the graphs that follow, s is called “slope” for historical reasons.

7.5 Performance Results

For each of the workloads, we present performance results using the metrics previously described. Excluding completion time, all metrics are for a single LP. In many cases, the LPs were indistinguishable and the choice was arbitrary. In one case (MSG-BIAS), we present the results for two LPs representing the two sets of LPs on either side of the bias. In other cases, we chose an LP that was most relevant based primarily on the topology and the nature of the workload. Recall ETA is essentially identical to Time Warp when $s = 0$. In graphs that use a logarithmic scale for s , the s -coordinate of the origin is artificially set to a nonzero value, since a logarithmic scale reaches zero only at $-\infty$.

7.5.1 MSG-BIAS. Figure 10 shows the performance of ETA with MSG-BIAS. The absence of the characteristic parabolic shape in the completion time curve is attributed to a lack of sufficient concurrency in this workload. This is primarily because: (i) a message is sent after each event (since the event grain is small ($100 \mu\text{s}$) and the communication latency is high (1–2 ms), the computation-to-communication ratio is too small to support concurrency); and (ii) there are only four processors. Recall that for large values of s , ETA behaves similarly to a conservative protocol. Since the communication graph is strongly connected and there is no lookahead, a conservative protocol will effectively execute this simulation sequentially (albeit with messaging latencies and other overheads). Correspondingly, we observe that the completion time does not increase beyond the minimum. ETA is able to eliminate nearly all of the rollback cost. Where completion time is minimized ($s = 50$), the rollback time is nearly zero and the decrease in completion time (~ 34 sec) is greater than the decrease in rollback time (~ 25 sec). The effect of the bias is apparent in the rollback behaviors of LP_0 and LP_1 as seen in parts (c) and (d) of the figure. At $s = 20$ the number of rollbacks is reduced by about 60% and the average rollback depth is close to 1, the minimum depth. Thus, most of the benefit of ETA appears to be from reducing the rollback depth rather than the number of rollbacks. This is as expected since completely eliminating rollbacks requires either conservative processing (ETA approaches this at $s = 1,000$) or perfect guessing (which is not in ETA’s design). Furthermore, these results support the intuition that the benefits from eliminating rollbacks completely (as in a conservative approach) are far outweighed by the resulting lost opportunity cost. Part (e) of the figure plots the average amount of time an LP waits after each event. Since the bias in the workload was chosen arbitrarily and ETA was not tailored in any way to compensate for it, this graph shows that ETA adapts to the load imbalance by throttling LP_0 (and LP_3) more than LP_1 (and LP_2), and thus demonstrates the adaptive nature of ETA. In Srinivasan [1995] we describe another biased workload similar to MSG-BIAS, called ASYM, where the

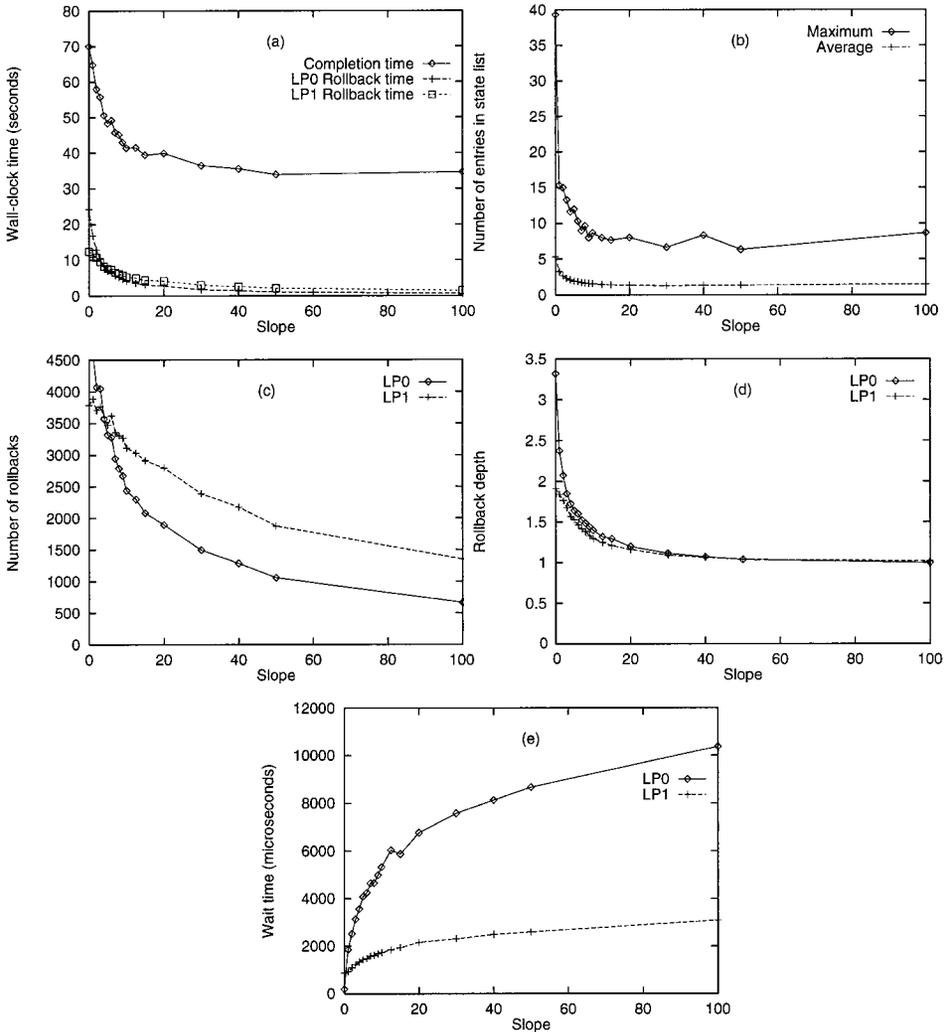


Fig. 10. Performance for MSG-BIAS.

bias is caused by different event execution rates rather than messaging probabilities. The results for ASYM are similar to those for MSG-BIAS.

7.5.2 LGRAIN. Figure 11 shows the performance of the LGRAIN workload. Since the event grains are fairly large and the mean number of local events between messages is five, we observe that rollback costs are small. The completion time exhibits the expected parabolic shape due to the tradeoff shown in Figure 1 (Section 3). Two points to note: rollback time is nearly zero at the point where completion time is minimized ($s = 5$); and the maximum reduction in completion time (~ 10 sec at $s = 5$) is greater than the reduction in rollback time (at $s = 5$). This implies that ETA is able to eliminate most of the rollback cost before any appreciable lost

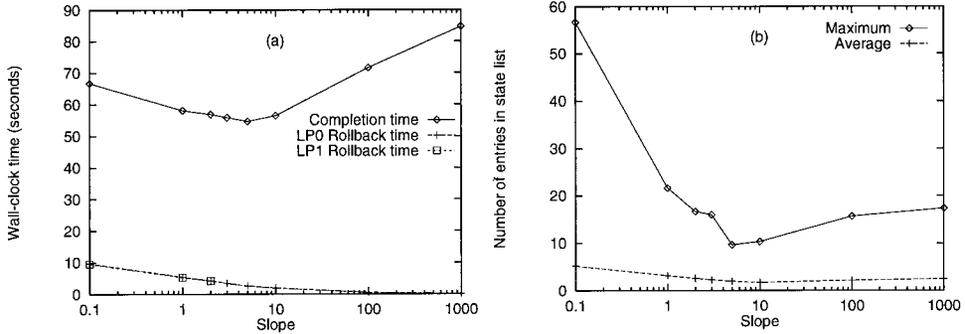


Fig. 11. Performance for LGRAIN.

opportunity cost is introduced. Furthermore, since rollback cost is nearly zero, the performance of ETA is close to the best possible performance for protocols with controlled optimism, as defined in Section 7.1. The reduction in maximum state list size is significant. As s increases beyond the point at which completion time is minimized ($s = 5$), we see that the maximum state list size tends to increase. This is because the rate of progress of GVT is decreased in this region due to high lost opportunity cost. This implies that state list entries are converted into fossils at a slower rate, resulting in longer state lists. Reduction in rollback costs for LGRAIN was similar to that for MSG-BIAS and thus is not depicted.

7.5.3 ECHO. Figure 12 shows the performance of ETA with ECHO. ETA produces substantial reductions in completion time and rollback time. Recall that ECHO is inherently unstable in that the amplitude of rollbacks increases without bound, reducing the efficiency of the simulation to zero asymptotically (this was observed experimentally [Srinivasan 1995]). By limiting optimism, ETA eliminates this instability (shown theoretically for the NPSI algorithm ZETA in Section 5.3). The absence of the characteristic parabolic shape of the completion time curve is due to the inherently sequential nature of the workload—the only significant activity in the simulation is the exchanging of the single message between LP_0 and LP_1 . ETA produces correspondingly significant reductions in memory consumption as well. Memory consumption is high in ECHO because although one of LP_0 and LP_1 rolls back, the other simulates aggressively far into the logical future. Limiting optimism reduces this memory consumption.

7.5.4 PHASES-MSG. Figure 13 shows the performance of ETA with PHASES-MSG. Due to the large event grains in the slow phases of the computation, there is some concurrency in this workload, as seen by the slightly parabolic shape of the completion time curve. As with previous workloads, the rollback time is nearly zero at the point where completion time is minimized, and the reduction in completion time (~ 15 sec) exceeds the reduction in rollback time (~ 12 sec). Figure 14 shows the average rollback depth as the simulation proceeds, for both Time Warp (part (a)) and ETA (part (b)). From this figure, the adaptive nature of ETA is clear in

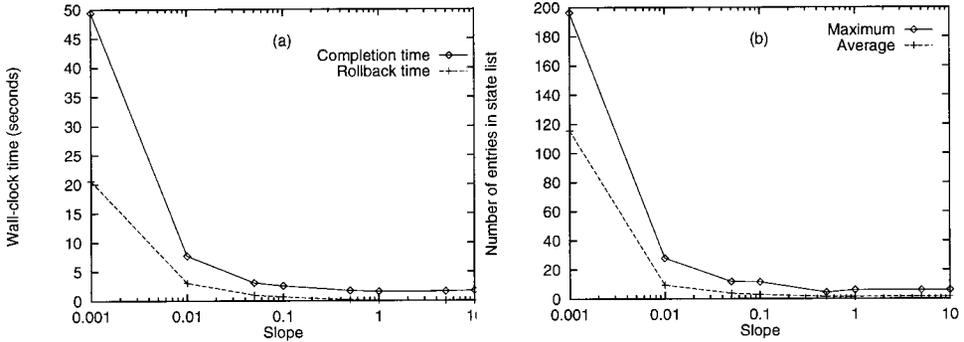


Fig. 12. Performance for ECHO.

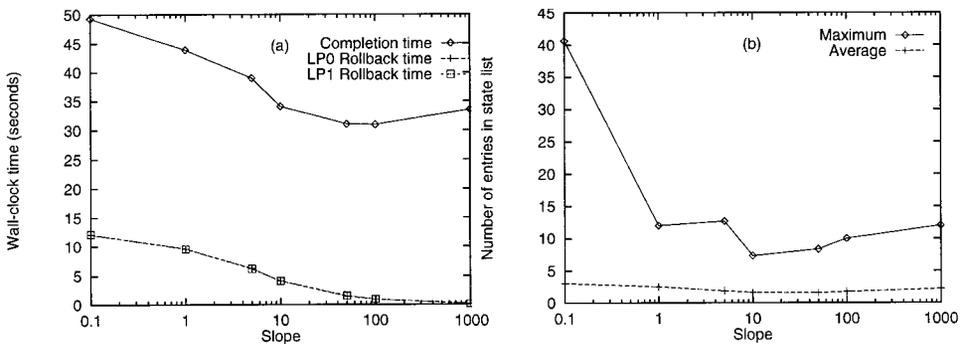


Fig. 13. Performance for PHASES-MSG.

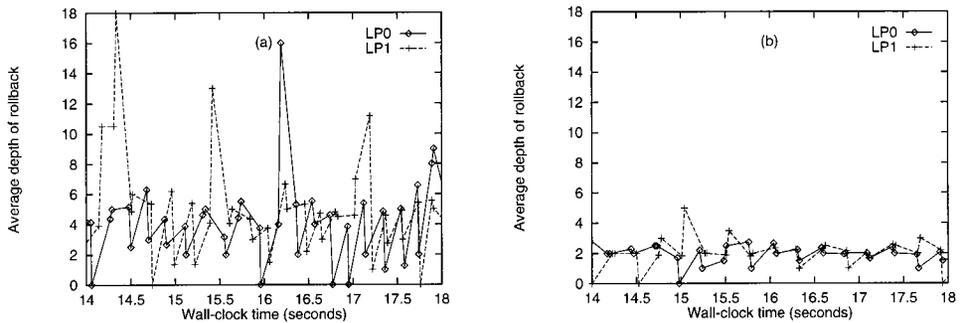


Fig. 14. Adaptive response of ETA to phase behavior.

that it smooths out the phase behavior by throttling the appropriate LPs in each phase.

7.6 Scalability

We conducted a simulation study to obtain some indication regarding the scalability of ETA (in general, the NPSI approach). The study focused on determining whether the trends observed in the experiments just described continued to exist as both the problem size and the number of processors

were scaled up. The specific trends of interest are: the completion time curve has a parabolic shape as a function of s , especially since larger systems will have more concurrency; at the point of minimum completion time, the reduction in completion time (relative to the Time Warp execution) equals or exceeds the reduction in rollback time; and rollback time is very low (relative to the completion time) at the point of minimum completion time.

The simulation model, developed using the SES/Workbench simulation tool, consists of detailed models of the hardware and software components of the testing environment used in the previous experimental study (see Section 7.2). Hardware components include the workstations, the auxiliary processors (AP), the host communication network and a target-specific reduction network that implements the NPSI calculator. Software components are modeled as programs executing on processors, with delays associated with segments of code to model execution times. The LPs execute on the SPARC-2 workstations, and the synchronization algorithms execute on the APs, which are custom-designed single-board computers using the M68020 CPU. The execution time delay for a segment of code is determined by the product of the number of instructions to execute that segment, estimated from a compiler's assembly output, and the average instruction cycle time of the workstation, derived from the technical specifications of the computer in the case of the workstations and in the case of the auxiliary processors, from detailed timing information obtained through low-level instrumentation of the auxiliary processor boards with a logic analyzer. The host communication network is modeled as a delay with mutually exclusive access. Thus, the transmission delay is proportional to the number of LPs attempting to transmit concurrently. Since the focus of this study was on the utility of NPSI to ETA rather than the efficient computation of NPSI itself, we used a simple design for the target-specific reduction network model that uses $O(n^2)$ hardware to achieve NPSI computation in $O(\log n)$ time [Pancerella and Reynolds 1993].

With regard to validation of the simulation model, several factors contributed to a high level of confidence in the face validity of the model: (i) the software component models used actual code from the implementation of ETA on the four-processor prototype; (ii) input parameters for the target-specific reduction network model were obtained by profiling the four-processor network using a logic analyzer; and (iii) very detailed simulation traces were produced and analyzed to confirm that the model behaved as expected. The model was also validated by comparing its output with that of the four-processor implementation on three workloads (MSG-BIAS, SELF-BIAS, and ECHO). In each case, a number of statistics such as the number of events simulated, number of messages sent, and percentage of events committed were matched for trends as slope was varied. After validation, the input parameters for the target-specific reduction network model were scaled to values projected for a production version of such a network (our implementation was a prototype and thus considerably slower than a production version). Due to excessive running times (up to six hours

Table II. Trends Demonstrating the Occurrence of the Chasing Phenomenon

LPs	<u>Number of Rollbacks</u> Total Number of Events	Avg. Depth of Rollback
4	0.79	1.8
8	0.5	3.2
16	0.18	11

to simulate one second), the simulation runs were terminated after simulating approximately twice the amount of time to reach steady state, where steady state was defined as a variation of less than 5% over three consecutive samples of the rate of advance of GVT.

The scalability tests were performed using scaled-up versions of a representative set of workloads from the previous experiments. This set included a biased variant of SELF-SYM, MSG-BIAS, ECHO, and two new workloads called CHASE and LOGIC-NW. We increased both the number of LPs in the workload and the number of processors, with one LP executing on each processor. In all cases, it was observed that the trends listed in the preceding continued up to the largest number of LPs tested in each case (64, 64, 16, 16, and 21, respectively) with no apparent decline. We present the results for CHASE and LOGIC-NW only.

7.6.1 CHASE. CHASE represents another kind of adverse rollback behavior in which erroneous messages outpace corresponding antimes- sages. The topology consists of LP_1 through LP_n in a ring and LP_0 as a slow source of external messages to LP_1 (T_4 in Figure 9). The mean event execution times are 10 ms for LP_0 and 100 μ s for the others. The erroneous computation proceeds in the form of the five events circulating among the $n - 1$ LPs. A message from LP_0 causes LP_1 to roll back and start a chain of antimes- sages that chases the erroneous computation around the cycle. This chasing phenomenon will persist only if the logical distance (i.e., the number of LPs) between an erroneous message M and the corresponding antimes- sage $-M$ is sufficiently large. As the separation increases (i.e., for larger systems), chasing will cause fewer and deeper rollbacks, as evi- denced by the measured values in Table II. Figure 15 shows that ETA significantly reduces the effects of the chasing phenomenon (results for 8 LPs are similar).

7.6.2 LOGIC-NW. For LOGIC-NW, we used a topology that is not strongly connected (as compared to a torus). Such topologies are typical in logic networks, which consist of sources of signals and nets that define the flow of signals. Sequential elements in the simulated circuit introduce feedback in the topology. Alternatively, this workload can represent a shop-floor with multiple assembly line systems. The performance of ETA with LOGIC-NW (Figure 16) conforms to general trends observed with other workloads.

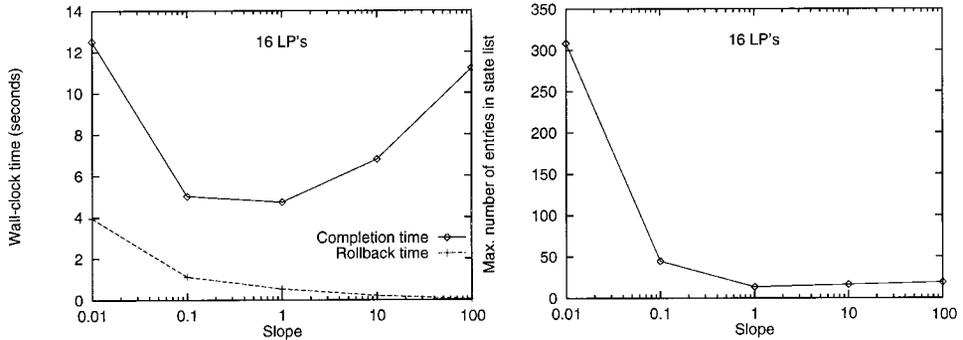


Fig. 15. Scalability results for ETA with CHASE.

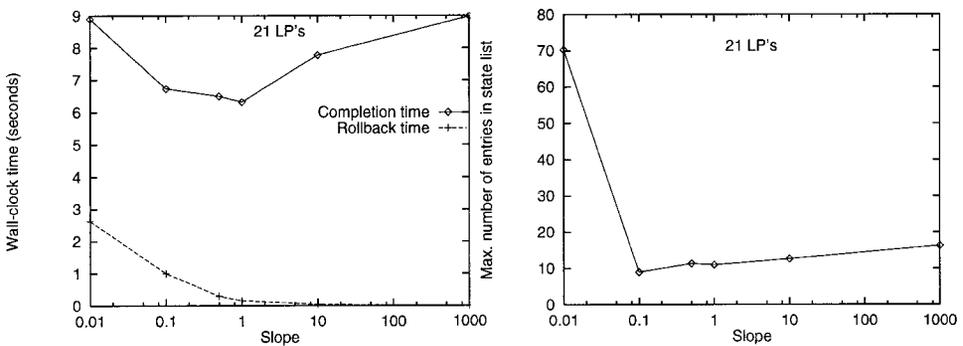


Fig. 16. Scalability results for ETA with LOGIC-NW.

7.7 Power Distribution Grid Application

Finally, we present the performance of ETA with an application model of a large distributed electric power distribution grid. This model was used to study the performance of the MIMDIX environment for parallel simulation in Madisetti et al. [1993]. The simulated system is a power distribution grid consisting of power plants distributed across a wide geographic area. The power plants engage in either local activity or load sharing. In the load-sharing mode, a power plant attempts to satisfy excess load requirements by requesting additional power capacity from neighboring plants. In the model, local activity is simulated by a delay and the load-sharing activity by the exchange of timestamped messages. A process receiving a message either propagates it to another process or consumes it, based on a probability matrix. A self-propagating source process injects messages into this system to ensure that the processes have work to do.

We studied the performance of ETA using the simulation environment described in Section 7.6 by simulating the execution of the power grid application (PGRID) under five test cases presented in Madisetti et al. [1993]. In each of these cases, the system consisted of 10 power plants, each simulated by one LP executing on its own processor. One LP simulated the source and another simulated a sink that consumed all messages sent to it. The remaining eight worker LPs were divided into two sets: set A with two

Table III. PGRID Instances

Case	$[p_{AA}, p_{AB}, p_{BA}, p_{BB}]$	Timestamp Increment	Mean Event Delay (ms)
I-0.2	[0.2, 0.2, 0.2, 0.2]	Uniform (0.1, 100)	1.5
I-0.4	[0.4, 0.4, 0.4, 0.4]	Uniform (0.1, 100)	1.5
II-left	[0.1, 0.6, 0.6, 0.1]	Uniform (0.1, 100)	1.5
II-right	[0.6, 0.1, 0.1, 0.6]	Uniform (0.1, 100)	1.5
III	[0.2, 0.6, 0.6, 0.2]	A: Uniform (100, 200) B: Uniform (900, 1000)	1.5

LPs and set B with six LPs. The source could send messages to any of the eight workers; similarly, each worker could send a message to another worker. The five workloads are described in the Table III. Note that Case III includes two kinds of imbalance: one due to asymmetric messaging probabilities and the other due to asymmetric timestamp increments between the two sets of LPs. The event delays were distributed in a narrow band (10%) around the means indicated. The states of the random number streams at each LP were saved and restored appropriately to ensure repeatable executions with a given set of initial seeds. We simulated up to 50 seconds of execution of the PGRID application.

Overall, the performance of ETA with PGRID is consistent with the performance observed in previous experiments, with ETA consistently outperforming Time Warp (Figure 17). Specifically, rollback time is completely eliminated at the balance point of s , and the overall reduction in completion time exceeds the reduction in rollback time. For values of s greater than the balance point, completion time increases due to lost opportunity cost. Other relevant indicators of ETA's performance described in the case of the MSG-BIAS workload (Section 7.5.1) were also collected and similar behavior was observed. These results are omitted due to space constraints.

7.8 Summary of Results

In summary, we draw the following conclusions about ETA's performance.

- ETA consistently outperforms Time Warp in both completion time and memory consumption.
- At the point where completion time is minimized, rollback time is very low (relative to the completion time). Furthermore, at this point, the reduction in completion time exceeds the reduction in rollback time. This implies ETA eliminates nearly all of the rollback costs in the application without introducing significant lost opportunity cost. Thus, the performance of ETA is close to the best possible by controlling optimism (Section 7.1).
- ETA produces significant reductions in maximum memory consumption. This decreases the dependence of ETA on memory management schemes

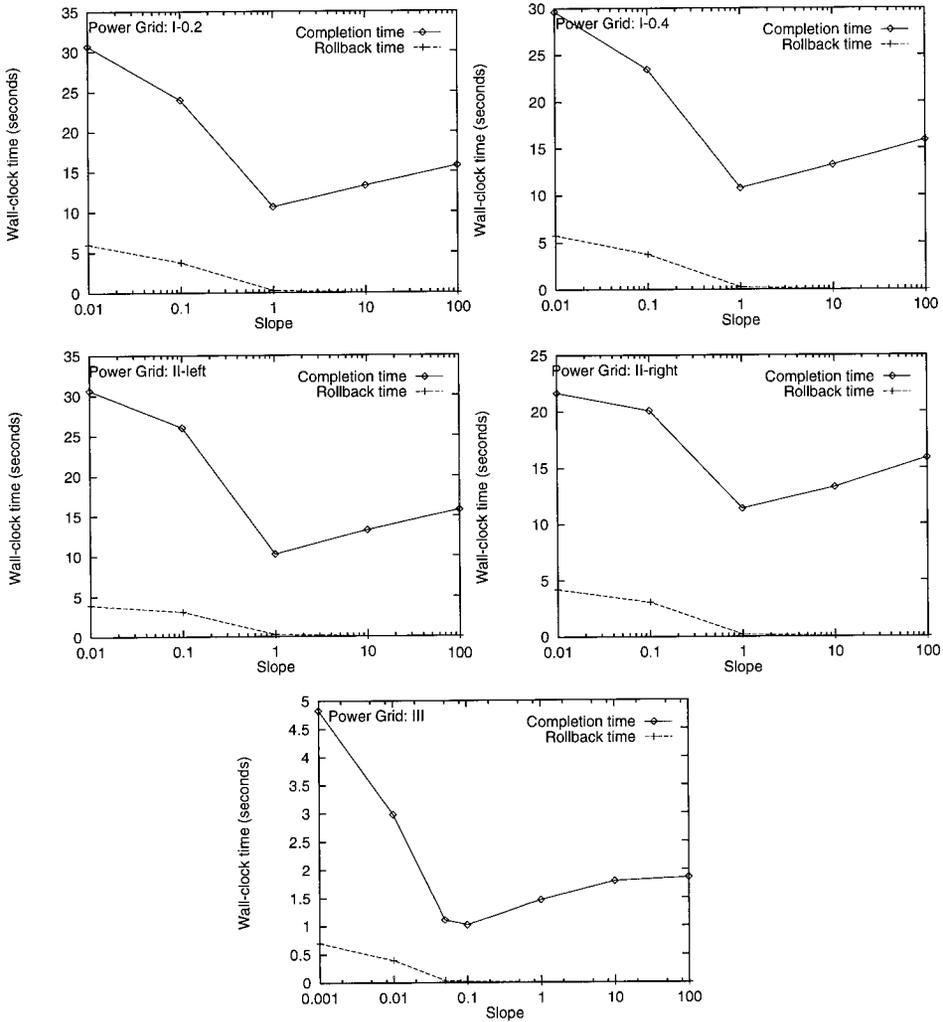


Fig. 17. Performance for PGRID.

such as Cancelback or Artificial Rollback [Lin 1992b]. By incorporating memory-based flow control into ETA's M_1 , we believe the need for such costly memory management schemes can be eliminated altogether.

- ETA is adaptive, even for a fixed value of s . This is demonstrated clearly by the performance of ETA on the biased workloads as well as the PHASES workloads. Since ETA was not tailored in any way to compensate for the bias or the phase behavior (i.e., the same algorithm was used in all experiments), we have shown that ETA adapts to changes in load locality.
- The performance of ETA scales well over moderately parallel systems. We have studied the performance of ETA for systems containing 8, 9, 10, 16, 21, 36, and 64 LPs and processors. The lack of any declining trends in

Table IV. Efficiency of Automatic Tuning Scheme

Workload	Time Warp	Automatic	Hand-Tuned	η (%)
LGRAIN	1047	901.5	882	88.2
MSG-BIAS	843.67	419.75	394.67	94.4
ASYM	614.33	424.67	397	87.3
ECHO	—	120.33	102.67	—
CHASE	810	671.67	663.33	94.3
PHASES-MSG	557	397.33	373.33	86.9

performance is a clear indication that ETA's performance will continue to scale. A comprehensive scalability study over larger systems and topologies other than those considered here is a topic for future work.

8. DETERMINING s

One of the conclusions drawn from the results presented in Section 7 is that the scaling factor s must be chosen properly for ETA to perform well. Too small a value will not reduce rollback costs sufficiently whereas too large a value will introduce lost opportunity cost. The value of s that minimizes the completion time depends on the application and hence cannot be determined a priori in general. The user may be able to provide some hints about this value; on the other hand, there is little application-specific theory to aid the user. We refer to such parameters whose values must be determined in an application-specific manner or adjusted dynamically to ensure good performance, as *tunable parameters*. Although s is an artifact of the particular M_2 chosen for ETA, such tunable parameters will exist in any adaptive protocol. Accordingly, we observe that all existing adaptive protocols admit tunable parameters of some sort (the time-window bound in Turner and Xu [1992], N_1 and N_2 in Steinman [1993], cluster size in Rajaei et al. [1993], blocking window size in Ball and Hoyt [1990], the scaling factor c_i in Hamnes and Tripathi [1994], M_f and M_p in Das and Fujimoto [1994], and degrees of optimism and risk in Arvind and Smart [1992]).

In Srinivasan [1995], we present a simple algorithm that automatically tunes s as the simulation proceeds. The algorithm is based on a characterization of parameters along two dimensions: static versus dynamic and global versus local. We have argued that s is a static global parameter (the adaptiveness of ETA comes primarily from the error potential) which makes the task of automatically tuning its value easier than for other parameters. The basic principle of the algorithm is to control the value of s based on the monitored state of the system. For this purpose, we have proposed and employed a global metric R_C which is the total rate of commitment of events over all LPs. R_C is computed easily using the reduction model of Section 3.

Table IV captures the performance of the algorithm on the workloads described in Section 7. For each workload, the table lists time (in seconds) to complete the simulation using Time Warp, completion time using ETA

with the tuning algorithm, smallest completion time using ETA by manually tuning s and efficiency of the tuning algorithm relative to the manually tuned version. Efficiency (η) is computed as

$$\eta = \frac{TW - Automatic}{TW - Handtuned} \times 100.$$

In the case of ECHO, we were unable to obtain data for Time Warp because the inherently unstable nature of the workload caused system buffers to overflow during long runs. Nevertheless, the automatic scheme was able to eliminate the instability and performed nearly as well as the manually tuned version.

9. CONCLUSIONS

We concur with Reynolds [1993] that a consistently efficient protocol is still one of the most important goals for PDES researchers. As a significant step towards this goal, we have introduced and developed the theory for a new class of synchronization protocols for PDES—NPSI adaptive synchronization protocols. The NPSI class is characterized by the use of low-cost near-perfect state information to adaptively control the optimism of processes in a PDES. We have proposed a framework for the design of NPSI protocols that, using two mappings M_1 and M_2 , isolates the two phases in the design of NPSI protocols: identifying relevant state information and controlling optimism using this information. Based on this framework, we have described the Elastic Time Algorithm. A detailed performance study of ETA conducted using a suite of test workloads with widely different characteristics, shows that ETA consistently outperforms Time Warp, in both completion time and memory consumption. Furthermore, our results suggest that the observed reduction in completion time produced by ETA is close to the best possible for the optimism-controlling class of protocols. Finally, we have presented an analysis that demonstrates that Time Warp and a general class of adaptive protocols we call asynchronous adaptive waiting protocols can outperform each other arbitrarily.

PDES synchronization requirements are irregular, highly data-dependent and dynamic. We believe the direct approach of NPSI protocols, which assume the availability of critical state information, can make good adaptive decisions and thus satisfy the synchronization requirements of PDES. We have demonstrated that the NPSI adaptive approach does indeed hold significant potential to be consistently efficient.

It may be argued that one of the strengths of our approach, namely, the assumption of availability of low-cost near-perfect state information, is also its weakness. We justify the NPSI assumption as follows.

- (1) Protocols that use state information (state-based protocols) have significant potential to be consistently efficient. Many kinds of state information can be utilized effectively. Thus, state-based protocols warrant a thorough study. Recent activity in this area of research is encouraging.

- (2) We do not assume the availability of large amounts of detailed, nonlocal state information (at each LP); clearly, such a model would be difficult to realize in practice. As with adaptive load sharing in distributed systems [Eager et al. 1986], easily computed and disseminated information can produce significant benefits for PDES. Therefore, we assume a simple reduction model that captures relevant state information at each LP in a small set of values and provides each LP with a consistent snapshot of these values in a reduced form (i.e., as minima, sums, logical ORs, etc.).
- (3) The reduction model we have assumed is very general: it makes no PDES-specific assumptions. Nonlocal information, even in reduced form, can benefit most parallel computations. It can be used to implement common synchronization constructs such as barriers. Another area includes isotach concurrency control for parallel systems [Reynolds et al. 1997]. The usefulness of reductions is apparent in the fact that many state-of-the-art high performance architectures support reductions either in hardware (CM-5) or in software (T3D, Intel).
- (4) The reduction model can be realized. It can be implemented directly in shared memory or using a high-speed asynchronous reduction network. Recently, we completed an efficient implementation of the model on a cluster of workstations connected by a high-speed Myrinet switch.

The results reported here represent only a starting point into the exploration of the very promising NPSI adaptive protocols.

REFERENCES

- ARVIND, D. K. AND SMART, C. R. 1992. Hierarchical parallel discrete event simulation in Composite Elsa. In *Proceedings of the Sixth Workshop on Parallel and Distributed Simulation* (Jan.), 147–155.
- BALL, D. AND HOYT, S. 1990. The adaptive Time-Warp concurrency control algorithm. In *Proceedings of the SCS Multiconference on Distributed Simulation* (Jan.), 174–177.
- BELLENOT, S. 1993. Performance of a riskfree Time Warp operating system. In *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation* (May), 155–158.
- DAS, S. R. 1996. Adaptive protocols for parallel discrete-event simulation. In *Proceedings of the 1996 Winter Simulation Conference* (Dec.), 186–193.
- DAS, S. AND FUJIMOTO, R. M. 1994. An adaptive memory management protocol for Time Warp parallel simulation. In *Proceedings of SIGMETRICS '94* (May), 201–210.
- DICKENS, P. M. 1993. Analysis of an aggressive global windowing algorithm. Ph.D. Thesis, Department of Computer Science, University of Virginia (May).
- DICKENS, P. M. AND REYNOLDS, P. F., JR. 1990. SRADS with local rollback. In *Proceedings of the 1990 SCS Multiconference on Distributed Simulation* (Jan.), 161–164.
- EAGER, D. L., LAZOWSKA, E. D., AND ZAHORJAN, J. 1986. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Softw. Eng. SE-12* (May), 662–675.
- FERSCHA, A. AND TRIPATHI, S. K. 1994. Parallel and distributed simulation of discrete event systems. *Tech. Rep. CS-TR-3336*, University of Maryland at College Park (August).
- FUJIMOTO, R. M. 1990. Parallel discrete event simulation. *Commun. ACM* 33, 10 (Oct.), 30–53.
- FUJIMOTO, R. M. 1993. Parallel discrete-event simulation: Will the field survive? *ORSA J. Comput.* 5, 3 (Summer), 213–230.
- FUJIMOTO, R. M. AND HYBINETTE, M. 1997. Computing global virtual time in shared-memory multiprocessors. *ACM Trans. Model. Comput. Simul.* 7, 4 (Oct.).

- GIMARC, R. L. 1989. Distributed simulation using hierarchical rollback. In *Proceedings of the 1989 Winter Simulation Conference*, 621–629.
- HAMNES, D. O. AND TRIPATHI, A. 1994. Evaluation of a local adaptive protocol for distributed discrete event simulation. In *Proceedings of the 1994 International Conference on Parallel Processing* (August), Vol. III, 127–134.
- JEFFERSON, D. R. 1985. Virtual time. *ACM Trans. Program. Lang. Syst.* 7, 3 (July), 404–425.
- LIN, J.-J. 1992. Efficient parallel simulation for designing multiprocessor systems. Ph.D. Thesis, University of Michigan.
- LIN, Y.-B. 1992. Memory management algorithms for optimistic parallel simulation. In *Proceedings of the Sixth Workshop on Parallel and Distributed Simulation* (Jan.), 43–52.
- LIPTON, R. J. AND MIZELL, D. W. 1990. Time Warp vs. Chandy-Misra: A worst-case comparison. In *Proceedings of the 1990 SCS Multiconference on Distributed Simulation* (Jan.), 137–143.
- LUBACHEVSKY, B., WEISS, A., AND SHWARTZ, A. 1989. Rollback sometimes works . . . if filtered. In *Proceedings of the 1989 Winter Simulation Conference* (Dec.), 630–639.
- LUBACHEVSKY, B., WEISS, A., AND SHWARTZ, A. 1991. An analysis of rollback-based simulation. *ACM Trans. Model. Comput. Simul.* 1, 2 (April), 154–193.
- MADISETTI, V. K. 1993. Randomized algorithms for self-synchronization. Private communication.
- MADISETTI, V. K., HARDAKER, D. A., AND FUJIMOTO, R. M. 1993. The MIMDIX environment for parallel simulation. *J. Parallel Distrib. Comput.* 18, 473–483.
- MADISETTI, V. K., WALRAND, J., AND MESSERSCHMITT, D. 1988. WOLF: A rollback algorithm for optimistic distributed simulation systems. In *Proceedings of the 1988 Winter Simulation Conferences* (Dec.), 296–305.
- MCATTER, J. 1990. A unified distribution simulation system. In *Proceedings of the 1990 Winter Simulation Conference*, 415–422.
- MEHL, H. 1991. Speed-up of conservative distributed discrete event simulation methods by speculative computing. In *Proceedings of the Fifth Workshop on Parallel and Distributed Simulation* (Jan.), 163–166.
- NICOL, D. M. AND HEIDELBERGER, P. 1995. On extending parallelism to serial simulators. In *Proceedings of the Ninth Workshop on Parallel and Distributed Simulation* (June), 60–67.
- NICOL, D. M. AND REYNOLDS, P. F., JR. 1990. Optimal dynamic remapping of parallel computations. *IEEE Trans. Comput.* 39, 2 (Feb.), 206–219.
- ORSA 1993. *ORSA J. Comput.* 5, 3 (Summer), 213–248.
- PANCERELLA, C. M. AND REYNOLDS, P. F., JR. 1993. Disseminating critical target-specific synchronization information in parallel discrete event simulations. In *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation* (May), 52–59.
- PRAKASH, A. AND SUBRAMANIAN, R. 1991. Filter: An algorithm for reducing cascaded rollbacks in optimistic distributed simulations. In *Proceedings of the 24th Annual Simulation Symposium* (April), 123–132.
- RAJAEI, H., AYANI, R., AND THORELLI, L.-E. 1993. The Local Time Warp approach to parallel simulation. In *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation* (May), 119–126.
- REIHER, P. L. AND JEFFERSON, D. 1989. Limitation of optimism in the Time Warp operating system. In *Proceedings of the 1989 Winter Simulation Conference* (Dec.), 765–770.
- REYNOLDS, P. F., JR. 1988. A spectrum of options for parallel simulation. In *Proceedings of the 1988 Winter Simulation Conference* (Dec.), 325–332.
- REYNOLDS, P. F., JR. 1993. The Silver Bullet. *ORSA J. Comput.* 5, 3 (Summer), 239–241.
- REYNOLDS, P. F., JR., PANCERELLA, C. M., AND SRINIVASAN, S. 1993. Design and performance analysis of hardware support for parallel simulations. *J. Parallel Distrib. Comput.* 18, 435–453.
- REYNOLDS, P. F., JR., WILLIAMS, C. C., AND WAGNER, R. R., JR. 1997. Isotach networks. *IEEE Trans. Parallel Distrib. Syst.* 8, 4 (April), 337–348.

- SOKOL, L. M., BRISCOE, D. P., AND WIELAND, A. P. 1988. MTW: A strategy for scheduling discrete events for concurrent execution. In *Proceedings of the SCS Multiconference on Distributed Simulation* (July), 34–42.
- SRINIVASAN, S. 1995. NPSI adaptive synchronization algorithms for PDES. Ph.D. Thesis, Department of Computer Science, University of Virginia (August).
- SRINIVASAN, S. AND REYNOLDS, P. F., JR. 1993. Non-interfering GVT computation via asynchronous global reductions. In *Proceedings of the 1993 Winter Simulation Conference* (Dec.) 740–749.
- SRINIVASAN, S. AND REYNOLDS, P. F., JR. 1995a. Adaptive algorithms vs. Time Warp: An analytical comparison. In *Proceedings of the 1995 Winter Simulation Conference* (Dec.), 666–673.
- SRINIVASAN, S. AND REYNOLDS, P. F., JR. 1995b. NPSI adaptive synchronization algorithms for PDES. In *Proceedings of the 1995 Winter Simulation Conference* (Dec.), 658–665.
- STEINMAN, J. S. 1991. Interactive SPEEDES. In *Proceedings of the 24th Annual Simulation Symposium*, 149–158.
- STEINMAN, J. S. 1993. Breathing Time Warp. In *Proceedings of the Seventh Workshop on Parallel and Distributed Simulation* (May), 109–118.
- TURNER, S. J. AND XU, M. Q. 1992. Performance evaluation of the Bounded Time Warp algorithm. In *Proceedings of the Sixth Workshop on Parallel and Distributed Simulation* (Jan.), 117–126.

Received March 1996; accepted January 1998