Adaptive Memory Management and Optimism Control in Time Warp

Samir R. Das
The University of Texas at San Antonio
and
Richard M. Fujimoto
Georgia Institute of Technology

It is widely believed that the Time Warp protocol for parallel discrete event simulation is prone to two potential problems: an excessive amount of wasted, rolled back computation resulting from "rollback thrashing" behaviors, and inefficient use of memory, leading to poor performance of virtual memory and/or multiprocessor cache systems. An adaptive mechanism is proposed based on the Cancelback memory management protocol for shared-memory multiprocessors that dynamically controls the amount of memory used in the simulation in order to maximize performance. The proposed mechanism is adaptive in the sense that it monitors the execution of the Time Warp program, and using simple models, *automatically* adjusts the amount of memory used to reduce Time Warp overheads (fossil collection, Cancelback, the amount of rolled back computation, etc.) to a manageable level. We dscribe an implementation of this mechanism on a shared memory, Kendall Square Research KSR-1, mutiprocessor and demonsrtate its effectiveness in automatically maximizing performance while minimizing memory utilization, for several synthetic and benchmark discrete event simulation applications. We also demonstrate the adaptive ability of the mechanism by showing that it "tracks" the time-varying nature of a communication network simulation.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiprocessors — MIMD; C.4 [**Performance of Systems**]: Measurement techniques; D.4.1 [**Operating Sys-**

A preliminary version of this work appeared in the Proceedings of the 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, May 1994.
This work was supported by NSF grant number MIP-94085550 and Innovative Science and Technology contract number DASG60-93-C-0126 provided by the Ballistic Missile Defense Organization and managed through the Strategic Defense Command Advanced Technology Directorate Processing Division. The work of the first author is also supported by DoD/AFOSR grant number F49620-96-1-0472 and NSF grant number CDA-9529541.
Name: Samir R. Das
Affiliation: Division of Computer Science, The University of Texas at San Antonio
Address: San Antonio, TX 78249-0667, email: samir@cs.utsa.edu

Name: Richard M. Fujimoto
Affiliation: College of Computing, Georgia Institute of Technology
Address: Atlanta, GA 30332-0280, email: fujimoto@cc.gatech.edu

tems]: Process Management — synchronization; I.6.7 [**Simulation and Modeling**]: Simulation Support Systems

General Terms: Experimentation, Measurement, Performance

Additional Key Words and Phrases: Parallel discrete event simulation, rollback, Time Warp

1. INTRODUCTION

Time Warp [Jefferson 1985] is an optimistic synchronization protocol that uses run-time detection of errors caused by out of order execution of portions of a parallel computation, and recovery using a rollback mechanism. To date, it has been largely applied to discrete event simulation problems, although other applications, e.g., concurrency control in database systems, have been proposed [Jefferson and Motro 1986]. The principal advantages of Time Warp over more conventional, blocking-based, synchronization protocols is that Time Warp offers the potential for greater exploitation of parallelism and, perhaps more importantly, greater transparency of the synchronization mechanism to the simulation programmer. The latter is due to the fact that Time Warp is less reliant on application specific information regarding which computations depend on which others. Time Warp has demonstrated some success in speeding up simulations of combat models [Wieland et al. 1989], communication networks [Carothers et al. 1994; Presley et al. 1989], queueing networks [Fujimoto 1989], and digital logic circuits [Briner 1991], among many others.

Despite these advantages, Time Warp is not without its difficulties. It is widely believed that Time Warp is prone to inefficient execution in certain situations due to "thrashing" behaviors where the system experiences excessively long and/or frequent rollbacks. Examples of pathological, overly optimistic behaviors that lead to extremely poor performance are described in [Fujimoto 1990a; Lubachevsky et al. 1991]. Further, Time Warp has the potential of requiring large amounts of memory because a certain amount of history information must be maintained to allow rollback. This increases memory usage. In shared memory systems large memory usage coupled with the unpredictable memory access dynamics of a parallel simulation system may cause loss of spatial locality, leading to performance degradations by inducing excessive paging or poor cache performance.

To address the rollback thrashing problem, several protocols have been proposed that attempt to capture the principal benefits of Time Warp, but without its liabilities (see [Fujimoto 1990a; Nicol and Fujimoto 1994] for a review). These protocols usually attempt to limit the amount of optimistic (i.e., prone to rollback) execution in order to avoid rollback thrashing. For instance, many protocols limit execution to a sliding interval or "window" of simulated time that prevents processes from advancing too far ahead of others (e.g., see [Sokol and Stucky 1990]).

A second, largely independent thread of research has been concerned with the problem that Time Warp may consume an excessive amount of memory. Much of this work has focused on developing and understanding protocols that allow Time Warp to execute in limited memory environments. Examples include Jefferson's *Cancelback* protocol [Jefferson 1990] and Lin and Preiss's *Artificial Rollback* protocol [Lin and Preiss 1991] that allow Time Warp to execute using any amount memory greater than or equal to that required by a sequential execution of the program. In prior work, we have analyzed performance/memory tradeoffs in the Cancelback protocol both analytically [Akyildiz et al. 1993] and experimentally [Das and Fujimoto 1993; Das and Fujimoto 1997], and observed that Time Warp is able to achieve efficient execution with only a modest amount of memory beyond that required for sequential execution.

Here, we propose the marriage of memory management and limited optimism synchronization protocols. This is motivated by the fact that the amount of memory allocated to a Time Warp simulation automatically limits the amount of optimistic execution, i.e., the degree to which processes may advance ahead of other processes. Specifically, building on an understanding of the behavior of Time Warp in limited memory environments, we propose a protocol that provides sufficient memory for Time Warp to execute efficiently, but does not provide so much memory that overly optimistic execution can occur. Thus, we use one mechanism to simultaneously address both the rollback thrashing and memory management issues.

The proposed protocol is adaptive in that it monitors the execution of the Time Warp program, and automatically adjusts the amount of memory provided to the parallel simulator to maximize performance. An adaptive protocol is necessary because the appropriate parameters of both the synchronization and memory management protocols are dependent on characteristics of the application, e.g., the amount of symmetry and homogeneity among the simulation processes, as well as the minimum amount of memory required to execute the program using Time Warp. These characteristics may change dramatically during the course of a single simulation run. For instance, "hot spots" or surges in message traffic may come and go throughout a simulation of a communication network. Combat models move through phases of "advance," "engage the enemy in conflict," and "clean up." The synchronization and memory management protocols must adapt to these changing behaviors so that it can perform equally well in each one. We take a "holistic" approach to adaptation that attempts to minimize total execution time rather than concentrating on one specific criteria, e.g., minimizing the amount of rolled back computation. This prevents the mechanism from optimizing for one aspect of the computation, e.g., minimizing rollback, at the expense of a disproportionate increase in another, e.g., memory reclamation (fossil collection) overheads.

The remainder of this paper is organized as follows. In the next section we briefly review the basics of the Time Warp mechanism and the Cancelback memory management protocol which forms the basis for our work. We then present a simple model to characterize the execution of Time Warp programs as a flow of memory buffers among different "buffer pools," and decompose the execution time of a Time Warp program into certain measurable quantities, e.g., the amount of time spent performing memory reclamation and rollback. We then describe how these components of the total execution time are affected as the amount of memory is varied. Experimental data is presented to support these predictions. Based on the above characterization, we propose an adaptive protocol that measures the quantities used in the performance model, and automatically adjusts parameters used by the memory system to maximize performance. Finally, we demonstrate the effectiveness of this approach by presenting measurements of the overall performance of an implementation of this adaptive protocol on a Kendall Square Research KSR-1 machine across a variety of simulation workloads.

2. TIME WARP, CANCELBACK, AND OTHER RELATED WORK

A Time Warp program consists of a collection of *logical processes* (LPs) that interact by exchanging timestamped *messages* or *events* (these terms are used synonymously here). The timestamp indicates when the event occurs in simulated time. Each LP must process its incoming messages in timestamp order in order to guarantee correct results, i.e., results that are identical to a sequential execution of the program. If an LP processes events out of timestamp order, e.g., because it receives a message containing a timestamp smaller than some other message(s) that it has already processed, it rolls back the corresponding event computations that were processed out of sequence, and reexecutes them in timestamp order. Rolling back an event entails restoring the state of the LP to that which existed prior to processing the event, and "unsending" messages that were sent during the course of processing the event. Checkpointing or incremental state saving techniques are used to restore the state of the LP. Unsending a previously sent message is accomplished by sending an *anti-message* that will *cancel* (annihilate) the previously sent message. If the cancelled message had already been processed when the anti-message is received, the receiver is first rolled back (possibly generating additional anti-messages) prior to the message cancellation.

In order to reclaim memory (e.g., processed messages and snapshots of the LP's state), and to allow operations that cannot be rolled back (e.g., I/O), *global virtual time* (GVT) is defined. GVT is a lower bound of the timestamp of any rollback that might later occur, and is operationally defined as the smallest timestamp of any unprocessed or partially processed message or anti-message in the system.[1] GVT defines the *commitment horizon* of the simulation. Events with timestamp less than GVT are referred to as *committed* events. Irrevocable operations that were invoked by committed events can be performed, and storage occupied by them or their state can be reclaimed. The latter operation is called *fossil collection*.

2.1 Protocols Using Limited Optimism

As mentioned earlier, several protocols have been proposed to limit the amount of optimistic computation in Time Warp, thereby limiting the amount of rollback. Many protocols limit execution to only those processes that lie within a simulated time window [Reiher et al. 1989; Sokol and Stucky 1990; Turner and Xu 1992]. Other approaches to reducing optimism include limiting the number of events each LP may execute beyond GVT [Steinman 1993], generating additional rollbacks (in addition to those produced by synchronization errors) at stochastically selected points in real time [Madisetti et al. 1993], not sending messages unless they are guaranteed to be correct, thereby eliminating the need for anti-messages [Steinman 1992; Dickens and Reynolds, Jr. 1990; Mehl 1991], and utilizing a process's ability to predict future simulation events (lookahead) to limit optimism [Lubachevsky et al. 1989]. Rajaei et al. [Rajaei et al. 1993] and Arvind and Smart [Arvind and Smart 1992] propose "composite" simulation protocols that integrate conservative

[1] Certain memory management protocols used with Time Warp require a somewhat different operational definition of GVT, but this aspect is ignored here.

and optimistic protocols. Many of the aforementioned approaches utilize one or more "tuning parameters" (e.g., size of the time window in time window based schemes). However, it is not readily clear how users, particularly those that are not intimately familiar with details of the underlying simulation protocol, should set these parameters for their particular application. Further, these protocols do not address the question of adapting the synchronization mechanism to a dynamically varying workload. While one could easily add the ability to dynamically change these control parameters to many of these protocols, it is not clear what criteria an application program would use, nor to what degree they should adjust these parameters during the program's execution.

By comparison, only a modest amount of work has been devoted to studying adaptive simulation mechanisms. Early efforts were made by Reiher and Jefferson [Reiher et al. 1989] and Ball and Hoyt [Ball and Hoyt 1990]. In the former, the process scheduling algorithm is manipulated to reduce the frequency of executing processes that have experienced many rollbacks in the past. In the latter, processes are scheduled only within a time window whose size is frequently adjusted by monitoring the execution. The goal is to minimize the sum total of rollback overheads and loss of performance due to blocking at the edge of the window. Reiher and Jefferson report little improvement over ordinary Time Warp for several benchmark simulation models. Ball and Hoyt do not report any performance study on real parallel systems.

More recently, Hamnes and Tripathi [Hamnes and Tripathi 1994b] has introduced the *local adaptive protocol* which uses input channel specific information for each LP to estimate a real time blocking window. Average inter-arrival times (both in simulation time and real time) of messages in each input channel are used to predict whether an event arriving later in one of the empty input channels will cause a rollback. This scheme has been shown to outperform both Time Warp and purely conservative methods for certain closed queueing network simulations [Hamnes and Tripathi 1994a]. Ferscha introduced a probabilistic, adaptive optimism control scheme that predicts the arrival pattern of incoming messages to an LP using certain time-series based forecasting, and probabilistically blocks the LP based on the forecast [Ferscha 1995]. Several forecasting methods have been explored, such as methods based on central tendencies as well as autoregressive moving average (ARMA) models. He demonstrates a superior performance to Time Warp for small scale stochastic petri net simulations. Srinivasan and Reynolds has proposed a class of adaptive algorithms, called *near perfect state information* (or NPSI) protocols [Srinivasan and Reynolds 1995]. They rely on the availability of a "near perfect" information on some aspects of the global simulation state and use such information to estimate the *error potential* for each individual LP. Each LP is selectively blocked based on its error potential. Performance advantages over Time Warp are reported for small synthetic workloads.

These techniques attempt to optimize only one aspect of the dynamics of Time Warp, namely rolled back computation, and do not directly control the usage of memory resources. We contend that even if sufficient memory (i.e., the amount required for uncontrolled execution) can be provided, keeping a tight rein on memory usage is important, especially on shared memory systems. In such systems, it is advantageous to allocate memory from a shared pool so that the available memory can be effectively utilized. The deallocated memory is returned to this shared pool. However, this can lead to the loss of spatial locality in a Time Warp based parallel simulator for the following reason. The necessary memory for each event message is allocated from the shared pool by the sender. This memory is then read and possibly written by the receiver before it is finally deallocated, when it is free to be allocated by another processor. In a parallel simulator, message sending is very frequent (may be several per event) and event granularity is often small. Thus the cache transactions and virtual memory related overheads due the lack of spatial locality may have a significant effect on the performance. Using a limited amount of memory with a suitable memory management protocol may improve spatial locality for better performance. One, however, must be careful so that memory management overheads may not dominate.

The reported optimism controlling techniques can control memory usage, but only indirectly. Moreover, their effectiveness for unbalanced and/or dynamic, time-varying simulation models is yet to be demonstrated. In the present work we take a memory based approach to control Time Warp's optimism. The goal is take a broader view for adapting Time Warp's optimism, optimizing the total execution time which includes *all* major operational overheads, which include rollbacks, and memory related overheads. The latter in turn includes protocol overheads (such as GVT computation, fossil collection, and Cancelback), and overheads due to the loss spatial locality (such as virtual memory and caching overheads).

2.2 Memory Management Protocols

Approaches to limit memory utilization in Time Warp include *passive* schemes such as infrequent [Lin et al. 1993; Fleischmann and Wilsey 1995] and incremental state saving [Palaniswamy and Wilsey 1993; Cleary et al. 1994]. They reduce "average" memory usage, but cannot necessarily recover memory when storage has been exhausted. Infrequent state saving schemes may be adaptive in that they follow the dynamics of the Time Warp system to determine, at run-time, the optimal state saving frequency [Fleischmann and Wilsey 1995]. *Active* schemes, on the other hand can reclaim memory "on demand." Some active schemes based on rollbacks have the storage optimal property, i.e., they are able to complete the simulation using no more memory than the corresponding sequential execution [Lin and Preiss 1991]. Here, we focus attention on these active schemes because of their ability to reclaim memory on demand and the storage optimal property. It is important to note that the passive schemes can also be used in conjunction with active ones.

Two storage-optimal active schemes are Cancelback [Jefferson 1990] and Artificial Rollback [Lin and Preiss 1991]. They are based on rollbacks and are invoked when the system has run out of memory, and fossil collection cannot reclaim any more. When this occurs, it may *not* be appropriate to terminate the program because this situation may have resulted from over-optimistic execution by the underlying system rather than excessive memory requirements of the application itself. To resolve this situation, Cancelback and Artificial Rollback roll back some process(es), and reclaim the memory that is freed when the rollback causes previously saved state vectors to be released and message/anti-message pairs to be annihilated. To maximize efficiency, the processes that are the furthest ahead in virtual time are rolled back. In the absence of other information one would expect that these processes are the most likely to be over-optimistic in their execution. In essence this approach retracts some possibly correct computations that are far ahead in virtual time to make room for more older computations to proceed.

Artificial Rollback simply identifies the LP(s) that is (are) the furthest ahead in simulated time, and then rolls it (them) back. Cancelback accomplishes rollback indirectly by "sending back" certain messages that have been received, which in turn, rolls back the sender. In both cases, the virtual time at which the memory was allocated (e.g., the timestamp of the sender in Cancelback) is used as the criteria for determining which state should be reclaimed; higher timestamped state is reclaimed in preference to lower timestamped state. If no such memory can be reclaimed because the resulting rollback will extend beyond (before) GVT, the program is terminated and an error message is returned to the user. In this case, the user has asked Time Warp to execute the program using less memory than that required in a sequential execution, so it is reasonable to terminate the program. Both protocols rely on a globally shared pool of memory to achieve the storage optimal property, so they are best suited for implementation on shared memory multiprocessors.

The protocol used here, described next, is based on Time Warp using the Cancelback protocol, but can also be applied to the Artificial Rollback protocol.

3. AN ADAPTIVE, MEMORY-BASED PROTOCOL

The basis for the proposed adaptive mechanism lies in an understanding of the performance of Time Warp using Cancelback as the amount of memory provided to the computation is varied. In analytic and experimental studies of the Cancelback protocol for various simulation models, it was found that performance is poor for very small amounts of memory (close to that required for sequential execution) because there is not enough memory to sustain sufficient parallel execution and memory management overheads are excessive [Akyildiz et al. 1993; Das and Fujimoto 1993; Das and Fujimoto 1997]. As memory is increased, there is a dramatic improvement in performance, and a distinct "knee" in the performance vs. memory curve is observed (see the upper curve in Figure 1) [Das and Fujimoto 1997]. This behavior occurs for both symmetric and asymmetric, unbalanced workloads. The location of the knee, however, may be different for different simulation models. In the experimental studies, performance losses were also observed when using large amounts of memory due to loss of spatial locality and less efficient use of the host cache [Das and Fujimoto 1997]; rollback thrashing would similarly degrade performance when large amounts of memory are used for certain workloads.

It is clear from Figure 1 that it would be best to operate just beyond the "knee" of the performance curve, or at least in the "flat" portion before performance begins to decline with additional memory. Although Figure 1 indicates a relatively large range of memory values where performance is insensitive to the amount of memory, one should keep in mind that for large programs that require most of the physical memory available in the system, the "flat" portion of the curve in Figure 1 becomes rather small, so providing only a modest amount of memory beyond the knee could incur significant performance degradations due to the virtual memory system. Thus, a reasonable goal is to control the amount of memory provided to

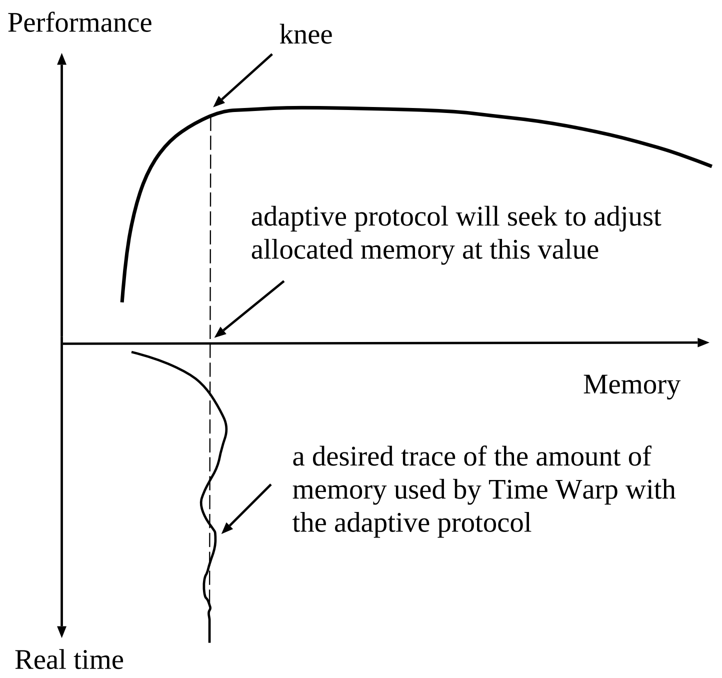


Fig. 1. Performance memory trade-off in Time Warp and the adaptive protocol. The protocol begins with a very small amount of memory and automatically adjusts it to reach the knee of the performance memory curve.

operate the simulation right at the knee of the curve.

Unfortunately, users cannot easily determine the exact location of the knee for most applications because (1) it depends in complex ways on aspects of the simulation such as the amount of homogeneity and symmetry in the workload, amount of computation per simulation event (i.e., grain size) and memory management overheads and (2) its location will usually change dynamically during the the course of the computation; the amount of memory required by most simulations (even sequential ones) varies dynamically, and the symmetry and homogeneity characteristics of the application also change as the system being simulated evolves. This necessitates a runtime mechanism that automatically monitors the execution and adjust the amount of memory to the optimal value, as shown in the lower curve in Figure 1.

To simplify the discussion that follows, we assume the system manages some number of identical *memory buffers.* Each buffer includes storage for a single event message, a state vector (to hold a copy of the state of an LP), and pointers to other events scheduled by processing this event (these pointers are, in effect, anti-messages) [Fujimoto 1989]. Here, the buffer is used as the atomic unit of memory provided by the system. However, our approach easily generalizes to objects with different sizes.

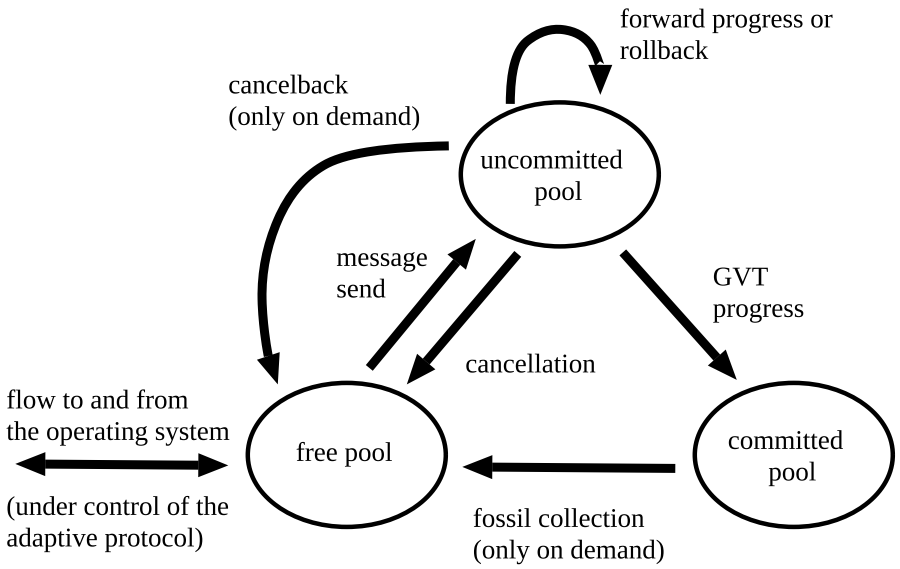


Fig. 2. Different transitions responsible for growth and shrinkage of three individual buffer pools.

3.1 Behavior of the Cancelback Protocol

It is instructive to consider the behavior of Time Warp with Cancelback in terms of its usage of memory buffers. At any instant, the buffers in the system can be viewed as being partitioned into three pools, as shown in Figure 2:

(1) *uncommitted* buffers holding uncommitted events,

(2) *committed* buffers holding processed, committed events that will be reclaimed when fossil collection is invoked next, and

(3) *free* buffers that are currently unallocated.

The uncommitted buffer pool holds two types of events: *processed* and *unprocessed* events. Execution of an unprocessed event changes its state to processed, while a rollback of a processed event has the opposite effect. The uncommitted buffer pool can also be classified into *future* and *present* events based on the relation of the timestamps of the events to GVT. The events with both send and receive timestamps greater than GVT are called future events. Those with a send time less than or equal to and receive time greater than or equal to GVT are called present events. It can be seen that the present events are also those that exist in a sequential simulation that had advanced to a simulated time equal to GVT.

The execution of Time Warp with Cancelback can be viewed as a *flow of buffers* among these three pools (see Figure 2). Each message send causes a single buffer to move from the free to the uncommitted event pool (an unprocessed event is created). When that event is processed (forward progress), it becomes a processed, uncommitted event, and remains in the uncommitted pool. Processing an event may cause GVT to advance. If this occurs, one or more processed buffers move from the uncommitted to the committed pool. Committed buffers are returned to the free pool when fossil collection is invoked. Finally, a buffer moves from the uncommitted to the free pool with each message cancellation. This cancellation may occur as the result of an ordinary Time Warp message annihilation or an invocation of Cancelback. Note that here, fossil collection is invoked only on demand, when the free pool is empty. Cancelback is also invoked on demand, when both the free pool and the committed pools are empty. In addition, the adaptive protocol can allocate (deallocate) buffers to (from) the free pool from (to) the system memory managed by the operating system, thus varying the total number of buffers allocated to Time Warp.

Figure 3 shows a typical trace of a Time Warp execution, indicating the number of buffers in each of these three pools. To simplify the explanation, let us assume that there are always a fixed number of unprocessed messages (the actual adaptive protocol makes *no* such assumption). This will be true for simulations where message fanout is one, i.e., each event sends exactly one message. Thus the number of unprocessed messages remains constant. Let us also assume that there are a fixed number of buffers available to Time Warp making the total number of buffers in the three buffer pools a constant. Again this assumption is only to simplify presentation, as the adaptive protocol routinely varies the number of buffers being used.

A "cycle" begins with a call to Cancelback (marked "cb" in Figure 3). After Cancelback is invoked, there will be some number M_f buffers in the free pool (with the fixed memory assumption, exactly M_f events need to be canceled by the Cancelback protocol to achieve this), and some number M_p buffers in the uncommitted pool holding processed uncommitted events. The committed pool is empty. When execution resumes, the uncommitted pool expands as new events are created (via message sends) and processed, depleting the free buffer pool. Similarly, GVT advances cause the committed pool to expand at the expense of the uncommitted pool. Occasionally, cancellations cause the free buffer pool to be replenished taking buffers away from the uncommitted pool, but the overall trend is to drain the free buffer pool. When the free pool is empty, fossil collection is called, emptying all of the committed buffers into the free pool. This process repeats until we find the free pool is empty, and there are no committed buffers, at which time Cancelback is again called, signaling the end of the cycle.

Note that the size of the committed pool must be monotonically increasing between fossil collections. However, the other two pools may grow and shrink depending on the forward progress rate, GVT progress rate, and rollback rate of the execution. However, there will often be a net rate of growth of the uncommitted pool in a cycle, particularly for unbalanced simulations. Cancelback is invoked whenever all memory buffers are occupied by the uncommitted buffers.

3.2 Execution Time Profile

Based on the buffer flow model just described, the execution time of the Time Warp program can be categorized into the following components:

(1) *the time spent processing events that eventually commit.* We include in this time a fixed state saving and queue management overhead for each event. These overheads are independent of memory management effects, and can be treated

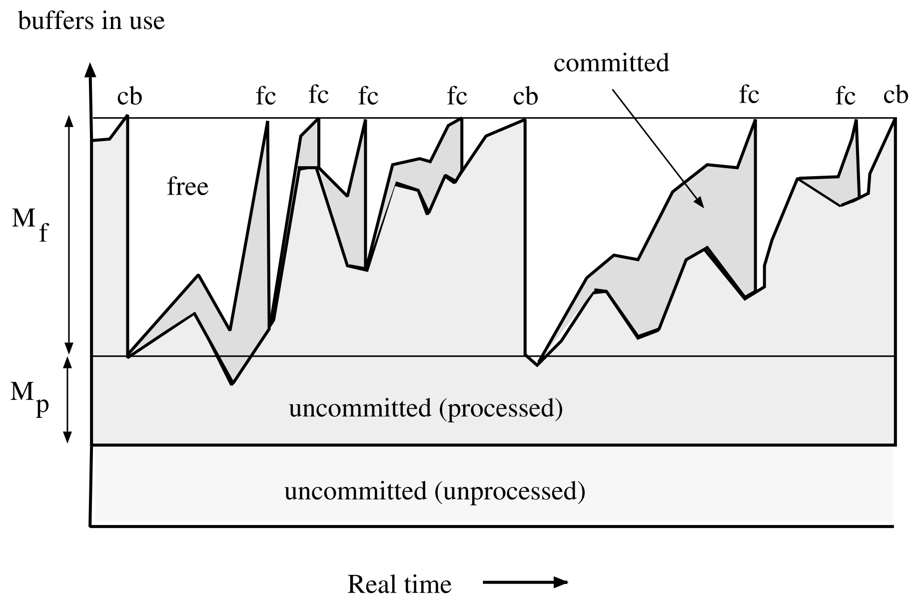


Fig. 3. A typical trace of showing flow of buffers into different pools during a Time Warp execution under memory constraints. This shows how fossil collection (fc) and Cancelback (cb) are invoked. M_f is the number of buffers in the free pool after Cancelback is invoked. M_p is the number of buffers in the uncommitted pool after Cancelback is invoked. (In the figure M_f and M_f are assumed to be constant for all Cancelback invocations and the total number of buffers is also assumed to be constant. But in reality M_f and M_p, and thus the total number of buffers will be dynamically determined by the adaptive protocol.)

as constants because constant time queue insertions are possible with a data structure such as the calendar queue [Brown 1988; Rönngren et al. 1993], and state saving overhead is constant if full copies are made at regular intervals or incremental state saving overheads in each event are the same.

(2) *the time spent processing events that are later rolled back during Time Warp execution.* We similarly include in this time a fixed state save and queue management overhead for each event. In addition we also include a fixed overhead for each event to realize the rollback operation (e.g., message cancellations). One would expect the *average* fanout (number of messages sent by an event) for each event to be 1.0, because otherwise, the event list grows or shrinks in proportion to the length of the simulation run. This seldom occurs in practice. Thus it is reasonable to assign a fixed rollback overhead to each event computation that is later rolled back.

(3) *the time spent processing events that are later rolled back by the Cancelback protocol.* We similarly include state save, queue management, but not the rollback overhead, which is counted in the next category, in this time.

(4) *the time associated with invoking Cancelback and fossil collection.* This includes the time to compute GVT, select committed events (if any), and return them (if any) to the free pool. For Cancelback, this time also includes determining

the highest send timestamped events to rollback, and the cancellations and rollbacks associated with Cancelback. Because of the global computations involved (GVT computations and selection of events to Cancelback) this component was found to be a major factor in the execution time, if it is invoked very frequently.

(5) *"system" time in the memory management system of the underlying architecture.* This time includes paging and cache management overheads.

The first component listed above is fixed for any simulation run, independent of how the program is executed. The adaptive protocol attempts to directly minimize the 2nd, 3rd, and 4th components, and indirectly minimize the last component. The last component is difficult to control directly because virtual memory performance degradations may not be easily distinguished from large event computation times. We control this overhead indirectly by minimizing the amount of memory usage, subject to constraints on the other overheads. If the multiprocessor provides on-line information regarding the performance of the virtual memory system, this could be factored directly into the adaptive scheme. However, we do not pursue this aspect here.

3.3 The Adaptive Protocol

In the discussion that follows, we use the following notation:

—P denotes the number of processors in the system.

—M denotes the number of memory buffers provided by the system *beyond* that is required in the sequential execution.

—M_f denotes the number of buffers that are moved to the free pool after Cancelback is invoked. These M_f buffers are obtained by cancellations caused by the Cancelback protocol and/or allocating additional memory from the multiprocessor operating system (in contrast with Figure 3 where all M_f buffers were obtained by cancellations).

—M_p denotes the number of *processed* uncommitted events that are left in the uncommitted pool after Cancelback is invoked. Note that the uncommitted pool also contains unprocessed events in addition to these.

—T_{ev} denotes the average time to process an event, including queue management and state saving overheads.

—N_{orb} denotes the number of events rolled back via ordinary Time Warp rollbacks (i.e., *not* rollbacks via Cancelback) during a cycle.

—N_{com} denotes the total number of events committed in the system during a cycle.

—e, the efficiency, is defined as $N_{com}/(N_{com}+N_{orb})$, i.e., the fraction of processed events (ignoring those that are rolled back via Cancelback) that were committed in the last cycle. Both N_{com} and e is based on the total system rather than individual processors or LPs.

Briefly, the adaptive protocol attempts to optimize execution time by focussing on the execution time components described in the previous subsection. As described below, M_f, which controls the initial size of the free pool immediately after Cancelback is called, is used to control the amount of time before Cancelback will have to be called again. This in turn controls the amount of overhead incurred by the Cancelback and fossil collection mechanisms. M_p is used to control the amount of degradation resulting from Cancelback rolling back events; M_p also helps to limit the amount of overly optimistic execution.

From the flow model it is apparent that larger values M_f tend to reduce the frequency of Cancelback and fossil collection by providing more free memory to the Time Warp system at the beginning of the cycle. Later, this is verified experimentally. Thus a larger value of M_f will reduce Cancelback and fossil collection overheads. However, too large an M_f value may also lead to rollback thrashing and/or virtual memory effects as described before. Thus our goal is to allow Cancelback and fossil collection to occur at a "controlled" rate, which will imply a non-zero but tolerable performance degradation due to these operations. But this will indirectly keep M_f, and in turn M, from becoming too large, which might risk introducing rollback thrashing and memory management effects.

However, in spite of this control, in situations where it appears that the system is entering a rollback thrashing behavior (e.g., due to a sudden change in behavior within the application), M is immediately reduced by cutting back both M_f and M_p simultaneously. Thus, in general, the adaptive protocol attempts to minimize both M_f and M_p subject to the constraints that (i) M_f is not so small that the time spent in Cancelback and fossil collection is beyond a tolerable limit, and (ii) M_p is not so small that there is a perceptible degradation resulting from Cancelback rolling back too many events.

3.3.1 *Minimizing Cancelback and Fossil Collection Overheads.* The overhead associated with executing Cancelback and fossil collection is directly proportional to the frequency at which they are invoked. The adaptive protocol predicts this frequency as a function of M_f based on measurements of the Time Warp system in the last cycle. It then sets M_f to obtain the desired frequency of Cancelback calls. Here we control *only* the frequency of Cancelback calls (as opposed to fossil collection calls), because this automatically controls the fossil collection frequency. In the appendix, we develop and validate a model that predicts the frequency of fossil collection calls relative to Cancelback calls, allowing the adaptive protocol to control both of these quantities.

Recall that the time between two consecutive Cancelback invocations is the length of a cycle. Thus the expected length of a cycle determines the frequency of invoking Cancelback. The average time between calls to Cancelback (average length of a cycle) can be controlled by observing that it is invoked when all of the buffers in the system are in the uncommitted pool (see Figure 2). The rate of growth of the uncommitted pool is equal to:

$$R_p \ = \ eP/T_{ev} \ - \ R_{fc}, \tag{1}$$

where the first term is the rate at which new unprocessed events are being produced, assuming an average of one new event is produced for each processed event, and R_{fc} is the rate at which events are fossil collected. At the beginning of each cycle, there are M_f unused event buffers. Thus, the time until the next invocation of Cancelback is approximated as M_f/R_p, with M_f set to obtain the desired frequency of Cancelback invocations, i.e.,

$$M_f \ = \ (T_{cycle} - T_{cbfc})(eP/T_{ev} \ - \ R_{fc}), \tag{2}$$

where T_{cycle} is the desired length of a cycle and T_{cbfc} is the Cancelback and fossil collection overheads within the cycle. These overheads are directly proportional to the total number of Cancelback and fossil collection invocations in the cycle. The quantities e, T_{ev}, and R_{fc} are automatically measured by the Time Warp system to derive an appropriate value of M_f using the above equation.

It should be noted that the above equation is approximate because in practice, the number of processed, uncommitted events can change abruptly when rollbacks occur, rather than at the smooth continuous rate implied by equation (1). Similarly, R_{fc} is represented as a constant rate, but in practice fossils are collected in batches and possibly at unpredictable intervals. Also, because measured values are used to predict R_p, we implicitly assume that past behavior of the system is a good predictor of the future. Although this is not a good assumption when the simulation changes from one "phase" to another, measurements can be made at sufficiently frequent intervals to allow the system to rapidly adapt to the changes in the workload.

The above model indicates the expected value of T_{cycle} as a function of M_f. Our goal is to allow Cancelbacks to occur at a modest frequency. The allowable frequency is selected so that the time spent executing Cancelback and fossil collections is a very modest fraction of length of the cycle, i.e., $T_{cbfc} = \delta T_{cycle}$, where δ is small. A value of 0.1 was chosen for δ in all our experiments. Thus equation 2 can rewritten as

$$M_f \;=\; (T_{cycle} - \delta T_{cycle})(eP/T_{ev} \;-\; R_{fc}) \tag{3}$$

To assess the accuracy of the predicted estimates of T_{cycle} for different values of M_f (represented by equation 2), measurements were collected on an implementation of Time Warp with Cancelback on a Kendall Square Research KSR-1 multiprocessor. Details of the Time Warp kernel used can be found in [Das et al. 1994]. Implementation specifics of the Cancelback protocol are described in [Das and Fujimoto 1993; Das and Fujimoto 1997]. Two synthetic workloads are used in these initial studies. The first, called PHOLD, is a parameterized synthetic workload that is symmetric and homogeneous [Fujimoto 1990b]. This simulation contains a fixed number of messages that move from one LP to another in a stochastic manner. Each message generates one new message with a timestamp increment selected from an exponential distribution. The destination LP is selected from a uniform distribution. With respect to Time Warp, this simulation's behavior is similar to simulations of symmetric, closed queueing networks. In all our experiments with this workload, 8 LPs are used with each LP mapped onto one KSR-1 processor. A message population of 256 was used. The event computation grain is exponentially distributed with a mean of 5 ms.

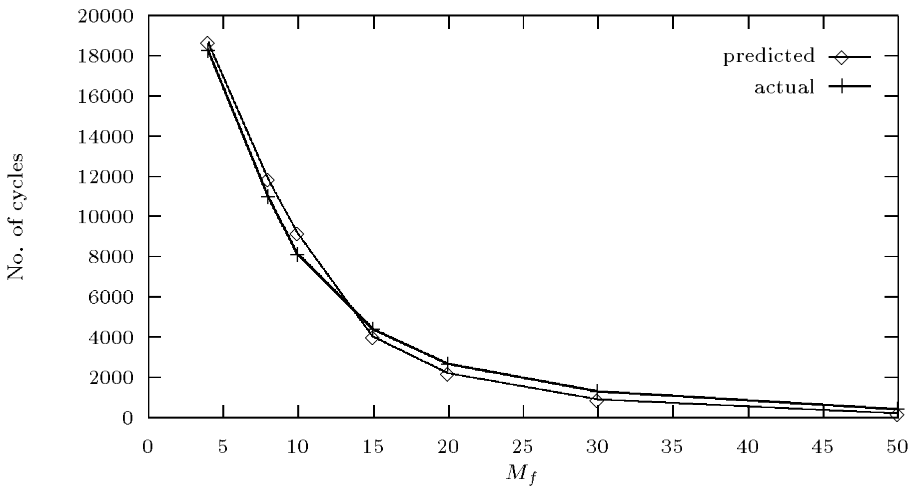
A second workload simulates an open and asymmetric network. It is called the *arbitrary flow network* model, and is based on simulations of electrical power distribution grids [Madisetti et al. 1993]. This simulation consists of a cluster of "fast" LPs that send messages to each other with a large timestamp increment, and "slow" LPs that use smaller timestamp increments (here, a factor of one hundred differentiates fast and slow LPs). Messages are occasionally exchanged between the slow and fast processes. Unlike PHOLD, this is a simulation of an open network, where the memory used in a sequential execution varies dynamically around some average value. A "source" process generates messages for each of the others. This source process never receives messages, so it never rolls back, but may be rolled back by the Cancelback protocol. A "sink" process receives messages, and discards them. Details of this workload are described in [Madisetti et al. 1993]. The unthrottled nature of the source process, and the fact that this workload is inherently asymmetric and unbalanced makes it a challenging test case for Time Warp. In all our experiments with this workload we use 10 LPs (1 source, 1 sink, 6 slow and 2 fast LPs) with a probability of communication 0.6 within the fast or slow set and 0.2 across the sets. Again each LP is mapped to one KSR-1 processor. The event grain is normally distributed with a mean of 1.5 ms.

Later, certain large communication network simulation benchmarks with dynamically varying message traffic (causing memory requirement to vary dynamically) will be used to demonstrate the ability of the mechanism to adapt to time varying workloads.

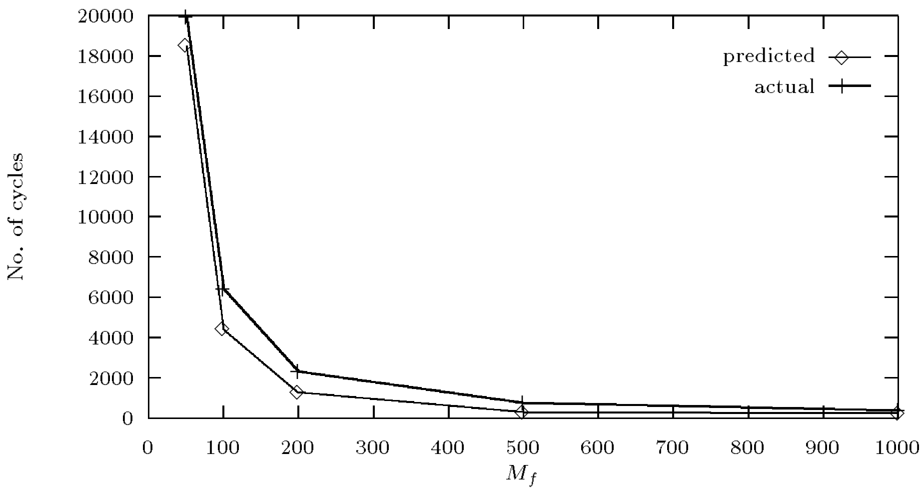
Figure 4 compares the predicted frequency of Cancelback invocations with the actual, measured frequency, for different values of M_f for both these workloads. The total number of Cancelback calls (i.e., total number of cycles) are plotted against M_f. This number is proportional to the frequency of invocations. For the "predicted" curve, equation 3 is used to compute T_{cycle} using measured averages for the parameters e, T_{ev} and R_{fc} over the total run-length of the simulation. The number of cycles is derived by dividing the run-length by T_{cycle}. The "actual" curve represents the number of cycles observed in the real experiment. As can be seen, there is a good agreement between the predicted and actual curves for both workloads, suggesting that the simple flow model described above can be used to accurately predict the frequency of Cancelbacks. In this set of experiments, M_p was fixed at an appropriate value as will be described next.

3.3.2 *Rolled Back Events.* M_p controls the degree to which some LPs are allowed to remain ahead of others when Cancelback is invoked. On the one hand, it is desirable to limit the number of such optimistic events because this will reduce the likelihood that the simulation will begin rollback thrashing behaviors. Also limiting M_p limits the total memory usage by the Time Warp system (which is the sum of M_f, M_p and the number of unprocessed events at the time of Cancelback). This reduces system memory management overheads, e.g., paging. On the other hand, limiting optimism (making M_p too small) can have a detrimental effect because it reduces the forward progress of the computation. In general, if Cancelback rolls back an event that is not only correct, but also on the critical path of the computation, then the overall execution time may be increased.

Thus, it must be determined how to set M_p to limit memory usage and the number of optimistic events, but not so much as to impact the critical path of the computation. Determining the critical path "on the fly" during a Time Warp execution may entail a significant amount of overhead. Also, whether an event has been on the critical path cannot be determined with complete certainty until it is committed. Thus an exact "on the fly" determination of the critical path was not considered to be a practical alternative. Rather than computing the exact critical path, we use an approximate, but less time consuming approach. When Cancelback is invoked, we roll back events containing high timestamps, as called for by the Cancelback mechanism. In addition, we mark those events that are rolled



(a) PHOLD benchmark on 8 processors



(b) Arbitrary Flow Network benchmark on 10 processors

Fig. 4. Comparison of the predictor for the frequency of Cancelback and the actual, measured frequency.

Initial condition:

1. G = 0.
2. All events in the free pool are always unmarked.

During Cancelback operation:

Mark all events rolled back.

During Time Warp operation:

1. Unmark any marked event rolled back.
2. For any marked event being processed check whether it is the GVT-regulator. If it is, increment G.

Fig. 5. The counter G estimates the number of critical events rolled back by Cancelback.

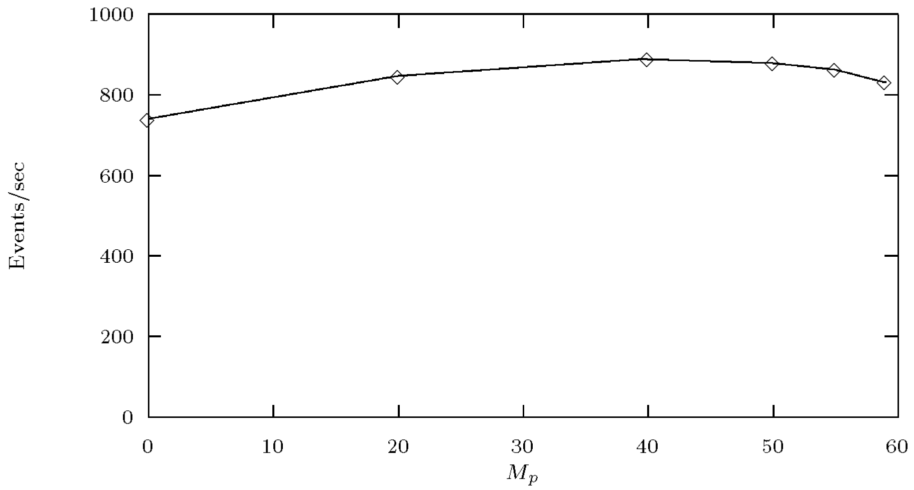
back by Cancelback. Once the computation resumes, we check whether or not each marked event has become the smallest timestamped event being processed in the system (the so-called GVT-regulator). If an event that is rolled back via Cancelback does become the GVT-regulator event,[2] then this indicates that GVT might have advanced at a more rapid rate had we not rolled back that event. Note that we remove the mark for an event that is rolled back during execution, as rollback of a marked event during Time Warp execution indicates that the event was incorrect and cannot be on the critical path (see Figure 5).

Let G denote the number of marked events that become the GVT-regulator. The adaptive protocol will continually reduce the parameter M_p so long as G remains negligible. This ensures that M_p remains at a point that minimizes memory usage, but without degrading performance. If G becomes non-negligible during the execution, e.g., because of a change in the workload, M_p could be doubled (say) to eliminate this effect.

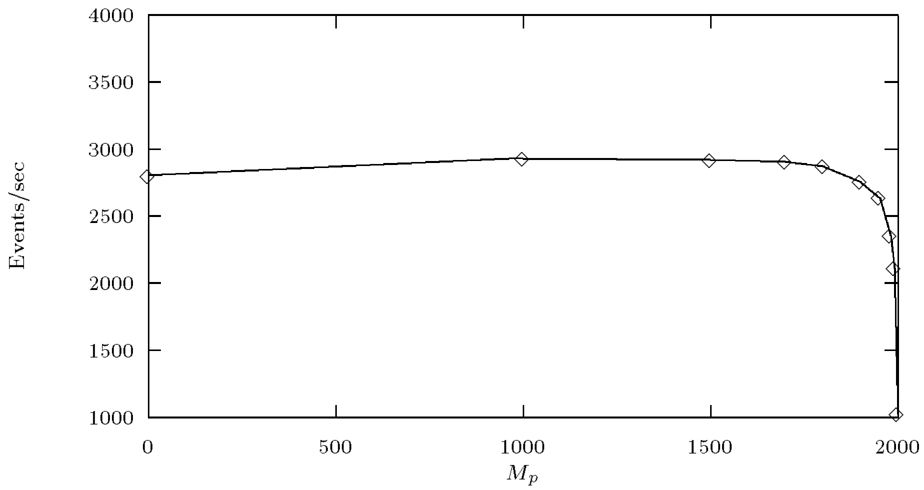
The algorithm clearly only approximates the number of events that should *not* have been rolled back in the Cancelback procedure. The effectiveness of this approach was evaluated experimentally. Figure 6 shows the overall execution time of the PHOLD and arbitrary flow simulations as M_p is varied for a fixed M. Here, M is fixed near the knee of the performance-memory curve for the workload. The curves in Figure 6 demonstrate that the procedure described above is effective in locating the point where execution time is minimized. Further, the curves reveal that the execution time is relatively insensitive to M_p except at extremely large and small values. This is, in fact, our general observation other fixed values of M. This is a positive result in that it indicates that in practice, the adaptive algorithm need not be very precise in estimating M_p (or G) in order to maximize performance.

3.3.3 *Rollback Thrashing.* The above precautions directly control the frequency that fossil collection and Cancelback are invoked, and performance degradations resulting from rolling back too many computations via Cancelback. These techniques only indirectly reduce the likelihood of rollback thrashing behaviors. Small values of M_p effectively control thrashing at the beginning of a cycle. Also as we control

[2]Determining GVT regulators frequently may entail an additional overhead. We avoid this by estimating the GVT-regulator LP *only* at the time of the GVT computation and incrementing G whenever this LP processes a marked event.



(a) PHOLD, predicted lower bound on M_p = 35



(b) Arbitrary Flow Network, predicted lower bound on M_p = 109

Fig. 6. Effect of choice of M_p on the Time Warp performance. Performance is measured in terms of events/sec.

(1) **If** this is a forced Cancelback, **then**
set $M_p = nP$ and $M_f = n$,
Exit
(2) **Else if** $\Delta e/e < -l$, **then**
set $M_f = M_f/2$ and $M_p = M_p/2$.
Exit
(3) **Else**
(i) $f = \frac{T_{cbfc}}{T_{cycle}}$.
(ii) **If** $\delta - \Delta\delta < f < \delta + \Delta\delta$, **then goto step 4**
Else
(iii) $T^{goal}_{cycle} = T_{cbfc}/\delta$
(iv) Set M_f to meet T^{goal}_{cycle} (using equation 3)
(4) **If** $G > \epsilon N_{com}$, **then** $M_p = (1 + a)M_p$.
Else $M_p = (1 - a)M_p$.
Exit

Fig. 7. The adaptive protocol

M_f to allow Cancelback to occur at a controlled rate, and Cancelback always eliminates overly optimistic computation to reclaim memory, thrashing cannot sustain for long before the next Cancelback. Nevertheless thrashing is possible, e.g., if there is a dynamic change of the nature of the workload. To provide a direct control to prevent rollback thrashing, a third mechanism is introduced. This mechanism measures the efficiency of the computation dynamically, and detects when there is a substantial increase (say more than 40%) in the fraction of event computations that are rolled back. When this occurs, M is reduced by 50%. This approach was chosen to force the mechanism to reach a suitable value of M in time proportional to the logarithm of the change in M that is required. M is reduced by simultaneously reducing both M_p and M_f by 50%.

3.3.4 *Putting It All Together: The Adaptive Protocol.* Since the adaptive protocol is based on monitored information, the important aspects of Time Warp behavior, e.g., the number of events processed, rolled back, cancelled, and fossil collected, are always maintained by each of the processors. Because such monitoring only involves adjusting a few local counters when certain activities occur, it incurs only a negligible overhead. However, to make appropriate memory adjustment decisions, this information needs to be compiled, and appropriate flow rates computed. The most convenient time to do this is when Cancelback is invoked, because the memory adjustment is done only at the time of Cancelback. In most situations, this is reasonable because the protocol controls the average rate of Cancelback invocation, so memory adjustment decisions can be taken sufficiently frequently. However, it is conceivable that due to a sudden change in the behavior of the application program (e.g., a sudden decrease in the number of buffers required) might cause Cancelback not to be invoked for an unexpectedly long time or even not at all! In such cases we need to "force" a Cancelback. e.g, by using a time out to invoke Cancelback at the next fossil collection, at which time memory usage is reset to an appropriate value.

Thus the adaptive protocol is assumed to be invoked just before a Cancelback (forced or natural). The protocol is described in Figure 7. In step 1 of the protocol, the action taken for a forced Cancelback is described. Here, M_p is simply reset to a value such that each processor has a small number (n) of processed uncommitted events. In our experiments a value of 4 is used for n. Since forced Cancelbacks suggest that a new phase of the computation has been entered, and there is not sufficient information concerning the new behavior to make a good decision, memory allocated to Time Warp is made very conservative and M_f is reset to n. Note that we invoke a forced Cancelback at the start of the computation.

Step 2 of the protocol attempts to detect rollback thrashing. Recall that e is the Time Warp efficiency in a cycle. Δe is the change in efficiency from the previous cycle. If efficiency is decreased by a moderate factor (l), Time Warp memory is cut down by half as explained earlier. Here, a value of 0.4 is used for l.

In step 3 T_{cbfc} denotes the Cancelback and fossil collection overhead incurred in the cycle just completed. T_{cycle} is the duration of the cycle. If the fraction (f) of time spent in this overhead is within a small range ($\Delta\delta$) of a prespecified small value (δ), no memory adjustment is attempted. Otherwise, M_f is set to a value such that T_{cycle} changes to make T_{cbfc} nearly equal to the desired fraction of T_{cycle}. As mentioned before, in all our experiments δ is chosen to be 10%. $\Delta\delta$ is chosen to be $\delta/4$. As explained before, δ specifies the amount of overhead we are willing to incur for fossil collection and Cancelback invocations. A 10% value for δ value brings the memory usage towards the knee of the performance-memory curve. This makes a very economic usage of memory, while undergoing only a modest performance loss. A lower value of δ will yield somewhat better performance, but at the expense of greater memory utilization. As discussed before, an economic use of memory is beneficial because it improves locality and reduces the likelihood that the system memory management effects significantly degrade performance. It also reduces the possibility of rollback thrashing.

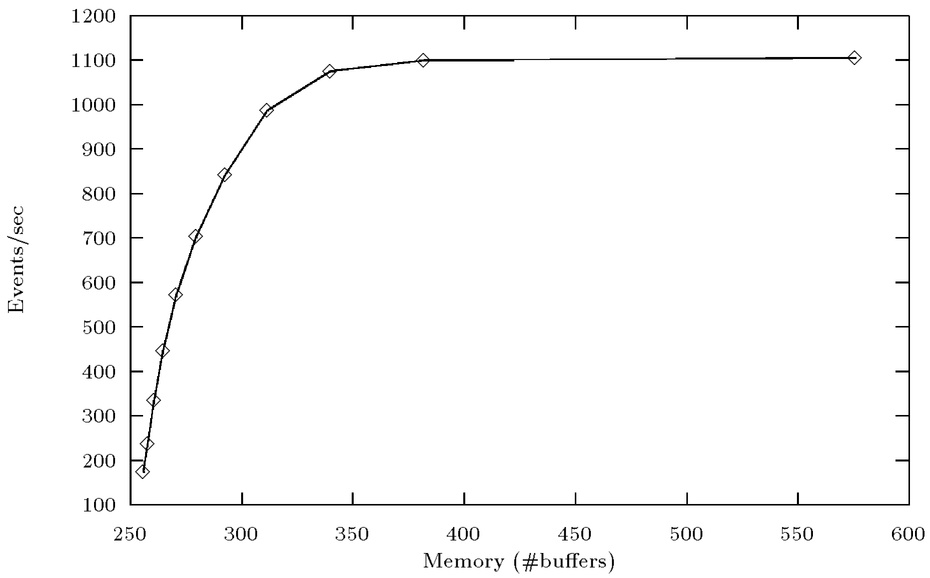
Step 4 of the protocol computes a good lower bound on M_p. It relies on an estimate G of the number of critical events rolled back by Cancelback in the past cycle. Note that G is freshly estimated in each cycle using the algorithm described in Figure 5. Ideally one would set G to be 0, but in practice it can only be made arbitrarily small and is seldom exactly zero. Thus M_p is always reduced unless G is more than a very small factor (ϵ) of the number of committed events in the past cycle. M_p is always increased or decreased by a constant factor a. In our experiments ϵ is chosen to be 0.001 and a to be 0.2.

Here, we add a note on the choice of the static parameters such as δ, $\Delta\delta$, n, l, ϵ and a. δ controls the fraction of time spent in Cancelback and fossil collection. One of the subgoals of the protocol is to allow the system to run out of memory at a controlled rate, so that a change in memory allocation can be made for the best overall performance. This rate is essentially controlled by δ and dictates how frequently memory allocation needs to be adjusted. $\delta = 0.1$ is considered to be a good compromise between the overheads and frequency of adjustments. In principle, it is possible to choose δ dynamically, however we feel that this is not necessary in the current context. $\Delta\delta$ determines the margin of the allowed deviation in the value of δ to protect against random, short-term fluctuations. In step 1, n controls only certain transient behaviors and the choice of n does not significantly affect performance for long simulation runs. In step 2, l provides a safe-guard against a sudden onset of rollback thrashing. Rollback thrashing behavior is improbable as controlled memory usage automatically controls optimism. Thus l also controls transient behavior, and the choice is not crucial for stable performance. In step 4, ϵ and a control the value of M_p. Their choice is also not crucial, so long as it keeps M_p away from its extreme values. In summary, except for δ all other parameters influence only the transient behaviors of the protocol. None of the parameters need to be tuned by the simulation modeler and can be hard-coded in the adaptive protocol.

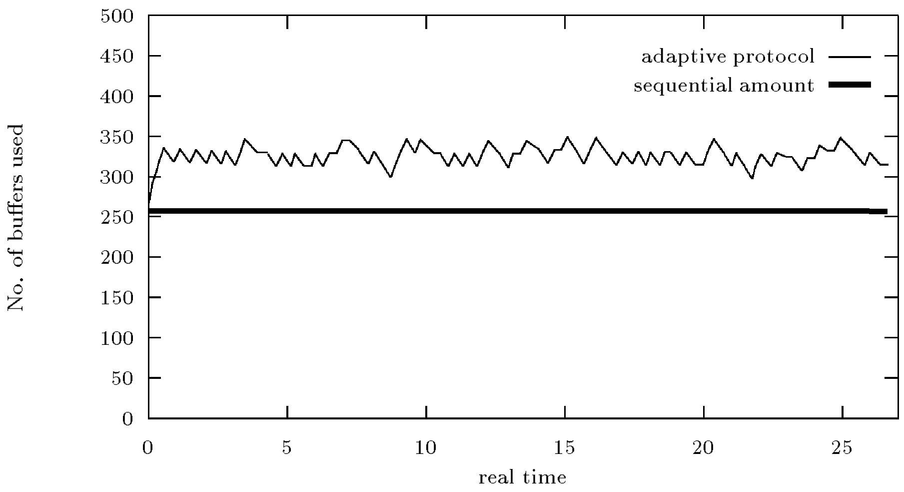
4. PERFORMANCE OF THE ADAPTIVE PROTOCOL

We implemented the adaptive protocol as described above into an existing Time Warp kernel using the Cancelback protocol [Das and Fujimoto 1993]. The kernel runs on a Kendall Square Research KSR-1 multiprocessor. The machine is a cache-only shared memory architecture (COMA) with a hardware cache coherence protocol. The Time Warp kernel works with a globally shared pool of memory buffers that grows and shrinks according to the adjustment decisions taken by the adaptive protocol. Both the PHOLD and the arbitrary flow workloads were tested with the adaptive protocol. It was found that not only the protocol automatically adapts to an appropriate value of memory, but also it is able to reach nearly (within a 10% factor) the performance of these benchmarks when the amount of memory is hand-tuned to an optimal value through extensive experimentation.

Figure 8(a) shows the performance (expressed in terms of the event rate, i.e., the number of events committed per second) vs. memory curve for the PHOLD benchmark for a non-adaptive Time Warp execution using Cancelback. Figure 8(b) shows a trace of the memory usage of the adaptive protocol, showing that the protocol rapidly adjusts memory usage to the knee of the performance-memory curve and then remains there throughout the execution. For comparison, the amount of sequential memory used by the computation is also shown. This is simply the number of present events or the number of events in the corresponding sequential simulation at the same GVT value. The performance obtained from the adaptive protocol is 1020 events/sec (a speedup of about 5.1 on 8 processors), compares favorably with the maximum performance achievable (around 1100 events/sec) with a fixed memory protocol. Figure 9 shows similar data for the arbitrary flow model. Here in Figure 9(a), we see that there is a moderate performance loss at large amounts of memory. Close monitoring of the execution revealed that it is due to a large number of cache update transactions resulting from frequent page misses in the local caches as a result of a severe loss of data locality. Such behavior is observed with most workloads when memory usage is very large [Panesar and Fujimoto 1995]. In figure 9(b) the memory trace for the adaptive protocol is shown. Note that for this workload, the sequential amount of memory varies dynamically, and accordingly there are peaks and valleys in the adaptive memory trace. For this benchmark, the adaptive protocol achieved an event rate of 2798 events/sec (compared to a rate of 3000 events/sec in the best non-adaptive execution) and operated near the knee of the performance memory curve. The event rate achieved corresponds to a speedup of about 4.2 on 10 processors. It should be emphasized that these benchmarks were designed to be stress cases for Time Warp (e.g., each processor contains only one LP). We have observed higher speedups for larger applications containing several

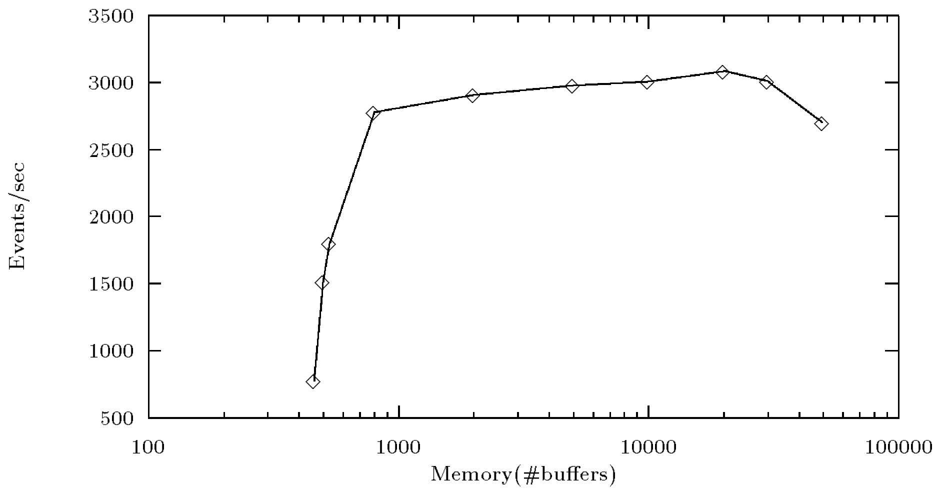


(a) Performance as a function of memory with a fixed memory protocol

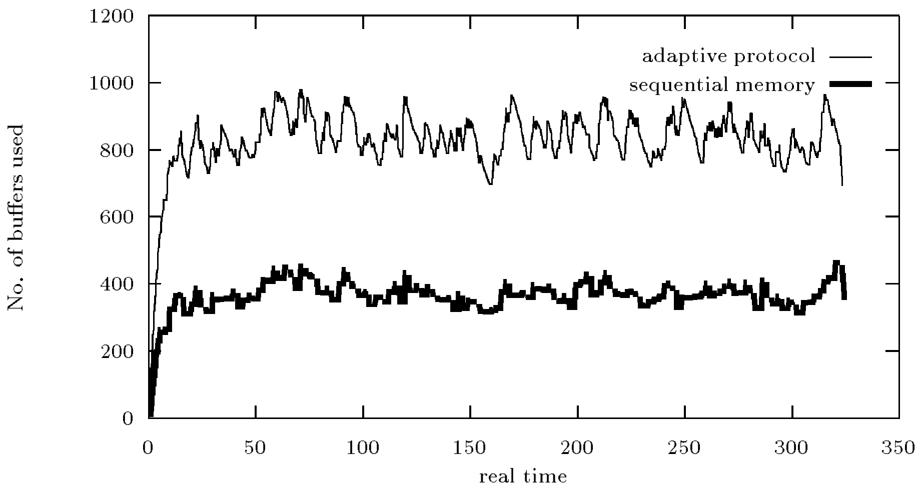


(b) Memory usage trace of the adaptive protocol

Fig. 8. The efficacy of the adaptive protocol for the PHOLD benchmark on 8 KSR-1 processors.



(a) Performance as a function of memory with a fixed memory protocol



(b) Memory usage trace of the adaptive protocol

Fig. 9. The efficacy of the adaptive protocol for the arbitrary flow benchmark on 10 KSR-1 processors.

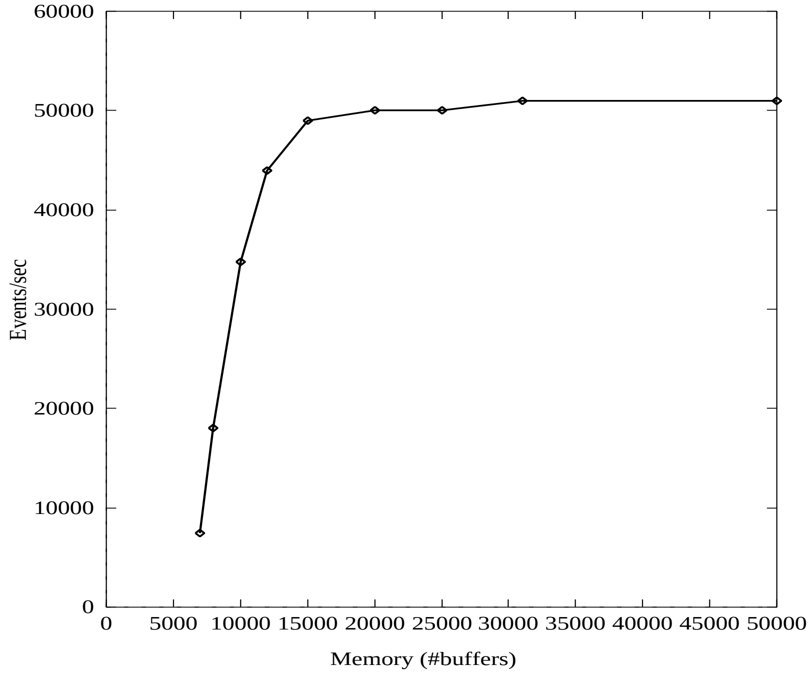


Fig. 10. Performance as a function of memory with a fixed memory protocol for the PCS network simulation benchmark on 16 KSR-1 processors.

LPs on each processor.

4.1 Benchmark Applications

Though many real simulations have properties that were exhibited in the synthetic workloads described above, a set of benchmark applications were studied to validate the behaviors observed in the synthetic workloads. The first benchmark is the simulation of a *Personal Communication Service* (PCS) network simulating a mobile communication system. The second benchmark is the simulation of a store and forward communication network in a hypercube configuration with dynamic changes in message traffic. The latter benchmark also demonstrates the adaptive ability of the protocol for a dynamically changing simulation mdoel.

A PCS (Personal Communication Services) network is a wireless communication network, which provides communication services to *mobile units* . In the simulation model, the service area of the network is partitioned into checkerboard-like subareas or *cells*. The simulation model consists of cells and portables. Portables model mobile units. Each cell represents a cellular receiver/transmitter that has some fixed number of channels allocated to it. The portable represents a mobile phone unit that resides within the cell for a period of time and then moves to one of the four neighboring cells. The behavior of a portable is modeled by different types of events, such as portable move, call arrival, and call completion. A detailed description of this application is presented in [Carothers et al. 1994].

Each cell is modeled by an LP. Each LP also models the behavior of all the

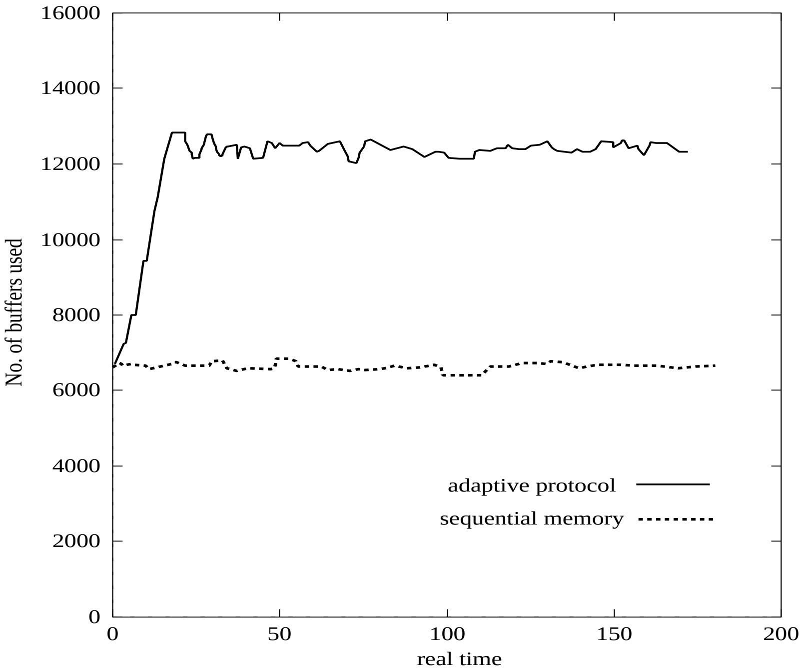


Fig. 11. Memory usage trace of the adaptive protocol for the PCS network simulation benchmark on 16 KSR-1 processors.

portables presently in the grid cell. One interesting aspect of the PCS simulation model is the *self-initiating* behavior of the LPs, i.e., that each LP often sends messages to itself to advance simulation time. In the PCS model, the portables generate their own incoming calls and this imparts the self-initiating behavior of the LPs [Carothers et al. 1994]. Another interesting aspect of the PCS model is the high degree of locality. Normally, there are a large number of grid cells in the simulation modeled by as many LPs. Since the neighboring LPs are mapped onto a single processor, there is much local communication and few remote messages. The self-initiating behavior coupled with few remote communications makes the simulation extremely asynchronous and thus there is a possibility of over-optimistic behavior. This gives rise to a very large memory usage without any form of throttling. There are, however, very few rollbacks. Figures 10 and 11 show the performance and memory usage for the PCS simulation on 16 processors. There are 16 LPs (modeling grid cells) per processor. We again observe that the adaptive protocol successfully finds the knee. The adaptive protocol simulates PCS at 45850 events/sec, while the best performance with a fixed memory protocol was found empirically to be approximately 51000 events/sec.

To evaluate the ability of the proposed protocol to adapt to a workload with an abrupt, dramatic change in behavior, we used the protocol in a simulation of a communication network. The network is a 256 node hypercube simulated on 8 processors. Messages are routed to randomly selected destination nodes using the traditional E-cube routing algorithm. Message lengths are selected from a

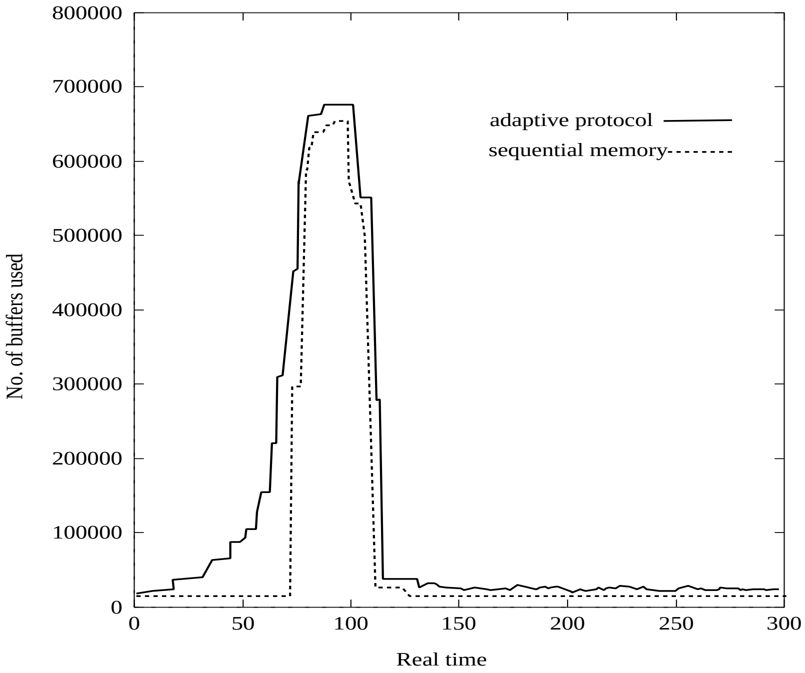


Fig. 12. Memory usage trace of the adaptive protocol for the communication network simulator application.

uniform distribution. In addition to transmission delays, there may be delays due to congestion at the nodes due to other queued messages. Messages are served by the nodes using a FCFS discipline. After a message has reached its destination, another destination is picked at random and the message is reinserted into the network. The system begins with a population of 4096 messages, and after some amount of simulated time a sudden flood of messages is introduced, as might be done to study transient congestion effects in the network. Here, the number of messages increases by a factor of 21. These additional messages are deleted from the network once they reach there selected destinations. Eventually these messages are all removed, and the network returns to its original, steady state behavior.

Figure 12 shows how the adaptive protocol automatically adjusts its memory usage to cope with the sudden message flood of messages. Note that memory usage rises for some time before the sequential memory actually rises. This is due to the fact that the optimistic execution begins simulating the message flood before the GVT actually has advanced, and they are counted in the sequential memory trace. When these messages are (optimistically) produced, there is a rapid growth of the uncommitted event pool, combined with actions taken by the adaptive protocol to limit memory usage. The protocol initially "thinks" that the flood of future events is over-optimistic, and cancels many of the flood events through Cancelback. However, as the flood persists, the protocol increases the amount of memory to accommodate it. Eventually GVT progresses past the beginning of the flood time, so the sequential amount of memory increases in proportion. As the flood dissipates, however, the protocol rapidly reduces its memory usage.

Any fixed memory protocol for this simulator must operate with a very large amount of memory to accommodate the message population during the flood. In certain situations, this could lead to rollback thrashing in the non-flood periods, though this was not observed in our experiments. When we ran a fixed memory protocol on this workload, however, it did exhibit poor performance due to caching and paging effects in the virtual memory system. Since the adaptive protocol has a very small average memory usage, these effects are largely avoided. This allows the adaptive protocol to achieve much better performance than *any* fixed memory scheme. By comparison, the adaptive protocol executes the simulation at an averaged speed of 13900 events/sec, but the best fixed memory execution achieved at most 8700 events/sec. Because of the large amount of memory required, the sequential execution is very slow (due to paging to disk) and runs only at a speed of 210 events/sec. This demonstrates that the adaptive memory protocol not only executes with an "appropriate" value of memory when confronted with a changing workload, but also can achieve better performance than its non-adaptive counterpart for time varying workloads.

5. DISCUSSION

Too much memory is detrimental to Time Warp performance primarily for two reasons. First, unthrottled optimism may generate rollback thrashing behavior for may simulation workloads. This has been demonstrated many times in the existing literature, and has been the principal motivation behind the research on limited optimism protocols. Second, random allocation of message buffers from a shared pool of memory tends to cause a loss of spatial locality, which becomes a problem if the working set is large. On the KSR architecture, careful monitoring using the event counters on the processing nodes revealed that the performance loss with large memory was due to the false sharing of virtual memory pages. In KSR's cache-only-memory protocol, a valid cache block can reside in a local cache only if the corresponding page also resides in the cache (all cache blocks in that page need not be in the valid state). On a page miss in the local cache, the page is allocated by replacing another page. Such replacement may be expensive, as the shared cache blocks owned by the local processor in the page being replaced must immediately propagate updates to the other caches that may have (invalid) copies of those blocks. Now, if the overall memory usage is small, there is a significant possibility that all pages in the working set always reside in the local cache. This reduces the chance of page misses.

Note that this behavior is different from page misses where the page must be brought from the disk, where the impact is even more significant. Paging from disks is still possible if memory usage is very large. The adaptive memory protocol automatically protects against any such excessive paging by always trying to restrict the memory usage to an appropriate level. In Time Warp, over-optimistic workloads (such as the flow networks) tend to exhaust whatever memory is available. Allocating larger memory only introduces such locality related problems. Even though the experiments are reported only for a specific COMA architecture, the general principles are still valid for other shared memory architectures, if Time Warp uses a globally shared memory pool for message buffers. In architectures with home locations for caches, similar false sharing of virtual memory pages may cause misses in the translation lookaside buffer (TLB) causing a slowdown. In the unlikely case, where such locality issues are not a problem in a specific architecture, controlling memory allocation eliminates harmful optimism in Time Warp programs by essentially providing the equivalent of a simuated time window.

It is worth remarking here that Time Warp is rather unique in its dynamics — it occupies all the available memory until garbage (fossil) collection is called, or memory is reclaimed by (for example) Cancelback. Thus the working set is determined by the characteristics of the memory management protocols in addition to the nature of the model being simulated and the load balance. Also, the memory allocation/deallocation behavior has poor spatial locality. A buffer deallocated by a processor, may be used by another processor to send a message to a third, different processor. Also, many simulation computations (such as queueing networks, communication networks, digital logic circuits) have very small granularity events which accentuates operational overheads. Note that such false sharing cannot be eliminated by simply having distributed pools of memory buffers without specific mechanisms to manage spatial locality. Some such mechanisms have been explored in [Panesar and Fujimoto 1995]. However, this gives rise to other unresolved issues, for example, determining the appropriate sizes of individual pools with the constraint that the total size is limited by the size of the available shared memory. Also, it is very difficult to determine such sizes statically. Thus dynamic exchange of memory between pools is necessary. Such exchange may use the protocol developed in this work.

6. CONCLUSIONS

We have examined the use of memory as a mechanism to control optimism in Time Warp. Uncontrolled optimism may have a detrimental effect on performance in systems with large rollback and cancellation overheads, where time spent in these overhead computations will allow other overly optimistic and possibly incorrect computations to execute when they should not, resulting in a thrashing effect. Uncontrolled memory usage may also result in a large amount of memory utilized by the Time Warp program (whose memory requirement, in principle, can be unbounded), which can have a negative impact on the performance owing to the loss of spatial locality.

Our adaptive protocol offers a unified solution to both the problems. It attempts to reduce unnecessary optimism by economizing on memory usage and without undergoing any significant protocol related overhead. The protocol controls memory usage by allowing the free pool of memory to "drain" at a controlled rate. We show that this rate can be predicted using a simple "buffer flow model" if certain information about the behavior of Time Warp can be dynamically measured. Similarly, the protocol controls optimism by rolling back prospective off-critical path events. The adaptive protocol is evaluated using symmetric and asymmetric workloads, modeling a closed and an open system respectively. It is observed that the protocol can rapidly adapt to a value of memory close to the knee of the performance-memory curve. Experiments with a time varying workload shows that the protocol automatically adjusts memory dynamically depending on the memory usage behavior of the underlying simulation model, and may achieve better performance than any fixed memory protocol.

APPENDIX

Let us represent the average number of fossil collection calls in a cycle by F, and the expected number of buffers fossil collected in the i-th fossil collection attempt in a typical cycle by B_i. The rate at which the free pool is depleting is given by eP/T_{ev}. Thus the first fossil collection in the cycle will be attempted after a time $M_f/(eP/T_{ev})$. Recall that we denote the average fossil collection rate by R_{fc}. Thus, the initial size of the free pool is M_f, the number of buffers fossil collected at the first attempt is given by

$$B_1 = M_f R_{fc}/(eP/T_{ev}) = kM_f,$$

where $k = R_{fc}/(eP/T_{ev})$ and is assumed to be constant. For the second fossil collection, the initial size of the free pool is B_1, for the third it is B_2 and so on. Thus $B_{i+1} = kB_i$, for $i \geq 1$. This gives

$$B_F = k^F M_f.$$

Now B_F is expected to be zero in practice signaling the end of cycle. This, however, is impossible assuming a constant value for k, which is an approximation because of the random nature of all buffer flow rates that constitute k. Thus we approximate B_F by 1. This gives

$$F = \log_k(1/M_f) = -\log_{10}(M_f)/\log_{10}(k). \tag{4}$$

Note that $k < 1$, because the total number of events fossil collected is expected to be less than the total number of events processed in any measurement interval. This may not be true in general for Time Warp executions, but is true under memory constraints, when the free memory pool always tends to deplete.

Thus the expected number of fossil collection calls in a cycle (F) grows logarithmically with the initial size of the free pool (M_f). This is a very slow growth when compared to the fact that the total number of cycles (N) varies almost inversely as M_f (see equation (2), N is inversely proportional to T_{cycle}). As the total number of fossil collection calls is equal to the product NF, the total number is very sensitive to the effect of M_f on N rather than the same on F. This validates our assumption that controlling N (or T_{cycle}) indirectly controls fossil collection frequency by approximately a proportionate amount, suggesting that a separate mechanism to control fossil collection frequency is not necessary.

We also ran a set of experiments on the arbitrary flow network benchmark to assess the validity of the logarithmic variation of F with M_f, as modeled by equation 4. We have chosen the flow model to validate this behavior, because of its large memory requirement. It is possible to operate this model under memory constraints with a wide range of M_f values. Note that this logarithmic behavior model has an underlying assumption that the system is operated under memory constraints.

Figure 13 shows the actual and predicted behavior. We see that they are under close agreement. We also note that the actual behavior starts deviating from the predicted behavior for very large values of M_f (extreme end of the plot) as our assumption about constrained memory execution becomes less valid.

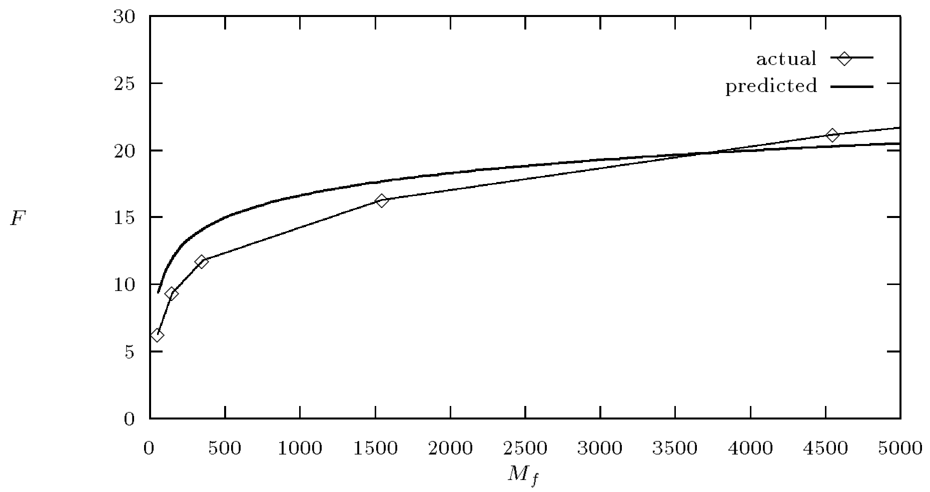


Fig. 13. Variation of the average number of fossil collection calls (F) in a cycle with initial value of free pool (M_f). The workload used is the arbitrary flow model.

REFERENCES

AKYILDIZ, I. F., CHEN, L., DAS, S. R., FUJIMOTO, R. M., AND SERFOZO, R. F. 1993. The effect of memory capacity on Time Warp performance. *Journal of Parallel and Distributed Computing 18*, 4 (August), 411–422.

ARVIND, K. AND SMART, C. 1992. Hierarchical parallel discrete event simulation in composite ELSA. In 6^{th} *Workshop on Parallel and Distributed Simulation* (January 1992), pp. 147–158.

BALL, D. AND HOYT, S. 1990. The adaptive Time-Warp concurrency control algorithm. *Proceedings of the SCS Multiconference on Distributed Simulation 22*, 1 (January), 174–177.

BRINER, J., JR. 1991. Fast parallel simulation of digital systems. *Proceedings of the Multiconference on Advances in Parallel and Distributed Simulation 23*, 1 (January), 71–77.

BROWN, R. 1988. Calendar queues: A fast $O(1)$ priority queue implementation for the simulation event set problem. *Communications of the ACM 31*, 10 (October), 1220–1227.

CAROTHERS, C. D., FUJIMOTO, R. M., LIN, Y.-B., AND ENGLAND, P. 1994. Distributed simulation of large scale PCS networks. In *Proceedings of the 2nd International Conference on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'94)* (Jan 1994), pp. 2–11.

CLEARY, J., GOMES, F., UNGER, B., ZHONGE, X., AND THUDT, R. 1994. Cost of state saving and rollback. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation* (1994), pp. 94–101.

DAS, S. R., FUJIMOTO, R., PANESAR, K., ALLISON, D., AND HYBINETTE, M. 1994. GTW: A Time Warp system for shared memory multiprocessors. In *1994 Winter Simulation Conference Proceedings* (December 1994), pp. 1332–1339.

DAS, S. R. AND FUJIMOTO, R. M. 1993. A performance study of the cancelback protocol for Time Warp. *Proceedings of the 7th Workshop on Parallel and Distributed Simulation 23*, 1 (May), 135–142.

Das, S. R. and Fujimoto, R. M. 1997. An empirical evaluation of performance-memory trade-offs in Time Warp. *IEEE Transactions on Parallel and Distributed Systems 8*, 2 (February). To appear.

Dickens, P. M. and Reynolds, Jr., P. F. 1990. SRADS with local rollback. *Proceedings of the SCS Multiconference on Distributed Simulation 22*, 1 (January), 161–164.

Ferscha, A. 1995. Probabilistic adaptive direct optimism control in Time Warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation* (1995), pp. 120–129.

Fleischmann, J. and Wilsey, P. A. 1995. Comparative analysis of periodic state saving techniques in Time Warp simulators. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation* (1995), pp. 50–58.

Fujimoto, R. M. 1989. Time Warp on a shared memory multiprocessor. *Transactions of the Society for Computer Simulation 6*, 3 (July), 211–239.

Fujimoto, R. M. 1990a. Parallel discrete event simulation. *Communications of the ACM 33*, 10 (October), 30–53.

Fujimoto, R. M. 1990b. Performance of Time Warp under synthetic workloads. *Proceedings of the SCS Multiconference on Distributed Simulation 22*, 1 (January), 23–28.

Hamnes, D. O. and Tripathi, A. 1994a. Evaluation of a local adaptive protocol for distributed discrete event simulation. In *Proceedings of the 1994 International Conference on Parallel Processing* (August 1994), pp. III:127–134.

Hamnes, D. O. and Tripathi, A. 1994b. Investigations in adaptive distributed simulation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation* (1994), pp. 20–23.

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

Jefferson, D. R. 1990. Virtual time II: The cancelback protocol for storage management in distributed simulation. In *Proc. 9th Annual ACM Symposium on Principles of Distributed Computing* (Aug. 1990), pp. 75–90.

Jefferson, D. R. and Motro, A. 1986. The Time Warp mechanism for database concurrency control. *Proceedings of the 2nd International Conference on Data Engineering*, 141–150.

Lin, Y.-B. and Preiss, B. R. 1991. Optimal memory management for Time Warp parallel simulation. *ACM Transactions on Modeling and Computer Simulation 1*, 4 (October), 283–307.

Lin, Y.-B., Preiss, B. R., Loucks, W. M., and Lazowska, E. D. 1993. Selecting the checkpoint interval in Time Warp simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation* (May 1993), pp. 3–10.

Lubachevsky, B. D., Shwartz, A., and Weiss, A. 1989. Rollback sometimes works ... if filtered. In *1989 Winter Simulation Conference Proceedings* (December 1989), pp. 630–639.

Lubachevsky, B. D., Shwartz, A., and Weiss, A. 1991. An analysis of rollback-based simulation. *ACM Transaction on Modeling and Computer Simulation 1*, 2 (April), 154–193.

Madisetti, V. K., Hardaker, D. A., and Fujimoto, R. M. 1993. The MIMDIX operating system for parallel simulation and supercomputing. *Journal of Parallel and Distributed Computing 18*, 4 (August), 473–483.

Mehl, H. 1991. Speedup of conservative distributed discrete-event simulation methods by speculative computing. *Proceedings of the Multiconference on Advances in Parallel and Distributed Simulation 23*, 1 (January), 163–166.

Nicol, D. M. and Fujimoto, R. M. 1994. Parallel simulation today. *Annals of Operations Research 53*, 249–286.

Palaniswamy, A. C. and Wilsey, P. A. 1993. An analytical comparison of periodic checkpointing and incremental state saving. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation* (May 1993), pp. 127–134.

Panesar, K. and Fujimoto, R. M. 1995. Buffer management in shared memory time warp systems. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation* (1995), pp. 149–156.

PRESLEY, M., EBLING, M., WIELAND, F., AND JEFFERSON, D. R. 1989. Benchmarking the Time Warp Operating System with a computer network simulation. *Proceedings of the SCS Multiconference on Distributed Simulation 21*, 2 (March), 8–13.

RAJAEI, H., AYANI, R., AND THORELLI, L.-E. 1993. The local Time Warp approach to parallel simulation. In *Proceedings of the* 7^{th} *Workshop on Parallel and Distributed Simulation* (May 1993), pp. 119–126.

REIHER, P. L., WIELAND, F., AND JEFFERSON, D. R. 1989. Limitation of optimism in the Time Warp Operating System. In *Proceedings of the 1989 Winter Simulation Conference* (December 1989), pp. 765–770.

RÖNNGREN, R., AYANI, R., FUJIMOTO, R. M., AND DAS, S. R. 1993. Efficient implementation of event sets in Time Warp. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation* (May 1993), pp. 101–108.

SOKOL, L. M. AND STUCKY, B. K. 1990. MTW: experimental results for a constrained optimistic scheduling paradigm. *Proceedings of the SCS Multiconference on Distributed Simulation 22*, 1 (January), 169–173.

SRINIVASAN, S. AND REYNOLDS, P. F. 1995. NPSI adaptive synchronization algorithms for PDES. In *1995 Winter Simulation Proceedings* (December 1995), pp. 658–665.

STEINMAN, J. 1992. SPEEDES: A multiple-synchronization environment for parallel discrete event simulation. *International Journal of Computer Simulation 2*, 3, 251–286.

STEINMAN, J. S. 1993. Breathing Time Warp. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation* (May 1993), pp. 109–118.

TURNER, S. J. AND XU, M. Q. 1992. Performance evaluation of the bounded Time Warp algorithm. *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation 24*, 3 (January), 117–126.

WIELAND, F., HAWLEY, L., FEINBERG, A., DILORENTO, M., BLUME, L., REIHER, P., BECKMAN, B., HONTALAS, P., BELLENOT, S., AND JEFFERSON, D. R. 1989. Distributed combat simulation and Time Warp: The model and its performance. *Proceedings of the SCS Multiconference on Distributed Simulation 21*, 2 (March), 14–20.