Performance Analysis of "Time Warp" with Limited Memory

Ian F. Akyildiz†, Liang Chen‡, Samir R. Das†, Richard M. Fujimoto†, Richard F. Serfozo‡

† College of Computing
‡ School of Industrial and Systems Engineering
Georgia Institute of Technology
Atlanta, Georgia 30332

Abstract

The behavior of n interacting processes synchronized by the "Time Warp" rollback mechanism is analyzed under the constraint that the total amount of memory to execute the program is limited. In Time Warp, a protocol called "cancelback" has been proposed to reclaim storage when the system runs out of memory. A discrete state, continuous time Markov chain model for Time Warp augmented with the cancelback protocol is developed for a shared memory system with n homogeneous processors and homogeneous workload. The model allows one to predict speedup as the amount of available memory is varied. To our knowledge, this is the first model to achieve this result. The performance predicted by the model is validated through direct performance measurements on an operational Time Warp system executing on a shared-memory multiprocessor using a workload similar to that in the model. It is observed that Time Warp with only a few additional message buffers per processor over that required in the corresponding sequential execution can achieve approximately the same or even greater performance than Time Warp with unlimited memory, if GVT computation and fossil collection can be efficiently implemented.

1 Introduction

The Time Warp mechanism has been proposed as a general technique for synchronizing asynchronous parallel computations [8]. Unlike conventional, so-called *conservative*, approaches to synchronization that utilize blocking to avoid the possibility of synchronization errors, Time Warp uses a mechanism that detects errors at runtime, and automatically recovers using rollback. Empirical studies have reported some success in speeding up the execution of discrete-event simulation applications using Time Warp [4].

A Time Warp program consists of a collection of *logical processes* (LPs) that communicate by exchanging timestamped event messages.[1] We assume timestamps are unique, real, values that are totally ordered by the relation "<". All computations are the result of processing messages, i.e., processes do not "spontaneously" begin new computations. Each process maintains a local *clock* variable that indicates the timestamp of the message now being processed, or the last message it processed if the LP is idle. Messages must be processed in non-decreasing timestamp order. If a message with timestamp T is received "in the past," the computations associated with messages with timestamps larger than T must be rolled back. This may involve sending *antimessages* to cancel previously sent messages, which in turn may induce additional rollbacks. Details of the mechanism are described in [8]. We will assume that the reader is familiar with the definitions and basic terminology described in that work.

A substantial amount of effort has been devoted to developing analytic models to characterize the behavior of Time Warp. Several models have been developed to model execution on two processors [2, 11, 19, 21]. Recently some work has also attacked the n processor case [7, 10, 18]. Bounds have been derived to compare Time Warp's performance with that of other approaches [1, 13, 15, 20], and conditions have been identified under which an excessive number of rollbacks may

[1] We use the terms *event* and *message* synonymously.

1992 ACM SIGMETRICS & PERFORMANCE '92-6/92/R.I., USA
© 1992 ACM 0-89791-508-9/92/0005/0213...$1.50

occur [17].

All prior work in modeling Time Warp has assumed that there are no constraints on the amount of memory that is available to execute the program. In effect, these models ignore the *message sendback* aspect of Time Warp that was proposed by Jefferson to implement flow control [8]. Several extensions of message sendback have since been proposed in the literature. As discussed later, message sendback based protocols are often used when the simulation runs out of memory and the normal garbage collection procedure (called *fossil collection*) fails to recover additional storage. These protocols roll back some processes that are ahead in virtual time relative to others in order to reclaim memory and enable the simulation to progress. While Time Warp models that ignore message sendback can still yield accurate predictions when there is adequate memory, substantial deviations could arise when memory is limited. Moreover, little is known concerning the performance of Time Warp with limited memory. This is the central issue that we address here.

In a prior work we discussed the performance of Time Warp with unlimited memory [7]. Although many of the assumptions used here are similar to those in that work, substantial changes have been made to model the limited memory situation. Furthermore, the solution to the present model differs significantly from the earlier one.

This paper is organized as follows: in section 2 we briefly overview different memory management schemes for Time Warp and discuss the scheme that we are analyzing. In section 3 we describe the model. The analysis of this model is discussed in section 4. In section 5 we compare predictions made by the model with experimental measurements. In section 6 we conclude and discuss directions for future research.

2 Memory Management in Time Warp

Time Warp consumes memory by storing three types of *objects*, viz., state vectors in the state queue, positive messages in the input queue, and negative (anti-) messages in the output queue. Several mechanisms can be used in Time Warp to control the amount of memory. Various approaches are enumerated below:

- *Fossil collection:* Storage used by objects that are older than *global virtual time* (GVT)[2] can be reclaimed and used for other purposes.
- *Message sendback and its extensions:* If fossil collection fails to recover any storage, but additional memory is needed, some other mechanism is required. Jefferson's message sendback protocol is one such approach [8]. Several variations and/or extensions of message sendback have also been proposed, as outlined below.
- *Infrequent state saving:* State vectors can be saved at a less frequent rate to reduce storage consumed by the state queue [14, 22].
- *Limiting optimism:* Variations on Time Warp that limit the degree to which processes can advance ahead of others implicitly reduce the amount of memory that is required [16, 18, 23, 24].

Several policies utilizing message sendback have been proposed. Jefferson's original proposal invokes message sendback when a process receives a message, but finds that there is no memory available to store it [8]. Then the message with the largest send timestamp in its input queue is returned to its sender. The sender rolls back on receipt of this message if its local virtual time is more than the send timestamp of the returned message, and possibly resends it in a later forward execution phase. Gafni's protocol [6] generalizes message sendback by removing any stored object (input message, state vector or output message) from the process P that runs out of memory. If the discarded object is an input message, it is returned to the sender, as in message sendback. If it is an output message, it is transmitted to its receiver where it will cancel the corresponding positive message, and P rolls back to the state before it sent the original positive message. If the stored object is a state, it is discarded. Typically, the object with the highest sendtime is selected for removal.

Jefferson proposed an alternative approach called *cancelback* [9] specifically targeted for shared memory systems, where there is a single shared pool of memory. In this protocol, if a process P needs storage for any object u, it is assumed that u is always allocated, but after allocation there may not be any free memory to continue the simulation. The protocol then discards a stored object from some process (not necessarily the same process that stores u) exactly as in Gafni's protocol to free memory. The discarded object must have its sendtime greater than GVT.

Lin observed that all these protocols rely on some process rolling back to free memory [12]; so the ability to *artificially* roll back any process to an earlier virtual time will be a simple, but efficient, memory management scheme. He described the artificial rollback protocol in connection with a shared memory architecture

[2] GVT is defined as a lower bound on the timestamps of all future rollbacks. For operational definitions of GVT see [8, 12].

with a shared memory pool. If any process runs out of memory, and fossil collection fails to reclaim enough storage, the process farthest ahead in virtual time is rolled back. How far to roll back is a parameter of the scheme. For efficiency reasons Lin recommends rolling back the process with the latest local clock to the second latest local clock. This continues until enough storage has been reclaimed.

Artificial rollback is semantically equivalent to cancelback. The syntactic difference, however, makes it somewhat easier to implement in most systems (there is no need to distinguish between messages in forward and reverse transit).

Here, we are concerned with the performance of Time Warp with cancelback. Cancelback is believed to be a complete solution for the Time Warp memory management problem, as it enables Time Warp to complete the simulation with the same amount of memory as the corresponding sequential execution [9].

3 Model Description

We assume that the Time Warp program is partitioned into n processes, each of which executes on a separate processor. All processors and processes are identical. This assumption of homogeneity is perhaps the strongest assumption made by the model. Since each process executes on a distinct processor we use the words "process" and "processor" interchangeably. We also assume that there are initially m unprocessed messages in the system, and that upon processing each message, exactly one new message is generated. The quantity m is referred to as the *message population*. This fixed message population assumption holds exactly for simulations such as closed queueing networks, and approximately for many other simulations where the size of the event list (in sequential execution) does not vary substantially throughout the simulation. The computation time associated with each message is assumed to be exponentially distributed with rate λ.

We assume rollback is non-preemptive; if a message in the past is received while an event is being processed, the rollback does not take effect until the processing of the current event is finished. We assume the time for rollback is negligible. We also assume that the process's state is saved prior to processing each event, and the time to save state is negligible. These latter assumptions are mild for medium to large grain simulations (e.g., many combat models), where the associated overheads are small relative to the computation time per event.

An event whose timestamp is less than GVT is called a *committed message*. Committed messages cannot be rolled back. Others are referred to as *uncommitted messages*. These may or may not have been processed previously and, with the exception of the message with timestamp equal to GVT, could later be rolled back.

We use a shared-memory model for memory allocation, i.e., we assume unallocated memory is stored in a global, shared, pool of buffers. Each message includes the associated state vector (recall that state is saved before *each* message is processed) and is assumed to require one memory buffer. M denotes the total number of memory buffers available in the system. We assume no additional buffers to hold messages in transit. When a message is sent, the sender requests and obtains a memory buffer (if one is available), fills it in, and puts the buffer directly in the receiving process's input queue. Any buffer freed by a process (e.g., via fossil collection) is returned to the shared pool. We assume cancellation is done by a pointer traversal as in *direct cancellation* [3] rather than through antimessages. As demonstrated in [3], each pointer consumes a constant amount of space for each positive message in the system, so they are included as part of the memory buffer. Thus, no additional buffers are required to hold antimessages.

The total number of messages in the system cannot exceed the memory capacity M. We assume that fossil collection is instantaneous and runs continuously on the background so that any fossil is immediately reclaimed. Thus all messages in the system correspond to uncommitted events. When there are no free message buffers in the system and process P_s attempts to send a message to process P_r, the Time Warp system returns the message u with the largest send timestamp back to the process P_u that originally sent it. This causes P_u to roll back to the last state prior to sending u, and the memory buffer holding u is freed. The only exception occurs when the local time of P_s is greater than the sendtime of u. In this case, P_s aborts the send, and rolls back the current event.

The processing of a message with timestamp τ involves the following operations:

i) Read the contents of the message.

ii) Compute and update state variables.

iii) Send a new message to one of the n processors (including possibly itself) chosen from a uniform distribution. The timestamp of the new message is $\tau + \zeta$, where ζ is an exponentially distributed random variable with rate μ.

The assumptions regarding the choice of message recipient and the timestamp increment may not be true

for many systems, but are necessary to make the analysis tractable. Empirical evidence suggests that the message routing function, computation time, and timestamp distributions have a secondary effect on performance for homogeneous applications [5].

A memory buffer needs to be allocated from the shared pool to hold the message to be sent. If none is free, the system already contains M uncommitted messages and the memory management protocol described above is invoked to free a buffer. We assume that the time to send a message, whether or not the protocol is invoked, is negligible. As noted earlier, this is consistent with the situation where computation grain is significantly larger than the associated overheads.

As a processor completes processing the message with timestamp τ, it looks for the lowest timestamped unprocessed message (or antimessage) in its input queue. Suppose the resulting timestamp is τ'. If $\tau' \leq \tau$, the process rolls back to the most recent state earlier than τ'. The process then (whether or not it rolls back) (i) sets its local clock to τ', (ii) copies the state vector from the most recent message with timestamp less than τ', and (iii) processes the message with timestamp τ' as described before. If the message with timestamp τ' is an antimessage, however, it annihilates the corresponding positive message (if it is already present in the queue), or is processed as a no-op. The time taken to do these operations in between processing of two messages is assumed to be negligible. It is assumed that there is always at least one unprocessed message in each processor's input queue, i.e., the message population is substantially larger than the number of processors n. This implies that all processors busy all the time. This is a reasonable assumption for simulation models that are much larger than the multiprocessor configuration.

Note that a change in the system occurs only when a processor completes processing a message and the above actions are taken. The number of processed uncommitted messages in the system may increase or decrease depending on whether or not there is a rollback, or whether the GVT advances (thus committing some messages) after processing the next message. The number of unprocessed messages is always constant and is equal to the message population m. Thus the total number of processed uncommitted messages cannot exceed $(M - m)$. It follows that M must at least be m for the simulation to complete. Our objective is to determine the effect of memory capacity M on the speed of the system.

4 Model Analysis

A complete Markov modeling of the system would entail keeping track of all the messages in the system, i.e., recording their timestamps, locations and whether they are processed or not. Since this is impractical we take another approach. The key idea in our approach centers on the assumptions that the processors are identical and the workload is homogeneous. We use these assumptions to extrapolate or generate information about the messages knowing only their quantities. For instance, each processed message in the system has probability $1/n$ of being located at a particular processor. This is valid in equilibrium and hence is a reasonable assumption for non-equilibrium states when one has little information containing the past.

Recall that in our model the timestamp increment (difference of the receive and send timestamps of a message and its source message) is exponentially distributed with rate μ. This assumption implies that the interdistance in virtual time of the processed uncommitted messages in any process is also exponentially distributed with rate α (say), where α may differ from μ (See [7] for an explanation). Similarly, we assume that the distribution of timestamp differences is also exponential for the unprocessed messages with rate $\gamma = m\mu/n$, and for antimessages is exponential with rate β. We use these assumptions to model the distribution in virtual time of the processed and unprocessed messages at a processor in order to determine the number of processed messages that must be undone in a rollback. Aside from this relative ordering of the virtual timestamps, their actual values are not important.

4.1 Equilibrium State Probabilities

We will represent the system by the continuous-time stochastic process $\{X(t) : t \geq 0\}$, where $X(t)$ denotes the total number of processed but uncommitted events in the system at real time t. Note that $X(t)$ does not include messages currently being processed (which are partially processed messages). Recall that $X(t)$ can be at most $(M - m)$, where M is the memory capacity in number of message buffers, and m is the message population. Information concerning the locations and types (processed or unprocessed) of messages will be generated from $X(t)$.

Under our assumptions, the process X is an irreducible Markov chain with state space $S = \{0, 1, ..., (M - m)\}$. The evolution of X is characterized by its transition rates

$$q_{jk} = \lim_{t \downarrow 0} t^{-1} P\{X(t) = k | X(0) = j\}, \quad j \neq k \in S, \quad (1)$$

The process X is irreducible and its equilibrium distribution π is the solution to the balance equations

$$\pi_j \sum_{k \neq j} q_{jk} = \sum_{k \neq j} \pi_k q_{kj}, \quad j \in S. \tag{2}$$

Under our assumption that all processors are always busy, the exponentially distributed time that the process X remains in any state is $n\lambda$, where λ is the rate of the exponential processing time of each processor. Then $q_{jk} = n\lambda P_{jk}$, where P_{jk} is the probability that X moves from state j to state k at a transition. Therefore the balance equations (2) simplify to

$$\pi_j = \sum_k \pi_k P_{kj}, \quad j \in S. \tag{3}$$

The following subsections describe the transition probabilities P_{jk} and a computational procedure for obtaining the equilibrium probabilities π_j from (3).

4.2 Expressions for Transition Probabilities

The transitions of the process X involve the following events:

Rollback : $\mathcal{R}_k = \{X$ moves to state k due to a rollback $\}$

GVT advance : $\mathcal{D}_k = \{X$ decreases to state k due to a GVT advance $\}$

Unit increase : $\mathcal{I} = \{X$ increases by one unit $\}$

The nonzero transition probabilities of the process X are therefore

$$P_{j\,j+1} = P_j\{\mathcal{I}\}, \qquad j, j+1 \in S. \tag{4}$$

$$P_{jk} = P_j\{\mathcal{R}_k\} + P_j\{\mathcal{D}_k\}, \quad 0 \leq k \leq j, \; j \in S. \tag{5}$$

$$P_{jk} = P_j\{\mathcal{R}_k\} + P_j\{\mathcal{D}_k\} + P_j\{\mathcal{I}\}, \quad j = k = M - m. \tag{6}$$

Here $P_j\{.\}$ is the conditional probability given $X(0) = j$. We will derive expressions for these probabilities in the following subsections. We make frequent reference to the following random variables associated with processor i at a transition:

X_i is the number of processed uncommitted messages (in processor i).

Y_i is the number of unprocessed messages in the local past.

Z_i is the number of antimessages in the local past.

$U_i = Y_i + Z_i$ is the number of messages in the local past. These are the messages that cause rollback in processor i.

4.2.1 Probability Distribution Messages in the Local Past

Our expressions for the transition probabilities will involve the following conditional distribution for the number of past messages U_i at processor i. Conditioning on Z_i,

$$P\{U_i = u | X_i = x\} = \sum_{z=\max(0,u-m+n)}^{\min(u,x)} P\{Z_i = z | X_i = x\} P\{Y_i = u - z | X_i = x\}, \quad u = 0, \ldots, x + m - n. \tag{7}$$

Here Z_i and Y_i are conditionally independent given $X_i = x$, and U_i cannot exceed $(x + m - n)$ since at most x antimessages may arrive at the processor being rolled back to cancel the processed uncommitted events and at most $m - n$ unprocessed events may arrive at that processor causing a rollback. We can write

$$P\{Z_i = z | X_i = x\} = h(z) / \sum_{l=0}^{x} h(l), \quad z = 0, \ldots, x, \tag{8}$$

where

$$\begin{aligned} h(z) &= P\{Z_i = z, X_i = x\} \\ &= \int_0^\infty e^{-\beta s} \frac{(\beta s)^z}{z!} \frac{\alpha(\alpha s)^x}{x!} e^{-\alpha s} ds \\ &= \binom{x+z}{z} (r_\beta)^z / (1 + r_\beta)^{z+x+1}, \quad z = 0, \ldots, x \end{aligned} \tag{9}$$

and $r_\beta = \beta/\alpha$. The integral follows since differences between the timestamps of the $X_i = x$ messages are exponential with rate α and the differences between the timestamps of the $Z_i = z$ antimessages are exponential with rate β. Thus, the first probability in the sum (7) is evaluated by (8) and (9). The last probability in (7) is expressed as

$$P\{Y_i = y | X_i = x\} = \sum_{w=y}^{m-n} P\{Y_i = y | X_i = x, W_i = w\} P\{W_i = w | X_i = x\}, \quad y = 0, \ldots, m - n, \tag{10}$$

where W_i is the number of unprocessed messages in processor i. As the processors are always assumed to be busy, there are always m unprocessed events in the system and that each of the n processors contains at least one of them. The rest of $(m-n)$ unprocessed messages are equally likely to be located at the processors, and so

$$P\{W_i = w | X_i = x\} = \binom{m-n}{w}\left(\frac{1}{n}\right)^w\left(1-\frac{1}{n}\right)^{m-n-w}, \quad (11)$$
$$w = 0, \ldots, m-n.$$

Consequently, W_i is independent of X_i. Then, similarly to (8), (9),

$$P\{Y_i = y | X_i = x, W_i = w\} = \frac{g(y)}{\sum_{l=0}^{w} g(l)}, \quad (12)$$
$$y = 0, \ldots, w,$$

where

$$g(y) = P\{Y_i = y, X_i = x\} = \binom{x+y}{y}(r_\gamma)^y/(1+r_\gamma)^{y+x+1} \quad (13)$$

and $r_\gamma = \gamma/\alpha$. Thus, the last probabilities in (7) are obtained by (10)-(13). This completes our evaluation of (7).

4.2.2 Unit Increase Consider the event $\mathcal{I}$ that there is a one-step increase in the process X. Let I denote the processor that advances and initiates the transition. Note that I is not the GVT regulator [7]. (GVT regulator is the processor that has the GVT event in its input queue. The GVT event is the uncommitted event with the minimum timestamp among all such events.) This is because whenever the regulator advances it immediately commits the event it has just processed. Since all processors are identical, we may assume the nth processor is the regulator at a transition, and let $P_j\{.\}$ denote the conditional probability under this additional condition. Then by standard conditioning,

$$P_j\{\mathcal{I}\} = \sum_{i=1}^{n-1}\sum_{x=0}^{j} P_j\{\mathcal{I}, I = i | X_i = x\} P_j\{X_i = x\}, \ j \in S. \quad (14)$$

Whenever $X = j$, we assume that these j processed uncommitted events are independently located at processors $1, \ldots, n-1$ (no processed uncommitted event can be in processor n, which is the regulator), and the probability of one being at processor $i \neq n$ is $1/(n-1)$. Then

$$P_j\{X_i = x\} = \binom{j}{x}\left(\frac{1}{n-1}\right)^x\left(1-\frac{1}{n-1}\right)^{j-x}, \quad x = 0, \ldots, j. \quad (15)$$

Also, the process X can increase by one unit at processor $I = i$ if and only if $U_i = 0$ (there are no past messages or antimessages at i). Then

$$P_j\{\mathcal{I}, I = i | X_i = x\} = n^{-1} P\{U_i = 0 | X_i = x\}, \quad (16)$$

where n^{-1} is the probability that processor i is the first of the n processors to finish their processing. The last probability in (16) is given by expression (7). Thus, $P_j\{\mathcal{I}\}$ can be computed from (14)-(16) and (7).

4.2.3 Rollback Consider the event $\mathcal{R}_k$ that process X moves to state k as a result of a rollback. As above, assume that processor n is the regulator at the transition and that $P_j\{.\}$ is conditioned on this event. Let I denote the processor that finishes processing first and triggers the transition. If $I = n$ and a rollback occurs, then the process X can move only from j back to j. Then conditioning on X_i and I, we have

$$P_j\{\mathcal{R}_k\} = \sum_{i=1}^{n-1}\sum_{x=j-k}^{j} P_j\{X_i = x\} P_j\{I = i | X_i = x\} . P_j\{\mathcal{R}_k | X_i = x, I = i\} + 1_{(k=j)}\, P_j\{\mathcal{R}_j, I = n, X_n = 0\}, \quad (17)$$
$$k = 0, \ldots, j.$$

Here, $1_{(k=j)} = 1$ if $k = j$, and 0 otherwise. The last probability involving $\mathcal{R}_j$ is evaluated by (33) in the next subsection. In the summation of (17), the first probability is given by (15) and the second probability is

$$n^{-1}[1 - P\{U_i = 0 | X_i = x\}],$$

which is evaluated by (7). Here n^{-1} is the probability that processor i finishes before the other $n-1$ processors and this processor can rollback if and only if $U_i \neq 0$.

The third probability in the sum (17) (conditioned on U_i) is

$$P_j\{\mathcal{R}_k | X_i = x, I = i\} = \sum_{u=1}^{x+m-n} P_j\{\mathcal{R}_k | C_i\} P\{U_i = u | X_i = x, I = i\}, \quad (18)$$

where $C_i = \{U_i = u, X_i = x, I = i\}$. The last probability in (18) is

$$P\{U_i = u | X_i = x, 1 \leq u \leq x + m - n\} = \frac{P\{U_i = u | X_i = x\}}{1 - P\{U_i = 0 | X_i = x\}}$$

where the probabilities on the right hand side are given by (7).

To complete the evaluation of (17), it remains to obtain an expression for the probability in (18). We can

express it as

$$P_j\{\mathcal{R}_k|C_i\} = \int_0^\infty P_j\{\mathcal{R}_k|C_i, T_i = t\}P\{T_i \in dt\}|C_i\}, \tag{19}$$

where T_i is defined as the local virtual time of processor i minus the GVT. To evaluate this integral, note that for processor i to rollback to move process X to state k when $(X = j, X_j = x, U_i = u, T_i = t)$, exactly $[x-(j-k)]$ processed uncommitted messages must have timestamps below the minimum of the timestamps of the u past messages and the rest of the processed uncommitted messages $[(j-k)$ of them] must have timestamps above this minimum. The minimum timestamp of the u past messages has the distribution

$$1-(1-s/t)^u, \quad 0 \le s \le t. \tag{20}$$

This is the probability that u samples from a uniform distribution on $[0,t]$ all fall in the interval $[0,s]$. The probability of this rollback event is therefore

$$P\{\mathcal{R}_k|C_i, T_i = t\} = \binom{x}{x-j+k} \cdot \int_0^v (s/t)^{x-j+k}(1-s/t)^{j-k}[u(1-st)^{u-1}t^{-1}]ds. \tag{21}$$

The last term in the integral is the density of the distribution (20) that the minimum of the u timestamps for past messages equals s. (We are simply conditioning on the distribution (20)). The rest of the right side of (21) is the binomial probability that exactly $(x-j+k)$ out of the x processed uncommitted messages have timestamps below s. Substituting $y = s/t$, the integral reduces to

$$u\int_0^1 y^{x-j+k}(1-y)^{u+j-k-1}dy = u\,B(x-j+k+1, u+j-k) \tag{22}$$

where

$$B(a,b) = \int_0^1 y^{a-1}(1-y)^{b-1}dy \tag{23}$$

is the beta function. Note that this integral is independent of t. Then (21) is also independent of t and hence it factors out of the integral (19). In other words, substituting (21), (22) in (19), we obtain

$$P_j\{\mathcal{R}_k|C_i\} = u\binom{x}{x-j+k} \cdot B(x-j+k+1, u+j-k) \tag{24}$$
$$\text{for } 0 \le u \le (x+m-n), \quad 0 \le x \le j,$$
$$0 \le j \le (M-m), \quad 0 \le k \le j, \quad 1 \le i \le n.$$

This completes our evaluation of the rollback probability $P_j\{\mathcal{R}_k\}$.

4.2.4 GVT Advance As above, we assume that, at a transition of X, the processor n is the regulator and $P_j\{.\}$ is conditioned on this assumption. As before T_i denotes the virtual time of processor i minus the GVT. Set $T = \min\{T_1, \ldots, T_{n-1}\}$. Then the probability of a decrease of the process from j to k due to a GVT advance is

$$P_j\{\mathcal{D}_k\} = \sum_{i=1}^{n-1}\sum_{x=0}^{j} P_j\{X_i = x\}P_j\{T = T_i|X_i = x\} \cdot P_j\{\mathcal{D}_k|X_i = x, T = T_i\} \tag{25}$$

The first probability in the sum is given by (15). The second probability in the sum has a closed form expression, but it is not practical for computations. We therefore use the approximation: The quantities X_l are the same for all the processors l that are not the regulator before or after the transition of X. Then, for such a processor, $X_l = \left[\frac{j-x}{n-2}\right]$ (the integer part of the number). Note also that

$$p(x,y) = P_j\{T_i \le T_l|X_i = x, X_l = y\} = \sum_{k=x+1}^{x+1+y}\binom{x+1+y}{k}\left(\frac{1}{2}\right)^{x+1+y}.$$

Then our approximation is

$$P_j\{T = T_i|X_i = x\} = \left[p\left(x, \left[\frac{j-x}{n-2}\right]\right)\right]^{n-2}. \tag{26}$$

To evaluate the last probability in (25), we will use the following events:

$\mathcal{A}$ = {The GVT advances}

$\mathcal{P}$ = {At a transition, the message at the new GVT was partially processed before the transition}.

Since we will consider the last probability in (25) for fixed i, j, x, we will write it as $\overline{P}\{\mathcal{D}_k\}$, where $\overline{P}$ is the probability defined for any event B by

$$\overline{P}\{B\} = P_j\{B|X_i = x, T = T_i\}, \quad (i, j, x \text{ are fixed})$$

Conditioning on $\mathcal{A}$ and $\mathcal{P}$, we have

$$\overline{P}\{\mathcal{D}_k\} = \overline{P}\{\mathcal{A}\}\overline{P}\{\mathcal{P}\}\overline{P}\{\mathcal{D}_k|\mathcal{A},\mathcal{P}\} + \overline{P}\{\mathcal{A}\}\overline{P}\{\mathcal{P}^c\}\overline{P}\{\mathcal{D}_k|\mathcal{A},\mathcal{P}^c\} \tag{27}$$

Here $\mathcal{P}^c$ is the complement of $\mathcal{P}$. If $T^\star$ is the minimum timestamp of all unprocessed events, then $T^\star - GVT$ is exponentially distributed with rate $n\gamma$ (it is the smallest timestamp from the n merged streams whose rates are γ). It follows that

$$P\{\mathcal{P}\} = \left(\frac{\alpha}{\alpha+n\gamma}\right)^{x+1} = (1+nr_\gamma)^{-x-1}, \quad x = 0, \ldots, j, \tag{28}$$

($r_\gamma = \gamma/\alpha$ as in (13)). This is the probability that $x+1$ events from a Poisson stream of rate α occurs before one event occurs from a Poisson stream of rate $n\gamma$.

To derive $\overline{P}\{\mathcal{A}\}$, we view the regulator as being in one of the two states:

> ***state 0*** : The regulator's virtual time is greater than GVT (the new GVT event is therefore an unprocessed one).
>
> ***state 1*** : The regulator's virtual time equals the GVT.

Considering the movement of the regulator as a Markov chain on these two states, its transition probabilities are

$$\begin{aligned} p_{01} &= 1, \\ p_{11} &= \sum_{j=1}^{n-1}\sum_{x=0}^{j} \overline{P}\{\mathcal{P}\}P_j\{X_i = x, T = T_i\}. \end{aligned} \quad (29)$$

The equilibrium distribution of this chain being in state 1 is therefore $1/(2-p_{11})$. It follows that

$$\overline{P}\{\mathcal{A}\} = 1/[n(2-p_{11})], \quad (30)$$

where n^{-1} is the probability that process i finishes processing before the other ones finish.

We now consider $\overline{P}\{\mathcal{D}_k|\mathcal{A},\mathcal{P}\}$. As the process X moves from j to k due to a GVT advance and the new GVT event was partially processed in processor i with $X_i = x$ before the advance, then $(j-k-x)$ processed uncommitted events will be committed from the other $n-2$ processors (excluding i and n). Thus

$$\begin{aligned} \overline{P}\{\mathcal{D}_k|\mathcal{A},\mathcal{P}\} &= \int_0^\infty e^{-(n-2)\alpha s}\frac{[(n-2)\alpha s]^{j-k-x}}{x!} \\ &\quad .\alpha(\alpha s)^x \frac{e^{-\alpha s}}{x!}ds \qquad (31) \\ &= c\binom{j-k}{x}\frac{(n-2)^{j-k-x}}{(n-1)^{j-k+1}}, \\ &\quad k = 0,\ldots,j-k. \end{aligned}$$

where $c = \left[\sum_{k=0}^{j-k}\binom{j-k}{x}\frac{(n-2)^{j-k-x}}{(n-1)^{j-k+1}}\right]^{-1}$. A similar argument yields

$$\begin{aligned} &\overline{P}\{\mathcal{D}_k|\mathcal{A},\mathcal{P}^c\} \\ &= \int_0^\infty e^{-(n-1)\alpha s}\frac{[(n-1)\alpha s]^{j-k}}{(j-k)!}\gamma e^{-\gamma s}\,ds \\ &= \tilde{c}\left[\frac{n-1}{r_\beta+(n-1)}\right]^{j-k}, \quad k = 0,\ldots,j, \quad (32) \end{aligned}$$

where $\tilde{c}^{-1} = \sum_{k=0}^{j}\left(\frac{n-1}{r_\beta+n-1}\right)^k$. Here if processor i is the new regulator and the new GVT event is unprocessed, then all $j-k$ processed uncommitted events will be committed from the $n-1$ processors excluding i.

Finally, we note that the probability that the regulator is in state 0 equals $(1-p_{11})/(2-p_{11})$. Thus, the probability $P_j\{\mathcal{R}_j, I = n, X_n = 0\}$, in the last section, that the regulator rolls back from j to j, is

$$P_j\{\mathcal{R}_j, I = n, X_n = 0\} = \frac{1-p_{11}}{n(2-p_{11})}. \quad (33)$$

4.3 Estimation of Parameters

The transition probabilities we have derived above are functions of the parameters $r_\beta = \beta/\alpha$, $r_\gamma = \gamma/\alpha$, where α, β are unknown. We now present an iterative procedure for computing r_β, r_γ and the equilibrium distribution π. We first express r_β, r_γ as a function of π. Whenever $X_i = x$ in processor i, its virtual time (with respect to GVT) T_i is the location of the $(x+1)$-th message in a Poisson process with rate α. Therefore,

$$E(T_i|X_i = x) = (x+1)/\alpha. \quad (34)$$

This T_i can also be viewed as location of the Y_i-th point of a Poisson process with rate γ. Therefore,

$$E(T_i|X_i = x) = E(Y_i|X_i = x)/\gamma.$$

Combining these expressions yields

$$r_\gamma = \gamma/\alpha = E(Y_i|X_i = x)/(x+1).$$

Now viewing X_i as a random variable whose distribution is determined from the equilibrium distribution π, a reasonable estimate of r_γ is

$$r_\gamma = \sum_{x=0}^{M-m}\frac{E(Y_i|X_i = x)}{(x+1)}P\{X_i = x\}, \quad (35)$$

where

$$P\{X_i = x\} = \sum_{j=x}^{M-m} P_j\{X_i = x\}\pi_j. \quad (36)$$

To estimate r_β, we must consider antimessages. The timestamps of antimessages are the same as that of their parent processed uncommitted events. As an approximation, we suppose that if there are $Z_i = z$ antimessages in processor i, then they correspond to processed uncommitted messages whose timestamps may be smaller than the antimessages' parents. Then similarly to the above,

$$\begin{aligned} E(T_i|X_i = x) &= \alpha^{-1}[x - E(Z_i|X_i = x)] \\ &\quad + \beta^{-1}E(Z_i|X_i = x) \\ &= [r_\beta x + (1-r_\beta)E(Z_i|X_i = x)]/\beta. \end{aligned}$$

Applying (34) to the first term here and then solving for r_β, we have

$$r_\beta = \frac{E(Z_i|X_i = x)}{1 + E(Z_i|X_i = x)}$$

Thus, a reasonable estimate of r_β is

$$r_\beta = \sum_{x=0}^{M-m} \frac{E(Z_i|X_i = x)}{1 + E(Z_i|X_i = x)} P\{X_i = x\} \qquad (37)$$

Our iterative procedure for computing r_β, r_γ, π is as follows. For an initial setting of r_β, r_γ we compute π from the balance equations. Using this π, we compute new values of r_β, r_γ from (35) and (37), and use these in the balance equations to compute a new π. We repeat this procedure until the new values of r_β, r_γ are within a distance of 0.001 from their previous values. Performance measures can then be computed as described in the following subsection.

Extensive computations with this procedure showed it to be very efficient. Although we cannot prove that there exist unique fixed points for r_β, r_γ, the data suggests that this might be true. In runs of the procedure for 100 initial values of r_β, r_γ ranging from 0.001 to 1,000, these parameters always converged to the same values. The number of iterations for the convergence depends strongly on their initial values. This number is very low, approximately 2 to 4, when $(M - m)$ is below 10. For larger values of $(M - m)$, we found that the number of iterations can be decreased significantly by using the previously computed values of r_β and r_γ as initial values for the new value of $(M - m)$.

4.4 Performance Measures

The main performance measure of the system is the *number of messages committed per unit time.* This is the same as the rate of GVT advancement per unit time. This quantity is given by

$$\rho = n\lambda \sum_{j=0}^{M-m} \sum_{k=0}^{j} (j + 1 - k) P_j\{D_k\}\pi_j \qquad (38)$$

This follows by a standard law of large numbers for a Markov process. The sum represents the expected decrease in the state of X which is the expected number of committed events per transition, and $n\lambda$ is the total rate of the transitions per unit time.

One can compare this n-processor model to a single one operating in series as follows. Consider a single processor working in series that processes messages over time according to a Poisson process with rate λ. Then the *speedup* of the n-processor system is expressed as

$$S = \frac{\rho}{\lambda} \qquad (39)$$

5 Experimental Results and Validation of the Analytic Model

The analytic model makes a number of simplifying assumptions (for instance, we assume that the timestamps of the processed uncommitted events in each processor follow a Poisson distribution) in order to make the analysis tractable. Measurements on a Time Warp kernel running on a shared memory multiprocessor (specifically, a BBN Butterfly GP-1000) were made and compared with the performance predicted by the analytic model in order to test the validity of the approximations underlying the analysis.

The assumptions used in the analytic model pertaining to the workload (exponential execution time per event, exponential timestamp increment, fixed message population, etc.) correspond to a specific instance of the parallel HOLD (PHOLD) workload model [5]. This model was used in the experiments performed here, and a synthetic application program was designed that corresponds to the workload assumed in the model. Also the mean computation time per event was assigned a relatively high value (50 millisecond per event) to reduce Time Warp overheads to a negligible level, and the experiments here utilize a high message population m (32 times the number of processor) to satisfy the non-idle processor assumption. Experiments evaluating the impact of relaxing these assumptions are currently under progress.

The original Time Warp kernel we have used is described in detail in [3]. This kernel has been modified to make the memory to store messages accessible as a globally shared pool. As described in [3] the antimessages in the kernel are mere pointers to the corresponding positive messages organized in a *causality record*, and these pointers require a constant amount of space per event. The state vector and the pointers implementing the causality record are included in each message, (recall that the state vector is saved before processing each message) and are not separately allocated/deallocated. Memory buffers to store messages are dynamically allocated from the global pool when a process wants to send a message to another process. The only way storage can be freed is when a message is cancelled or sent back, or when it is fossil collected. GVT computation and fossil collection are performed atomically and on-demand. All processors are involved in fossil collection using a global synchronization. When memory is needed and

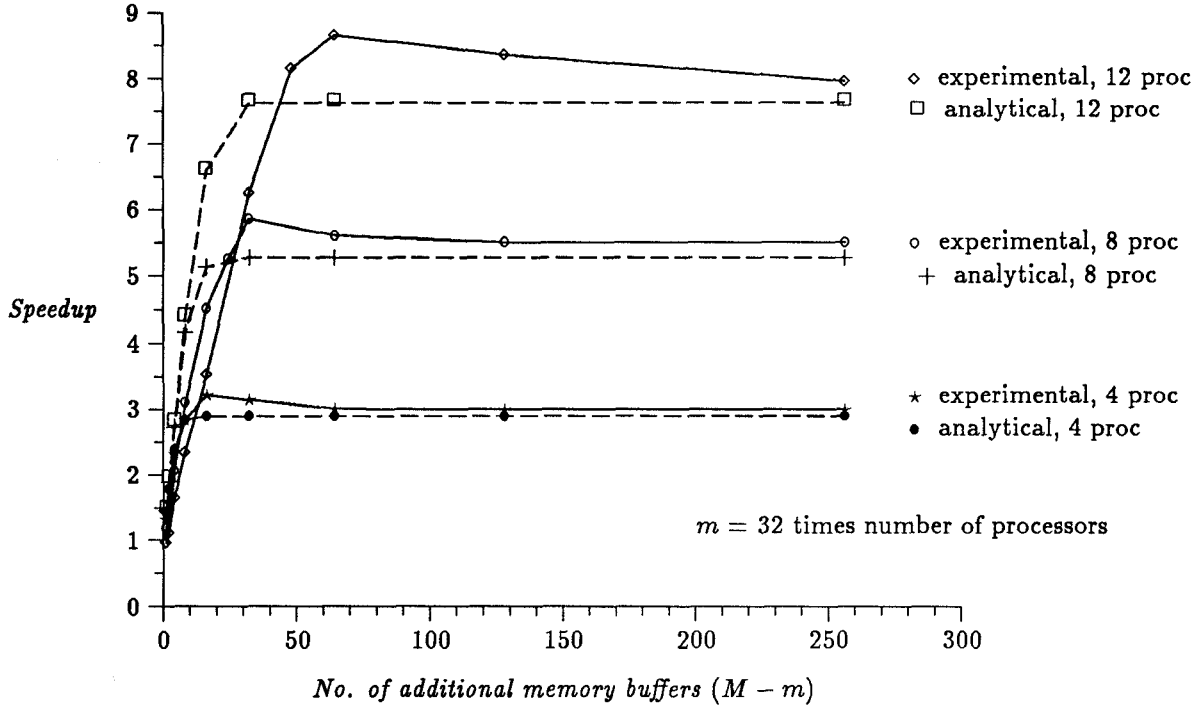

Figure 1: Effect of Limited Memory Capacity on Speedup

no free buffer is available in the shared pool, fossil collection is invoked to reclaim memory used by possible fossils. If no fossil exists, cancelback (as described in section 3) is invoked to free storage.

We have run the experiments with 4, 8 and 12 processors on the BBN Butterfly. The effect of memory capacity on speedup is shown in Figure 1. Performance monitoring indicated that GVT computation and fossil collection require a considerable amount of time. It is observed that these computations may consume as much as about 80% of the total execution time in our system when the amount of available memory is small. In this case the frequency of GVT computation and (possible) fossil collection is very high. To eliminate the effect of this large overhead, which is not included in the analytic models, we subtracted the total time spent in GVT computation and fossil collection from the total execution time, and use the difference to compute speedup. Thus the experimental data shown in Figure 1 is as if GVT computation and fossil collection are instantaneous.

If the total memory capacity M in the shared pool is larger than some threshold value, fossil collection alone is sufficient to produce enough free memory for the simulation to progress. However for smaller amounts of memory, cancelback needs to be invoked occasionally to enable progress. Frequency of cancelback invocations increases as memory is reduced further, and speed of execution is reduced as the progress of many processes in virtual time is throttled by frequent cancelbacks. Memory capacity M, however, cannot be reduced indefinitely. Recall that M must at least be m as this is the number of message buffers required in the corresponding sequential execution.

The modest discrepancies between the analytical prediction and experimental results shown in Figure 1 can be attributed to the assumptions used in developing the analytic model, which are only approximately true in the real experiments performed. It is noted that with increase in memory capacity from the value required by the sequential simulation, performance increases very sharply, and then becomes almost flat. It is observed that the speedup curve has a well-defined "knee." The location of this knee indicates the minimum number of message buffers needed to complete the Time Warp simulation as rapidly as in the unbounded memory situation. It is interesting to note that knee occurs at very low values of memory capacity. The analytical model predicts that only 2 to 3 additional buffers per processor beyond which is required for the sequential simulation are necessary to achieve the same performance as with unbounded memory. The measurements agree with this result, but require slightly larger number of buffers (about 5 per processor) to achieve the performance of the unbounded memory case.

The experiments also predict a slight *decrease* in per-

formance as memory capacity increases just beyond the knee of the curve. This may be due to the fact that limited memory capacity prevents processes from progressing far ahead in virtual time with respect to the GVT. Overly optimistic processes are more prone to rollbacks than others. If the rollback overhead is non-zero, infusing more optimism in the mechanism by increasing memory capacity actually degrades performance. Also, large memory capacity increases input queue sizes (note that the message population is relatively high), and queue management takes more time [3] thus adversely affecting performance. The analytic model cannot predict such performance decrease with increasing memory because of its zero-overhead assumptions.

6 Conclusions and Future Research

The principal contribution of this work is the development of an analytic model of Time Warp augmented with cancelback operating with limited memory. To our knowledge this is the first attempt to model the limited memory behavior of Time Warp. We developed a Markov chain model and derived expressions to predict the performance of the Time Warp system in terms of speedup.

In addition, we validated our analytical model through measurements of an operational Time Warp system. It is demonstrated that for homogeneous workloads Time Warp performs reasonably well with a very limited amount of memory (a few extra buffers per processor), if global computation overheads such as GVT computation and fossil collection are discounted. Further investigation is necessary to (i) speed up these computations or (ii) modify the memory management protocol (e.g., cancelling back more than one object at a time) to minimize the effect of these overheads.

In addition to this, our future work includes the following topics:

- developing analytical models for memory management schemes for distributed memory Time Warp systems (such as those using message sendback or Gafni's protocol),
- performance comparisons of different memory management schemes (for example infrequent state saving schemes versus flow control based schemes such as message sendback), and
- investigating the cases where Time Warp overheads and heterogeneous processors and/or workloads are taken into account.

Acknowledgement

We thank the anonymous referees whose insightful comments significantly improved this paper. The work of Akyildiz, Das and Fujimoto was supported by Innovative Science and Technology contract number DASG60-90-C-0147 provided by the Strategic Defense Initiative Office and managed through the Strategic Defense Command Advanced Technology Directorate Processing Division, and by NSF grant number CCR-8902362. The work of Chen and Serfozo was supported in part by AFOSR 89-0407 and NSF grant DDM-9007532.

References

[1] R. E. Felderman and L. Kleinrock. An upper bound on the improvement of asynchronous versus synchronous distributed processing. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):131–136, January 1990.

[2] R. E. Felderman and L. Kleinrock. Two processor Time Warp analysis: Some results on a unifying approach. *Proceedings of the Multiconference on Advances in Parallel and Distributed Simulation*, 23(1):3–10, January 1991.

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

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

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

[6] A. Gafni. Rollback mechanisms for optimistic distributed simulation systems. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):61–67, July 1988.

[7] A. Gupta, I. F. Akyildiz, and R. M. Fujimoto. Performance analysis of Time Warp with multiple homogenous processors. *IEEE Transactions on Software Engineering*, 17(10):1013–1027, October 1991.

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

[9] D. R. Jefferson. Virtual time II: The Cancelback protocol for storage management in distributed simulation. In *Proc. 9th Annual ACM Symposium on Principles of Distributed Computation*, pages 75–90, August 1990.

[10] D. R. Jefferson and A. Witkowski. An approach to performance analysis of timestamp driven synchronization mechanisms. *Proceedings of the 3rd Annual Symposium on Principles of Distributed Computing*, 1984.

[11] S. Lavenberg, R. Muntz, and B. Samadi. Performance analysis of a rollback method for distributed simulation. In *Performance '83*, pages 117–132, Elsevier Science Pub., (North Holland), 1983.

[12] Y-B. Lin. Memory management algorithms for optimistic parallel simulation. *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation*, 24(3):43–52, January 1992.

[13] Y-B. Lin and E. D. Lazowska. Optimality considerations of "Time Warp" parallel simulation. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):29–34, January 1990.

[14] Y-B. Lin and E. D. Lazowska. Reducing the state saving overhead for Time Warp parallel simulation. Technical Report 90-02-03, Dept. of Computer Science, University of Washington, Seattle, Washington, February 1990.

[15] R. J. Lipton and D. W. Mizell. Time Warp vs. Chandy-Misra: A worst-case comparison. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):137–143, January 1990.

[16] B. D. Lubachevsky, A. Shwartz, and A. Weiss. Rollback sometimes works ... if filtered. *1989 Winter Simulation Conference Proceedings*, pages 630–639, December 1989.

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

[18] V. Madisetti, J. Walrand, and D. Messerschmitt. Synchronization in message-passing computers-models, algorithms, and analysis. *Proceedings of the SCS Multiconference on Distributed Simulation*, 22(1):35–48, January 1990.

[19] D. Mitra and I. Mitrani. Analysis and optimum performance of two message passing parallel processors synchronized by rollback. *Performance Evaluation J.*, 7:111–124, 1987.

[20] D. M. Nicol. Performance bounds on parallel self-initiating discrete-event simulations. *ACM Transactions on Modeling and Computer Simulation*, 1(1):24–50, January 1991.

[21] B. D. Plateau and S. K. Tripathi. Performance analysis of synchronization for two communicating processes. *Performance Evaluation J.*, 8:305–320, 1988.

[22] B. R. Preiss, I. D. MacIntyre, and W. M. Loucks. On the trade-off between time and space in optimistic parallel discrete-event simulation. *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation*, 24(3):33–42, January 1992.

[23] L. M. Sokol, D. P. Briscoe, and A. P. Wieland. MTW: a strategy for scheduling discrete simulation events for concurrent execution. *Proceedings of the SCS Multiconference on Distributed Simulation*, 19(3):34–42, July 1988.

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