Performance Analysis of Time Warp With Multiple Homogeneous Processors

Anurag Gupta, Ian F. Akyildiz, *Senior Member, IEEE,* and Richard M. Fujimoto, *Member, IEEE*

***Abstract*— The behavior of n interacting processors (each process executes on a distinct processor; hence the words "process" and "processor" are used interchangeably) synchronized by the "Time Warp" protocol is analyzed using a discrete state, continuous time, Markov chain model. The performance and dynamics of the processes are analyzed under the following assumptions: exponential task times and timestamp increments on messages, each event message generates one new message that is sent to a randomly selected process, negligible rollback, state saving, and communication delay, unbounded message buffers, and homogeneous processors. Several performance measures such as the fraction of processed events that commit, speedup, rollback probability, expected length of rollback, the probability mass function for the number of uncommitted processed events, the probability distribution function for the virtual time of a process, and the fraction of time the processors remain idle are determined. The analysis is approximate, so the results have been validated through performance measurements of a Time Warp testbed (using the PHOLD workload model) executing on a shared memory multiprocessor.**

***Index Terms*— Time warp, parallel simulation, performance evaluation, Markov chain model, performance measures, synthetic workloads, validation.**

I. Introduction

OVER the last several years, research in synchronization mechanisms for parallel discrete event simulation programs has progressed along two fronts—the *conservative* [19] and *optimistic* [10] approaches. Conservative schemes do not allow an event with timestamp t to be processed if there is a chance that an event with timestamp s, where $s < t$, may arrive. In such a scheme, computation of an event with timestamp t can proceed only after it is determined that no event with a lower timestamp can later arrive. These schemes usually utilize data-dependency information about the simulation model in order to make this decision. On the other hand, Time Warp [10], an optimistic scheme, allows computation of an event with timestamp t to proceed without regard to the possibility of the arrival of another event with a lower timestamp. If an event with a lower timestamp does arrive, the Time Warp scheme "rolls back" to the most recently saved state with virtual time earlier than the timestamp of the arriving event. This approach assumes that state information is periodically saved in order to make rollback possible. The relative advantages and disadvantages of these schemes have been extensively debated [7].

The Time Warp scheme provides a means of synchronizing the execution of tasks on different processors that is superior to synchronous lock-step execution when simulating *asynchronous* systems. In the synchronous lock-step approach only those events containing the same timestamp may be processed in parallel. This approach is inefficient for simulating asynchronous systems, because there are usually few simulator events containing the same timestamp. Also, administering lock-step execution can lead to significant synchronization overheads for large systems.

The Time Warp scheme allows the processes to have different virtual times in their local clocks. It attempts to enforce a partial ordering of events [13]. Processes communicate only by exchanging messages. Each message is stamped with a *virtual send time,* and a *virtual receive time.* Virtual send time is the local time of the sender when the message is sent. Likewise, virtual receive time is the virtual time at which the message is to be processed by the receiving processor, and is also referred to as its *timestamp.* The receiver compares the timestamp of the message with its virtual time. If the message is "in the future," it is queued for later processing; if the message is "in the past," the computation must be rolled back to the most recently saved state with virtual time less than the timestamp of the received message.

The primary question that arises concerns the speedup which can be obtained when using n processors. A related issue is determining the fraction of processed events that are eventually committed, or equivalently, the fraction of "wasted" events that are eventually rolled back. In addition, there are several other performance measures (e.g., rollback probability, and probability mass function of the number of uncommitted processed events at a process) which characterize the dynamics of the Time Warp program. We obtain analytical solutions for these performance measures.

The analytical results are validated against performance measurements of an implementation of Time Warp executing on a shared memory multiprocessor [6]. A synthetic workload model called the parallel hold (PHOLD) model is used in these experiments. The PHOLD model was parameterized to be consistent with the assumptions used in the analytic model. Workloads of varying parallelism were executed and

Manuscript received February 1, 1991; revised June 25, 1991. Recommended by E. Gelenbe. This work was supported by Innovative Science and Technology under Contract No. DASG-60-90-C-0147 provided by the Strategic Defense Initiative Office and managed though the Strategic Defense Command Advanced Technology Directorate Processing Division, and by the NSF through Grant CCR-8902362. An earlier version of this paper appeared in the Proceedings of the 1991 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.

The authors are with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332.

IEEE Log Number 9102618.

the observed performance measures were used to validate the analytically obtained values. All simulations in the PHOLD model are run with adequate buffer capacity so that the unbounded buffer assumption is not violated. The relationship between speedup and the number of processors is especially crucial to the design of system prototypes. The analysis provides a theoretical basis for some of the empirical performance measurements in [6].

To our knowledge, this is the first analytic performance model for Time Warp that has been compared with measurements of an operational Time Warp system. Many analyses have dealt with the two-processor case, and could not be extended due to the complexity of the problem. Lavenberg, *et al.* [14] have obtained an approximate solution for two processors with very low interaction. Mitra and Mitrani [20] have obtained exact solutions for the two-processor case under more general conditions. Kleinrock [12] has considered a discrete state continuous time model for the two-processor case. Felderman and Kleinrock [4] have considered a discrete time, discrete state Markov model, and by taking limits have provided a unifying framework for previous work on two processor Time Warp. Plateau and Tripathi [22] have obtained numerical results for the rate of message exchange, and blocking probabilities for two communicating processors. They have employed tensor algebra to handle the three-dimensional Markov chain model of the system. Jefferson and Witkowski [11] have proposed a new stochastic process—the linear Poisson process—to model timestamp driven schemes. Performance analyses of the general n process case have appeared recently. Madisetti *et al.* [18] have derived an analytical estimate for the progress of distributed computation for the two processor case. They have also investigated different synchronization schemes for the general n process case. Felderman and Kleinrock [2] give an upper bound on the gain in speedup that n asynchronous processes can achieve relative to n synchronous lock-step processes for large n. Nicol [21] derives an upper bound on Time Warp's performance for the general n processor case. Nicol considers a self-initiating model which schedules its own state re-evaluation time, as opposed to our model where a process's state is affected when messages are received from other processes. Further, his analysis of Time Warp ignores effects due to rollback propagation. Greenberg *et al.* [9] present methods for parallel discrete event simulation when the number of processors is larger than the number of simulated objects. Lin and Lazowska [15] show that Time Warp always performs at least as well as any conservative mechanism (and possibly better) under certain conditions; like our analysis, they assume that the overheads for state saving and rollback are negligible. Lipton and Mizell [16] demonstrate that while Time Warp may outperform the Chandy/Misra algorithms (well-known conservative algorithms) [1], [19] by an arbitrarily large amount, the opposite is not true; i.e., the Chandy/Misra algorithm can only outperform Time Warp by a constant factor. Lubachevsky *et al.* [17] present a tunable "filter" to bound the lag so that their algorithm is conservative at one extreme, and is optimistic at the other extreme, and identify conditions that lead to instability.

This paper is organized as follows. The model is described in Section II. The model is analyzed and related equations are derived in Section III. The analytical results are compared with performance measurements of the prototype Time Warp system in Section IV. Finally, conclusions and suggestions for future research are given in Section V.

II. The Model

We assume the processors to be homogeneous with an unbounded buffer to store received messages. Each processor is assumed to have its own local buffer space. These assumptions are reasonable for current multiprocessors such as the Sequent, Butterfly, and iPSC if the memory is large relative to the size of the problem. If the buffers are bounded and the receiving processor has a full buffer, then the processor sends back the message just received to the sender, usually causing the sender to roll back. The "obvious" approach in which a sender process blocks until the receiver relinquishes its buffer space is prone to deadlock. Further, we assume that the processing time of events is exponentially distributed, and each event produces a single new event with a timestamp increment (i.e., receive time minus send time) which is selected from an exponential distribution. The new event message is equally likely to be sent to any other process. Many real simulations stabilize at a message population after the transient phase with minor fluctuations. In real simulations, one new event is generated *on an average;* in our model, exactly one event is generated after processing an event. This is a simplifying assumption to make the analysis tractable. This assumption *does* hold for some systems; e.g., simulations of closed queueing networks. In general, it may be true for systems in which a fixed number of objects interact. The assumption that a message is equally likely to be sent to another processor has been made to keep the analysis simple; in real simulations, there is some amount of locality and processors tend to communicate more often with those in its immediate neighborhood. Similarly, the timestamp increments and processing times have been assumed to be exponentially distributed to make the analysis tractable. However, measurements with other timestamp distributions and localities show that these have a secondary effect [6].

We assume that the simulation is partitioned into n processes, each of which is executed on a separate processor. We also assume that there are initially m unprocessed event messages in the system, and that upon processing each message, one new message is generated. The quantity m is referred to as the *message population.* Hence the "logical" number of messages remains the same (m) throughout. Note that there are simulations which run with a constant message population, and there are those which do not. The physical number of messages changes, since antimessages received "in the past" take a nonzero time to annihilate their counterparts. The nonzero time is due to a non-pre-emption assumption to be discussed shortly. Each processor is equipped with a virtual local clock which indicates the virtual time of that processor. Virtual times are real values totally ordered by the relation $<$. As the events are processed, the virtual local clocks in different processors move to higher virtual times, although they occasionally jump backwards when a rollback occurs. It is assumed that the time to rollback is negligible. This assumption is realistic when the computation granularity (processing time) of an event is large. We also assume a non-pre-emptive rollback; i.e., if a message in the past is received while an event is being processed, the rollback does not take effect until we finish processing the current event. If more than one message with a timestamp "in the past" arrive during processing, the effect is the same as if only the message with the least timestamp had arrived. Also, we assume that that state is saved after processing *every* event. This ensures that a process will rollback to the event with timestamp immediately less than the timestamp of the incoming late arrival, rather than to an earlier state, viz., the last saved state. The processing of an event involves the following operations:

i) receive a message with timestamp t

ii) compare the timestamp t with the receiver's virtual clock times

iii) if $t < s$, roll back to time t

iv)
- set virtual clock to t
- read contents of message
- update state variables
- send a message with timestamp $t + \xi$ (where ξ is an exponentially distributed random variable).

The message which is sent after processing an event is equally likely to be sent to any of the processes (including itself). Communication delay for the message is assumed to be negligible. The local clock does not change during the time an event is processed; it changes only between events, and then only to the timestamp of the next message to be processed.

Jefferson [10] defines the notion of *global virtual time* (GVT). GVT serves as a floor (lower limit) for the virtual time to which a process will ever roll back. The state of the system can then be defined as $(k_1, k_2, \cdots, k_n)$, where k_i, $1 \leq i \leq n$ is the number of events that have been processed at process i that have a timestamp greater than the GVT. Fig. 1 shows the case where there are three processors. However, our analysis holds for the general n-processor case. An event that contains a timestamp which is less than GVT is called a *committed* event, and others are called *uncommitted* events. Any event with timestamp less than the GVT cannot be rolled back and can be committed safely. Fig. 1 shows processor 1 with one uncommitted (but processed) event, and another event being processed. For the range of virtual times that are shown next to the events, processor 1 has no committed events, while processors 2 and 3 each have one. It may be noted that process 3 has an unprocessed event with a timestamp *less* than the timestamp on the one being processed. Due to the non-pre-emptive nature of the model, the arrival of event with timestamp 11.1 is not observed by the process. This is because the event with timestamp 13.1 arrived earlier than the event with timestamp 11.1, and the former was being processed when the latter arrived.

The virtual time of a processor is the timestamp of the event being processed. GVT is the minimum of all virtual processor times (in this case, the virtual time of processor 2) and the timestamps of all unprocessed messages. The latter is necessary to account for received messages which have not

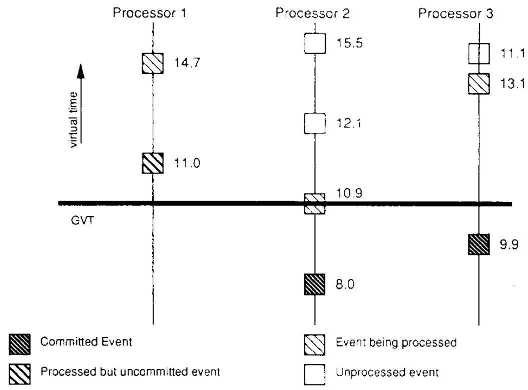


Fig. 1. System in state $(1, 0, 0)$; i.e., processor 1 has one processed uncommitted event, and processors 2 and 3 have none.

yet caused a rollback due to the non-pre-emptive nature of the model. In such a case, a message with a lower timestamp will be waiting in the input queue, while a message with higher timestamp is being processed. The integer values in the state tuple are measures of the amount of work done (number of processed uncommitted events) in the process which has not yet been committed. This characterization of the state is different from earlier studies which measure the progress of a process with respect to another process. The GVT-based characterization allows us to only consider the interaction of a process with GVT rather than with all of the processes. The rate of progress of GVT can be considered to be a measure of system progress.

In contrast to earlier studies mentioned in the previous section which have analyzed the system from the sender process point of view, our analysis is focused on the receiver process's point of view. Cascaded rollbacks are difficult to analyze from the sender process but are more easily accounted for when analyzed at the receiver process, because, at any time, we need to be concerned only with the rollback at the process under consideration. Appealing to symmetry and homogeneity arguments, we observe that it is sufficient to analyze one process and its interaction with GVT. The dynamics at the other processes are identical. (Timestamps of messages being sent out are Poisson distributed in virtual time.) The committed messages at all the receiving processors are also Poisson distributed. This is the well-known Markov implies Markov ($M \rightarrow M$) property. We note that the uncommitted messages will *not* be Poisson distributed. This is because these events include events which will be later cancelled and exclude true messages that have not yet arrived.

During the course of deriving expressions for speedup and the fraction of processed events that will eventually be committed, we will also derive expressions for the expected number of processed uncommitted events at a process, the expected length (in terms of number of events) of rollback, the probability of rollback, rollback probability as a function of number of processed uncommitted events in the buffer, the probability mass function for the number of processed uncommitted events at a process, the probability distribution

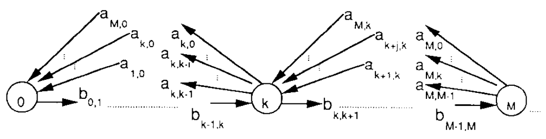


Fig. 2. The Markov chain model.

function of the virtual time at a processor assuming origin at GVT, and an estimate for the fraction of time the processors remain idle.

III. Analysis of the Model

In this section we present an approximate analysis of the performance of multiprocessor Time Warp which we have modeled as a Markov chain. We discuss the state space of the Markov model, derive transition rates and solve the model, determine performance measures, estimate the fraction of time the processors remain idle, and finally present an algorithm for numerical solution of the model.

A. State Description of the Markov Model

We analyze the behavior of n coupled processes, which we have modeled as a one-dimensional discrete state, continuous time Markov chain as shown in Fig. 2. Since we have assumed an unbounded buffer at each process, we could theoretically have an unbounded number of processed uncommitted events at a process. This is because a process can be arbitrarily far ahead of another. This implies that the associated Markov chain has an infinite number of states. However, to solve the system numerically we desire a finite number of states. We observed from subsequent characterizations of the transition rates that the greater the number of processed uncommitted events at a process, the greater the probability of the process being rolled back. Thus, if a process goes far ahead, there is a tendency for it to be pulled back. This indicates the existence of an equilibrium from which we assume that the states in the Markov chain model are not transient. In such a case, given a tolerance ϵ, we can find a finite integer M (dependent on ϵ) such that the difference between performance measures obtained by truncating the Markov chain at M and $M + 1$ states is less than ϵ. Essentially, we are approximating an infinite buffer space by a finite M such that the results obtained are within a tolerance ϵ of the actual results.

To facilitate subsequent discussion we introduce the notion of GVT-regulator. A process is a GVT-regulator if it has the event with the minimum timestamp among all the uncommitted events. Since communication delay is assumed to be zero, an event is guaranteed to be associated with a unique process—either the sender or the receiver. It may be noted that owing to the non-pre-emptive assumption, it is possible for the GVT-regulator to be processing an event with a timestamp higher than the GVT event, because the latter may be waiting to be processed in the GVT-regulator's input queue.

From a state k a process can make the following transitions (Fig. 2):

- rollback to state l_1, for $0 \leq l_1 \leq k$ (rollback)
- come down by l_2 states, $0 \leq l_2 \leq k$ (GVT advancement)
- move to state $k + 1$ after processing the current event (forward movement).

A state change occurs either when the process under consideration (receiver) completes processing of an event, or the GVT-regulator completes its event. A rollback to state l_1 occurs when the receiver process, upon completion of the current event ($k + 1$th), observes an event in the past with a timestamp between the timestamps of the l_1th and $l_1 + 1$th events. A process comes down by l_2 states (moves to state $k - l_2$) when the GVT-regulator completes its event and the GVT of the system moves past l_2 events while the current event is being processed. This is because the process now has l_2 fewer uncommitted processed events after its l_2 events were committed by GVT advancement. A process moves to state $k + 1$ if neither an arrival in the "past" nor GVT advance occurs prior to completion of its current event.

To solve the model we need to derive the transition rates (Fig. 2).

B. Determining Transition Rates

To determine the transition rates, we will deal with sums of independent and identically distributed (i.i.d.) exponential random variables. For this purpose, in Appendix A we develop $L_{j,i}$—the probability that the sum of j independent random variables is less than the sum of i independent random variables, all of which are exponentially distributed with the same rate. In Appendix B we derive $C_{j,i}$—the probability that the sum of j random variables with rate α and a random variable with rate β is less than the sum of $(i + 1)$ random variables with rate α. All of the random variables are independent and exponentially distributed.

We recall that the message population is m. This is the number of "threads" in the parallel simulation. We define the message density to be the ratio of message population m to the number of processors n. If the timestamp increment is exponentially distributed with rate λ, one might expect the messages at a processor to be Poisson distributed in virtual time with rate $m\lambda/n$ (merge m Poisson streams and then split them over n processes). While this is certainly true of the events which have committed and no longer influence computation, this is not true of uncommitted events. The parameter for the Poisson process which consists of processed uncommitted events is less than the above value $(m\lambda/n)$, because lagging processors have not yet sent out their share of messages. In reality, the nearer an event is to the GVT, the smaller will be the distance between uncommitted events. This is because events far ahead of GVT would not have been generated yet (we call these yet-to-be-generated events, "holes")—there will be fewer such holes near the GVT. In our analysis we make the following approximation: in Computing $C_{j,i}$ we have added j random variables, each of which is identical and represents the average distance between two uncommitted events. In reality, the distribution for uncommitted events will only be "pseudo" Poisson, in that although the interdistance between consecutive events will be exponentially distributed, the rates for two consecutive pairs will differ slightly. We approximate this by a Poisson process with rate $\rho\lambda = m'\lambda/n$, where ρ is the effective message density and is yet to be determined. Thus although m' does take into account the number of holes, we do not consider the fact that the virtual time between events is smaller for those near GVT than for those that are far into the (virtual time) future.

Let Q_k be the probability that the message processed after completion of the current event is "in the past," when there are k processed uncommitted events at the processor (excluding the event currently being processed). If the processor is currently idle, Q_k is the probability that the next message to arrive is "in the past," given that the processor k processed uncommitted events in its buffer. Let I (to be determined later) be the probability that a processor is idle. We approximately estimate the fraction of time a processor remains idle by using an urn model. If the processor under consideration is currently processing events (i.e., it is not idle) with a Poisson rate of ω in real time (processing times are exponentially distributed), the events processed by the remaining processors form a Poisson process with rate $(n-1)(1-I)\omega$. Hence the probability that exactly N events are processed by the other processors during the time the considered processor processes one event is:

$$A(N) = \left(\frac{(n-1)(1-I)}{1+(n-1)(1-I)}\right)^N \frac{1}{1+(n-1)(1-I)}.$$

If N events are processed during the time, exactly N (positive) messages are generated. The probability that exactly i_1 of these N positive messages arrive at the considered processor is (superscript "+" denotes that the expression relates to positive messages):

$$[N, i_1]^+ = \binom{N}{i_1}\left(\frac{1}{n}\right)^{i_1}\left(\frac{n-1}{n}\right)^{N-i_1}$$

because messages are routed to a randomly selected process. Let η (to be determined later) be the fraction of processed events that are eventually committed. Then the fraction of processed events that are rolled back is $(1-\eta)$. Each processed event that is rolled back generates an antimessage to annihilate the message that has already been sent. Thus if N messages are processed, then N positive messages will be sent out and, on an average, $N(1-\eta)$ antimessages will be generated. The following expression for Q_k, however, makes the assumption that *exactly* $N(1-\eta)$ antimessages are generated whenever N events are processed by the other processors during the time the considered processor processes its current event. The probability of exactly i_2 of the $N(1-\eta)$ antimessages arrive at the considered process is (superscript "−" denotes that the expression relates to antimessages):

$$[N(1-\eta), i_2]^- = \binom{N(1-\eta)}{i_2}\left(\frac{1}{n}\right)^{i_2}\left(\frac{n-1}{n}\right)^{N-i_2}.$$

Let P_j be the steady-state probability of the process being in state j. We make the following observation in Fig. 3. When a receiver in state k receives a message from a sender in state j "in the past," it implies that the sum of j random variables with rate $m'\lambda/n$ and one random variable with rate

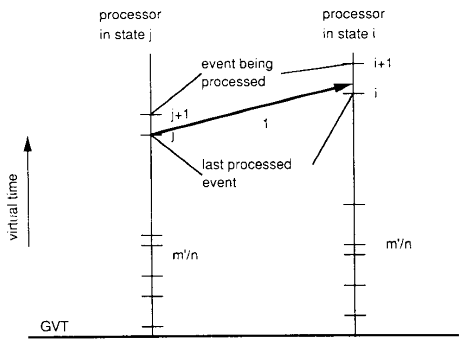


Fig. 3. A message in the past.

λ is less than the sum of $k+1$ random variables with rate $m'\lambda/n$. The j random variables correspond to the interdistance between j events "in the past." Each of these interdistances is assumed to be exponentially distributed with rate $\rho\lambda$. In this context, $\alpha = \rho\lambda$ and $\beta = \lambda$ (see Appendix B), because the processed uncommitted events are distributed with rate $\rho\lambda$ and the timestamp increment is distributed with rate λ. Hence we set $\gamma = \alpha/\beta = \rho\lambda/\lambda = \rho$ in the expression for $C_{j,k}$ given in Appendix B. $C_{j,k}$ can now be interpreted as the probability that a message will be "in the past" of the receiver if the sender process is in state j and the receiver process is in state k. A (positive) message is generated when an event is processed. The probability that a (positive) message arriving at a receiving processor was generated due to completion of the jth event above the GVT at the sending processor is precisely the probability that the sending processor is in state j, viz. P_j. Hence the probability that a (positive) message arrives "in the past" of the (receiving) processor when it is in state k is:

$$J^+(k) = \sum_{j=0}^{M} P_j C_{j,k}, \quad \text{for } k = 0, 1, 2, \cdots$$

where M is the number of states. It follows that the probability that all of the i_1 (positive) messages received by a processor in state k are "in the future" is:

$$B(i_1, k) = \left(1 - [J^+(k)]\right)^{i_1}, \quad \text{for } i_1, k = 0, 1, \cdots.$$

Now consider an antimessage that is the negative image of the jth processed uncommitted event above (later than) the GVT. The antimessage is generated when the processor has l processed uncommitted events in its buffer, $l \geq j$ (probability P_l); the rollback starts (probability Q_l) and continues far enough to include the jth event (probability q^{l-j}, q is a parameter indicating the length of expected rollback and will be determined later). The probability that the generated antimessage is the negative image of the jth event above the GVT is:

$$K^-(j) = \sum_{l=j}^{M} \left(P_l Q_l q^{l-j}\right), \quad \text{for } j = 1, 2, \cdots.$$

Given that the processor is in state k, the probability that an antimessage arriving at the considered processor happens to be "in the past" is:

$$J^-(k) = \sum_{j=0}^{M} K^-(j) C_{j,k}, \quad \text{for } k = 0, 1, \cdots.$$

Given that the processor is in state k, the probability that all of the i_2 antimessages received at the processor are "in the future" is:

$$D(i_2, k) = \left(1 - [J^-(k)]\right)^{i_2}, \quad \text{for } i_2, k = 0, 1, 2, \cdots.$$

Given that the processor is in state k, the probability that at least one of the i_1 (positive) messages and i_2 messages received by the processor is "in the past" is:

$$J(i_1, i_2, k) = 1 - B(i_1, k) D(i_2, k), \quad \text{for } i_1, i_2, k = 0, 1, 2, \cdots.$$

Given that the receiving processor is not idle and is in state k, the probability that the message that is processed after completion of the current event will be "in the past" is:

$$E(k) = \sum_{N=0}^{\infty} \left(A(N) \sum_{i_1=0}^{N} \left([N, i_1]^+ \sum_{i_2=0}^{N(1-\eta)} \left([N(1-\eta), i_2]^- J(i_1, i_2, k) \right) \right) \right), \quad \text{for } k = 0, 1, 2 \cdots$$

where N is the number of events processed by the remaining $n-1$ processors during the time the receiving processor processes its current event. If the receiving processor is idle, the processor rolls back if the next arriving message is positive (probability $1/(2-\eta)$) and "is in the past" (probability $J^+(k)$), or the message is an antimessage (probability $(1-\eta)/(2-\eta)$) and is "in the past" (probability $J^-(k)$). It may be noted that for every processed event (each of which generates a positive message), a fraction $1-\eta$ is rolled back (each event rolled back generates an antimessage). Hence on an average, $1/(1+(1-\eta))$ fraction of the messages are positive, and the remaining $(1-\eta)/(2-\eta)$ are antimessages. Given that the receiving processor is idle and in state k, the probability that the arriving message is "in the past" is:

$$E'(k) = \frac{1}{(2-\eta)} J^+(k) + \frac{(1-\eta)}{(2-\eta)} J^-(k), \quad \text{for } k = 0, 1, 2, \cdots.$$

In the following expression we assume that the steady-state distributions at all the processes are independent. This product-form assumption is also implicit in the derivation of some of the other expressions later on. Hence,

$$Q_k = (1-I)E(k) + IE'(k), \quad \text{for } k = 0, 1, \cdots, M. \tag{1}$$

1) Transition Out of State k: Now we determine the probabilities of rollback (R_k), GVT advancement (G_k), and forward movement of a process (F_k) while a process is in state k. Processing times of events at all of the processes are assumed to be identical and exponentially distributed. The mean processing time is independent of the mean timestamp increment.[1] Due to the non-pre-emption assumption, it is possible for the GVT-regulator (process having the unprocessed event with least timestamp) to be in state k, $k > 0$. However, we observe that the higher the value of k, the lower is the probability that the processor is GVT-regulator. This is later verified in (8). Intuitively, a process is more likely to roll back to the immediate past than it is to the distant past. This issue of locality has also been discussed in [10]. In deriving (2)–(7), we ignore the possibility that a process in state k, $k > 0$ is the GVT-regulator. When a process is in state k, $k > 0$ (process is not the GVT-regulator), one of the following can occur:

i) the process completes execution of its current event and observes a message "in the past" (rollback with probability R_k),
ii) the GVT-regulator completes execution of its current event and observes a message "in the past,"
iii) the process completes execution of its current event and does not observe a message "in the past" (forward movement with probability F_k), or
iv) the GVT-regulator completes execution of its current event and does not observe a message "in the past" (GVT advancement with probability G_k).

It is possible for the GVT-regulator in state 0 to observe an event "in the past" due to the non-pre-emptive rollback assumption. Now consider two processes—one is the GVT-regulator, and the other is the process under consideration (receiver process). As the processing times of the processes are i.i.d. exponential, each of the processes is equally likely (i.e., probability $1/2$) to complete execution of its current event first. Among the above four actions, only the second one does not affect the state of the process (GVT-regulator finishes before the process with probability $1/2$, and observes an event in the past with probability Q_0). Hence the normalization factor $(1 - Q_0/2)$ is used in the denominator of expressions (2)–(4). A rollback occurs when the process completes execution of its current event before the GVT-regulator (probability $1/2$) and observes an event in the past (Q_k):

$$R_k = \frac{1}{2} Q_k \bigg/ \left(1 - \frac{Q_0}{2}\right), \qquad k = 1, 2, \cdots. \tag{2}$$

The process moves forward when it completes execution of its current event before the GVT-regulator (probability $1/2$), and does not observe an event "in the past" (probability $1 - Q_k$):

$$F_k = \frac{1}{2}(1 - Q_k) \bigg/ \left(1 - \frac{Q_0}{2}\right), \qquad k = 1, 2, \cdots. \tag{3}$$

The GVT advances when the GVT-regulator finishes first (probability $1/2$) and does not observe an event "in the past" (probability $1 - Q_0$):

$$G_k = \frac{1}{2}(1 - Q_0) \bigg/ \left(1 - \frac{Q_0}{2}\right), \qquad k = 1, 2, \cdots. \tag{4}$$

[1]The value of this mean does not enter our analysis because the overhead costs such as rollback, state saving, communication, etc., have been assumed to be negligible in our model and all the phenomena at the uniprocessor and the multiprocessor are scaled in the same proportion, leaving the performance measures unchanged.

Note that G_k, for $k > 0$ is independent of k. This is because the rate at which the GVT-regulator processes events is independent of the state of the considered processor. When the considered process is in state 0, we must take into account the possibility of its being the GVT-regulator. Here we assume that the GVT-regulator is identical to all the processes in state 0. Specifically, whether a process is a GVT-regulator (with probability $1/i$) or not (with probability $(i-1)/i$) is assumed to be independent of the probability of observing an event in the "past" upon completion (Q_0). This is not strictly true because the GVT-regulator is known to have the unprocessed event with the least timestamp (either in the input queue or being processed), so it is less likely to receive another event with a timestamp lower than the timestamp of the event being processed. This assumption is implicit in (5)–(7).

Since the considered process is in state 0, we have at least one process in state 0. Given that at least one process is in state 0 (hence the normalization factor $1-(1-P_0)^n$), the probability that exactly i processes are in state 0 is:

$$Z_i = \frac{\binom{n}{i} P_0^i (1-P_0)^{n-i}}{1-(1-P_0)^n}, \quad \text{for } i = 1, 2, \cdots, n$$

where P_0 is the steady-state probability that a process is in state 0.

The state of the process is not affected (i.e., the number of processed uncommitted events at the process remains unchanged) if: (i) the process is not the GVT-regulator (probability $(i-1)/i$), (ii) the GVT regulator completes its event before the considered process (probability 1/2), and (iii) the GVT-regulator rolls back (Q_0). Hence the normalization factor $\left(1 - \frac{(i-1)Q_0}{2i}\right)$ occurs in the denominator of expressions (5)–(7). Only one of the above i processes is the GVT-regulator. A rollback occurs when the process is the GVT-regulator (probability $1/i$) and, upon completion, observes an event in the past (probability Q_0), or if it is not the GVT-regulator (probability $(1 - 1/i = (i-1)/i)$), completes its event before the GVT-regulator (probability 1/2) and observes an event in the past (Q_0):

$$R_0 = \sum_{i=1}^{n} Z_i \frac{\frac{Q_0}{i} + \frac{(i-1)Q_0}{2i}}{1 - \frac{(i-1)Q_0}{2i}} = \sum_{i=1}^{n} Z_i \frac{Q_0(i+1)}{2i-(i-1)Q_0}. \tag{5}$$

The forward movement of a process in state 0, F_0, occurs if it is not the GVT-regulator (probability $(i-1)/i$), completes its event before the GVT-regulator (probability 1/2), and does not observe an event in the past $(1-Q_0)$:

$$F_0 = \sum_{i=1}^{n} Z_i \frac{\frac{(i-1)(1-Q_0)}{2i}}{1 - \frac{(i-1)Q_0}{2i}} = \sum_{i=1}^{n} Z_i \frac{(i-1)(1-Q_0)}{2i-(i-1)Q_0}. \tag{6}$$

The GVT advancement for a process in state 0 occurs when either the process is GVT-regulator (probability $1/i$) and does not observe an event in the past upon completion $(1-Q_0)$, or it is not the GVT-regulator (probability $(i-1)/i$), the latter completes its event before $(1/2)$, and does not observe an event in the past $(1-Q_0)$:

$$G_0 = \sum_{i=1}^{n} Z_i \frac{\frac{1-Q_0}{i} + \frac{(i-1)(1-Q_0)}{2i}}{1 - \frac{(i-1)Q_0}{2i}} = \sum_{i=1}^{n} Z_i \frac{(i+1)(1-Q_0)}{2i-(i-1)Q_0}. \tag{7}$$

2) Transition Out of State k Into State j: After determining the collective probabilities, we now determine the individual transition rates from state k to state j. The virtual time distribution of uncommitted events at a processor has earlier been approximated by a Poisson distribution with rate $\rho\lambda$. This excludes events which are yet to be generated (holes), and includes events which will ultimately be cancelled. The virtual time distribution of committed events is Poisson $(m\lambda/n)$. The difference can be accounted for by a virtual-time Poisson distribution (corresponding to events which cause a rollback) with rate $(\rho_t + \rho_a)$ where the holes are Poisson distributed in virtual time with rate ρ_t (true messages arriving out of sequence in virtual time), and the antimessages (messages that will annihilate "false" events which currently exist) are Poisson distributed in virtual time with rate ρ_a. In addition, there will be Poisson-distributed streams in virtual time with rate ρ_f for messages, and corresponding antimessages that will arrive in the future (real time). These are the messages which have not yet arrived, but will arrive at a future time and subsequently become annihilated. The messages that arrive "in the past" (and hence cause a rollback) will be Poisson distributed in virtual time with rate $(\rho_t + \rho_a + \rho_f)$. Since the uncommitted events are Poisson distributed with rate ρ and the incoming messages "in the past" are Poisson distributed with rate $\rho_t + \rho_a + \rho_f$, the length of rollback is geometrically distributed with parameter, say,

$$p = \frac{\rho_t + \rho_a + \rho_f}{\rho}.$$

Since ρ_t, ρ_a, and ρ_f are unknown, the above expression is of little use. However, it does establish that rollback length is geometrically distributed—a fact that we will use shortly. It may be noted that since we have used M as an approximation to an infinite number of states, the geometric distribution for the rollback length is truncated.

Let $r_{k,j}$ be the probability of rollback from state k to state j. We note that a rollback of length 1 does not cause a change of state, since only the event being processed is rolled back and the number of processed uncommitted events remains unchanged. This occurs when the arriving message has a timestamp lower than the current event ($k + 1^{th}$ event) being processed, but greater than the last processed event. Thus a process rolls back from state k to state k with probability p (recall the definition of p from the previous unlabeled expression). A rollback stops at a state with probability p and continues beyond that state with probability $q = 1 - p$. Further, no arriving message will have a timestamp less than GVT. Hence the maximum length of rollback at state k is

$k+1$. Then

$$r_{k,j} = \begin{cases} R_k q^{k-j} p, & \text{for } 0 < j \le k \\ R_k q^k, & \text{for } k > 0 \text{ and } j = 0 \\ R_0, & k = 0 \text{ and } j = 0 \\ 0, & \text{for } k < j. \end{cases} \tag{8}$$

R_k is the probability of rollback from state k and is given by (2) and (5). The probability $r_{k,j}$ is due to the geometric distribution of the length of rollback.

We derive $g_{k,i}$, the probability that the GVT advances by exactly i events when the process is in state k. The timestamp increments at the processes are identically and exponentially distributed. For $k > i$ and $k > 0$, we observe that the GVT can advance in two ways (see Fig. 1):

1) The GVT moves up (probability G_k); its next event has a timestamp which lies between the timestamps of the ith and $i+1$th events of the considered process ($(\frac{1}{2})^{i+1}$, because the timestamp increments at the two processes are i.i.d. exponential) and all the remaining $(n-2)$ processes have their virtual-clock time greater than the timestamp of the ith event of the considered process. $L_{i,l+1}$ (Appendix A) is the probability that a process with l uncommitted events and processing the $l+1$th event has a virtual-time higher than the timestamp of the ith event on another process:

$$G'_{k,i} = G_k \left(\frac{1}{2}\right)^{i+1} \left(\sum_{l=0}^{M} P_l L_{i,l+1}\right)^{n-2}, \quad \text{for } k > i \ge 0.$$

2) The GVT moves up (probability G_k); its next event has a timestamp greater than the timestamp of the ith event of the process ($\frac{1^i}{2}$), and at least one of the remaining $(n-2)$ processes has a virtual-clock time that lies between the ith and $i+1$th event of the process:

$$G''_{k,i} = G_k \left(\frac{1}{2}\right)^{i} \left[\left(\sum_{l=0}^{M} P_l L_{i,l+1}\right)^{n-2} - \left(\sum_{l=0}^{M} P_l L_{i+1,l+1}\right)^{n-2}\right], \quad \text{for } k > i \ge 0.$$

Note that these two probabilities are not mutually exclusive. It is possible that the GVT moves up (G_k), the next event of the GVT-regulator lies between the ith and $i+1$th events of the process ($(\frac{1}{2})^{i+1}$), and at least one of the remaining $n-2$ processes has a virtual clock time between the ith and $i+1$th events. Hence,

$$g_{k,i} = G'_{k,i} + G''_{k,i} - G_k \left(\frac{1}{2}\right)^{i+1} \left[\left(\sum_{l=0}^{M} P_l L_{i,l+1}\right)^{n-2} - \left(\sum_{l=0}^{M} P_l L_{i+1,l+1}\right)^{n-2}\right], \quad \text{for } k > i \ge 0. \tag{9}$$

If the considered process drops from state k to 0 ($k = i > 0$), it must be that the timestamp of the next event of the GVT-regulator and the virtual clock time of all the remaining processes are greater than the timestamp of the kth event of the process:

$$g_{k,i} = \begin{cases} G_k (\frac{1}{2})^k \left(\sum_{l=0}^{M} P_l L_{k,l+1}\right)^{n-2}, & k > 0 \text{ and } i = k \\ G_0, & k = i = 0 \\ 0, & k < i. \end{cases} \tag{10}$$

Now we are ready to set up the transition rates in the Markov chain model of Fig. 2:

$$a_{k,j} = r_{k,j} + g_{k,k-j}, \qquad 0 \le j \le k \le M \tag{11}$$

$$b_{k,k+1} = F_k, \qquad 0 \le k \le M. \tag{12}$$

Informally, $r_{k,j}$ is the rate of transition out of state k and into state j due to rollback. Similarly, $g_{k,k-j}$ is the transition from state k to state j when the GVT advances by $k-j$ events. Thus $a_{k,j}$ is rate of transition from state k to state j when $0 \le j \le k \le M$. Upon forward movement (F_k), a process moves from state k to state $k+1$.

The following balance equations are obtained from Fig. 2:

$$P_k \left(b_{k,k+1} + \sum_{j=0}^{k-1} a_{k,j}\right) = P_{k-1} b_{k-1,k} + \sum_{j=k+1}^{M} P_j a_{j,k}, \quad \text{for } 0 < k < M \tag{13}$$

$$P_M \sum_{j=0}^{M-1} a_{M,j} = P_{M-1} b_{M-1,M} \tag{14}$$

$$P_0 b_{0,1} = \sum_{j=1}^{M} P_j a_{j,0}. \tag{15}$$

Note that the following is valid:

$$\sum_{j=0}^{M} P_j = 1. \tag{16}$$

C. Performance Measures

In this subsection we determine the expected length of rollback, the expected number of processed uncommitted events, the expected number of processed events above the GVT, the effective message density, the probability of rollback, the expected number of wasted events, the expected fraction of committed events, speedup, and an approximation to the probability distribution function of the process's virtual time with origin at the GVT.

The *expected length of rollback* is given by $1/p$, since rollback length is geometrically distributed with rate p.[2] Alternatively, it can be computed directly by multiplying rollback

[2] It is truncated geometric for which the mean is not $1/p$. However, the choice of a large enough M ensures that the error due to this is less than the specified tolerance ϵ.

lengths with their corresponding probabilities. Equating the two expressions for expected length of rollback, we have:

$$1/p = \frac{\sum_{i=1}^{M+1} i\left[P_{i-1}Q_{i-1}q^{i-1} + \sum_{j=i}^{M} P_j Q_j q^{i-1} p\right]}{\sum_{i=0}^{i=M} P_i Q_i}. \tag{17}$$

Note that p is unknown on both sides of the equation; however, an iterative algorithm that we present later makes use of this equation by starting with an arbitrary initial value of p, and updating the value of p on every iteration.

The *expected number of processed uncommitted events* is:

$$U = \sum_{k=1}^{M} kP_k. \tag{18}$$

The *expected number of processed events above the GVT* is:

$$T_{\text{processed}} = U + 1 \tag{19}$$

because the current event being processed will count as being processed, irrespective of whether or not a rollback occurs. This is due to the non-pre-emptive rollback assumption.

We now determine ρ, the *effective message density* of processed uncommitted events. It may be noted that although the *unprocessed* message population is greater than m due to the presence of antimessages, we require the message density of *processed* uncommitted events, since these are the ones which are liable to roll back. We recall that the effective message density of processed uncommitted events is less than m/n, since the lagging processors have not yet contributed their share of messages. Let us consider a "typical" processor. Such a processor would have half the processors ahead of it, and the other half lagging behind it. The processors which are ahead have contributed their share of messages to the typical processor. The processors which are lagging have only contributed a fraction of their share. If we ignore the messages sent by the lagging processors into "the future" of the "typical" processor, we observe that the set of messages sent out by the lagging processors to the "typical" processor is also the set of messages that caused a rollback in the latter. A "typical" rollback message is received $1/p$ events earlier than the virtual time of the "typical" processor. Since the lagging processors (half the total number of processors, approximately) have not yet contributed $\frac{1}{p}/T_{\text{processed}}$ fraction of messages, the effective message density is:

$$\rho = \frac{m}{n}\left(1 - \frac{\frac{1}{2p}}{T_{\text{processed}}}\right). \tag{20}$$

The equations for $C_{j,i}$'s (Appendix A), Q_k's (1), R_k's (2), (5), F_k's (3), (6), G_k's (4), (7), $r_{k,j}$'s (8), $g_{k,i}$'s (9), (10), transition rates (11), (12), balance equations (13)–(16), the average rollback length $1/p$ (17), the expected number of uncommitted events (18), the expected number of processed events $T_{\text{processed}}$ (19), and the effective message density ρ (20) can be solved numerically. In practice, the solution requires 3–4 iterations on ρ. More details on the solution procedure are given later.

Once the stationary probabilities are determined, other performance measures can also be computed.

The *probability of rollback* is:

$$Q = \sum_{i=0}^{M} P_i R_i. \tag{21}$$

On average, one will find $T_{\text{processed}}$ processed events at a processor above the GVT. Some of these events will be rolled back ("wasted"), while the others will be committed ("useful"). The expected number of rollbacks for this set of processed events is $QT_{\text{processed}}$. The average length of rollback is $1/p$. The expected number of "wasted" events is:

$$\text{waste} = QT_{\text{processed}}/p.$$

The *fraction of events expected to commit* is:

$$\eta = (T_{\text{processed}} - \text{waste})/T_{\text{processed}}. \tag{22}$$

Speedup is defined to be the ratio of useful events processed by n processors to the number of events processed by a single processor per unit *real* time. Since the processors are idle for a fraction of time I, the speedup is:

$$S = n\eta(1 - I). \tag{23}$$

The *distribution of processed uncommitted events in virtual time* has been approximated by a Poisson distribution with rate ρ. The virtual time of a process is the timestamp of the event it is currently processing. The virtual time of a process in state j (j processed uncommitted events above the GVT) is the timestamp of the $j+1$th event being processed. With the GVT as origin, the virtual time of a process in state j is the sum of $j+1$ i.i.d. exponential random variables with rate ρ. The probability distribution function (pdf) of a process's virtual time with the GVT as origin can be interpreted as a random sum of random variables and is given by ($\mathcal{L}^{-1}$ is the inverse Laplace transform):

$$\begin{aligned} f(t) &= \mathcal{L}^{-1}\left(\sum_{j=0}^{M} P_j\left(\frac{\rho\lambda}{\rho\lambda + s}\right)^{j+1}\right), \qquad t \geq 0 \\ &= P_0\rho\lambda e^{-\rho\lambda t} + \sum_{j=1}^{M} P_j \frac{(\rho\lambda)^{j+1} t^j e^{-\rho\lambda t}}{j!} \end{aligned} \tag{24}$$

where ρ, given by (20), is the effective message density, and $1/\lambda$ is the mean of the exponential distribution for timestamp increments.

The effect of idle processors on the performance of Time Warp can be approximated by observing that there are always at least m unprocessed messages in the system. There might be more, since message annihilation is not immediate (due to the non-pre-emptive nature of the model). We ignore this fact in the analysis which follows and assume that there are always exactly m unprocessed messages. Using the urn model, a processor is idle if all of the m messages (balls) are distributed

```
PROCEDURE

input number of processes ($n$), tolerance ($\epsilon$), message population ($m$)
set $\rho$ = m/n. /* equation ( 20) */
set I = $(\frac{n-1}{n})^m$ /* equation ( 25) */
set p to some initial value; q = 1 - p. /* equation ( 17) */
set $P_i$'s to any initial value such that sum is 1. /* equations ( 13- 16) */
compute L /* Appendix A */
set$\eta_{old} = \eta_{new}$ = 0 /* fraction of committed events */
set $S_{old} = S_{new}$ = 0 /* speedup */
do {
        $S_{old} = S_{new}$
        $\eta_{old} = \eta_{new}$
        compute C /* Appendix B, makes use of $\rho$ */
        /* $\gamma = \rho$ is the effective
        message density at this iteration */
        do {
                compute $Q_k$'s /* equation ( 1) */
                compute $a_{k,j}$ and $b_{k,k+1}$ /* equation ( 11- 12 */
                update $P_i$'s /* equations  13 -  16 */
                update p /* equation  17 */
        } while difference between any $P_i$'s is greater than $\epsilon$
        compute $\eta_{new}$, $S_{new}$ /* equations ( 22,  23) */
        update $\rho$ /* equation ( 20) */
        update $I$ /* equation ( 25) */
}while $S$ and $\eta$ not within desired accuracy
print $S$, $\eta$, and rollback probability
```

Fig. 4. Algorithm for computation performance measures.

over the remaining $n - 1$ processors (urns). The probability that a processor is idle can be estimated by:

$$I = \frac{(n-1)^m}{n^m}. \tag{25}$$

In the analysis in previous sections we have assumed that the probability of a processor being idle is independent of whether or not the other processors are idle. This approximation is good for the usual range of message densities, but breaks down when the message density is very small. In the latter case, knowing that a processor is not idle significantly increases the probability of other processors being idle, since there are very few messages in the system.

D. Algorithm

In the following we outline the procedure to solve the system of equations. Even though the highly interdependent set of coupled nonlinear equations appears formidable, the following procedure computes the performance measures efficiently by utilizing the mutual dependence and structure of the expressions derived in the earlier subsections. The number of equations is the same as the number of unknown variables. Since some of the equations are nonlinear, the possibility that the system has more than one solution is not ruled out. However, the procedure is iterative and our tests indicated that numerical solutions converged to the same value irrespective of the initial values assumed. Although the intermediate expressions are not listed, all labeled equations except (24) are necessary for solution of the system. The procedure is presented in Fig. 4.

In the experiments, the tolerance (ϵ) was set to 0.001. The final performance measures are accurate to at least two significant figures. The solution required 3–4 iterations of the outer loop. Each iteration of the outer loop corresponds to approximately 100 iterations of the inner loop. Computation of $C_{j,i}$ is the more expensive portion which, fortunately, is in the outer loop. This procedure must be run for a large enough value of M so that the error in accuracy of the results is within tolerance. For a tolerance of 0.001, a message density of 1, 4, and 16 approximately required (depending on the number of processors) M to be 10, 30, and 60, respectively. The complexity of the procedure depends most strongly on the value of the parameter M.

IV. Numerical Results and Validation

The analytic model makes a number of assumptions in order to make the analysis tractable: zero communication delays, unbounded buffers, etc. Measurements of a Time Warp program running on a shared memory multiprocessor (specifically, a GP-1000 BBN Butterfly) were made and compared with the performance predicted by the analytic model in order to test the significance of these assumptions and the validity of the approximations underlying the analysis. The Time Warp program is described in more detail in [5].

The assumptions used in the analytic model pertaining to the application program (exponential time per event, exponential timestamp increment, fixed message population, etc.) correspond to a specific instance of the parallel HOLD (PHOLD) workload model [6]. PHOLD was used in the experiments performed here. While this satisfies certain assumptions about the Time Warp program (e.g., exponential timestamp increment, random message routing, fixed message population, and others), assumptions such as zero rollback time and communication delay still do not hold.

Fig. 5 compares the analytical estimate for fraction of messages which are eventually committed (useful messages) η with experimentally obtained values. We observe that for very low message densities, the fraction of useful messages predicted by the analytical model is less than that which was observed experimentally. This can be attributed to the approximation in estimating the fraction of time that a processor is idle. Consider the following: with a message population of 1 (message density $1/n$), 100% of the processed events must be committed since no rollback can occur. In this case, the analytic model assumes that each of the processors is idle with probability $((n-1)/n)$, *independent* of whether or not the other processors are idle. In the analytic model it is probabilistically possible for several processors to be processing simultaneously; rollback can still occur and the predicted fraction of committed messages is less than that which is observed. Initially, as the message density is increased, the number of rollbacks increases, which reduces the observed fraction of committed events. Later, however, other effects come into play which cause the fraction of committed events to increase with message density. These effects are discussed below. For this reason we observe a "dip" in the observed fraction of committed events when the message density is low.

For higher message densities (Fig. 5), estimation of processor idleness is less critical since the processors are busy a larger fraction of the time. We observe that as the message density is increased, generally better agreement is obtained between the predicted and measured fraction of committed events.

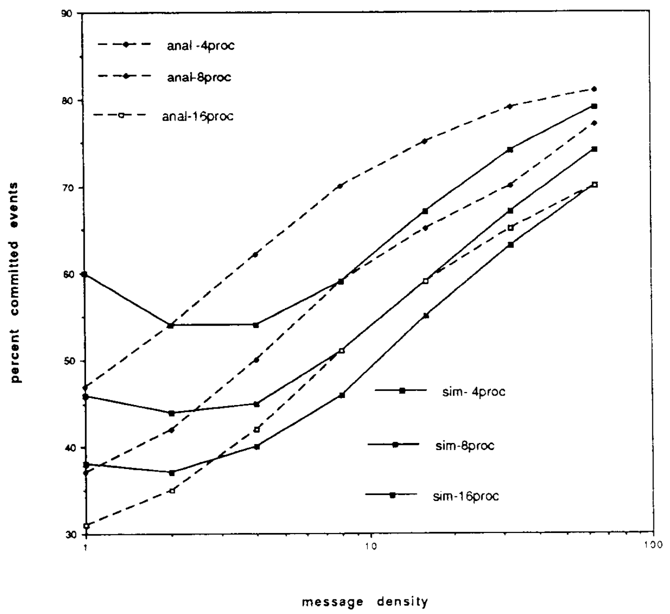


Fig. 5. Analytical and experimental results for η.

At moderate-to-high message densities the analytic model overestimates performance. We believe that the principal cause of inaccuracy is the assumption that rollback and message cancellation requires negligible time. Nonzero rollback cost causes overly optimistic processes to advance further into the future, processing more incorrect events than they would have had rollback required no time. This is because the rollback "wave" takes a nonzero time to propagate forward and "catch up" with the incorrect messages. Another assumption which contributes to the discrepancy is negligible communication delay. Since the communication delay is nonzero, a message might be in the future of the receiving processor when it was sent, but in the past of the receiving processor when it arrives. This contributes to increased rollback and hence a reduced fraction of committed events. The model assumes these overheads to be negligible, so it predicts better than actual performance. These effects are less prevalent for low message populations because, as noted earlier, processes tend to become idle rather than incorrectly advancing forward, and message traffic is lower.

For very high message densities, the finite approximation to infinite buffering that is used in the analytic model becomes more important. This leads to numerical errors in the computation of $C_{j,i}$ for large j, i. Stirling's approximation to factorials were used to reduce the size of intermediate calculations (Appendix B). The effect of this error on the final results was approximately 0.1% for a message density of 16, 02.–0.6% for a message density of 32, and 0.9–3.9% for a message density of 64. The errors were higher for larger number of processors. Overall, the effect was to overestimate $C_{j,i}$, which means that the fraction of useful events predicted by the model is conservative (low).

Analytical and experimental results for speedup are shown in Fig. 6. For very low message densities, the analytical model predicts a lower fraction of committed events, as discussed earlier—this leads to lower speedup predictions for low message densities. For higher message densities, speedup plots are similar to the plots for a fraction of committed events due to assumptions of zero rollback and communication costs. The ball and urn model assumes that there are m messages in the system when, in fact, there are more. Hence we overestimate the fraction of time when the processes are idle. In this sense the speedup curves are conservative and hence suitable for design purposes.

An analytical estimate for the fraction of committed events is shown as a function of the number of processors in Fig. 7. The larger the message density, the higher the inherent parallelism leading to better performance. For a constant message density and increasing number of processors, the fraction of committed events drops rapidly at first, and then stabilizes. Initially, adding another processor increases the probability of rollback sharply, but later the effect is only marginal.

Analytical estimate for the dependence of speedup on the number of processors is shown in Fig. 8. The larger the message density, the larger the inherent parallelism which leads to higher speedup. As the message density is increased the speedup plot approaches the maximum speedup possible, viz. speedup = n, the number of processors. As the message density approaches infinity, speedup is ideal (equal to number of processors). This is because the timestamp increment will be Poisson distributed in virtual time with finite rate λ, while the effective message density will be Poisson distributed with rate $\rho\lambda$, where $\rho -> \infty$. By setting $\gamma = \infty$ in the expression for $C_{j,i}$ (in Appendix B, $\alpha = \infty$ and $\beta = \lambda$), we obtain

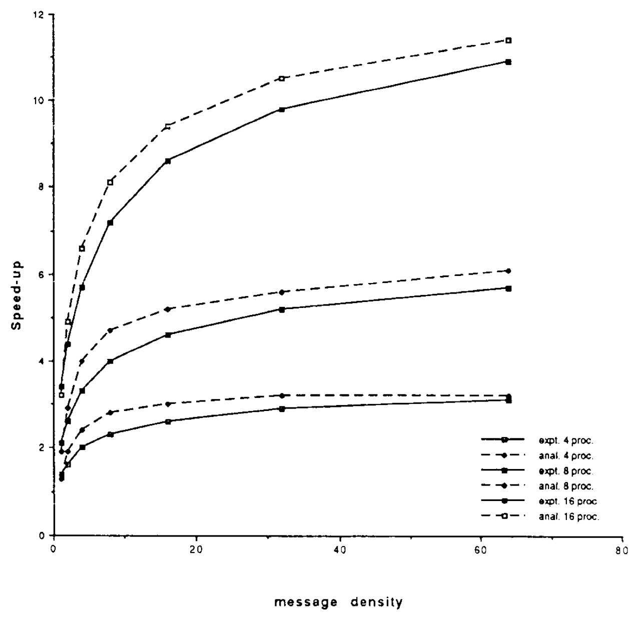


Fig. 6. Analytical and experimental results for speedup.

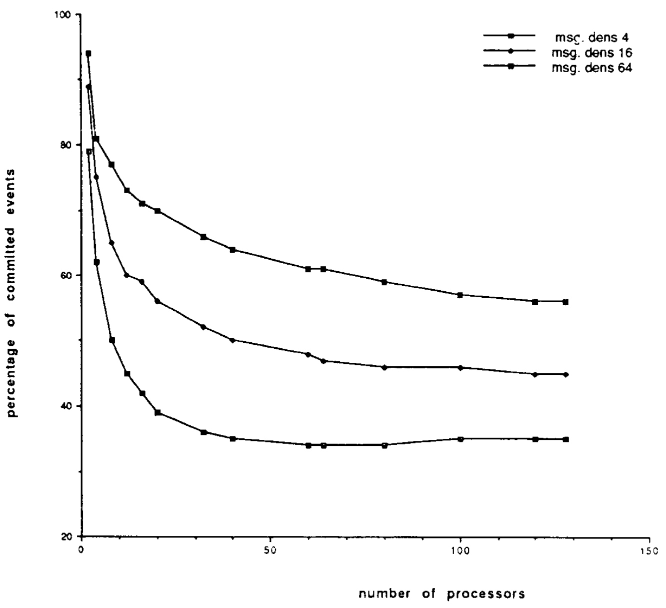


Fig. 7. Analytical estimate for the fraction of committed events.

$C_{j,i} = 0$ for all j, i. This implies that the rollback probability is 0 and the fraction of committed events is 1, from which speedup is equal to n. Intuitively, for an infinite message density, a processor needs to have processed infinitely more events than another processor in order for the latter to roll back the former. This has zero probability, and so the probability of rollback is zero. For a constant message density, Fig. 8 shows the scalability of the processors.

It is illuminating to compare out analytical results with simulations in [3]. Felderman's simulation uses a self-initiating model (as in [21]) where messages are only used for synchronization. Fig. 8 includes a line for his simulation speedups. His model assumes discrete (integer) virtual times, and exponential processing times. Although in their model the timestamp increment is geometrically distributed with rate $1/\beta$, the speedup shown in Fig. 8 corresponds to $\beta = 1$; i.e., a processor

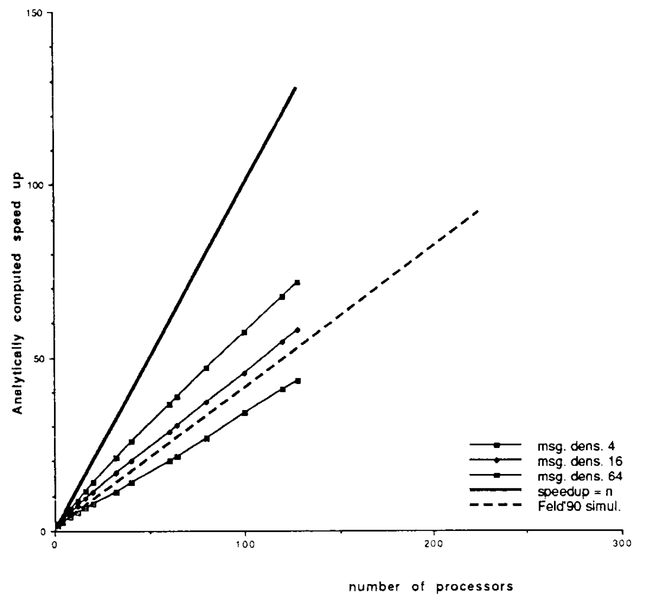


Fig. 8. Analytical estimate for speedup.

increments its integer clock by exactly one ($v = v + 1$) upon completion of an event. In his model a processor independently executes an event, advances its local clock upon completion, and then sends a message to one randomly chosen processor. The timestamp on the message is the processor's local clock time after processing the message. Let us now interpret his model in terms of our analytic model. The interdistance between events processed by the same process is at least 1. The timestamp increment is always 1. In our model this corresponds to the condition that $\rho\lambda \geq 1$ and $\lambda = 1$ or $\rho \geq 1$. It may be noted that there is a difference between the models which is not accounted for—Felderman's simulation has timestamp increments of 1, while timestamp increments in our model for the above conditions are exponentially distributed with rate 1. For the effective message density ρ to be 1, the message density m/n should be a real number greater than 1 (in the range of 1.5–2.0). It comes as no surprise, therefore, that Felderman's simulation results lie very close to our plot for low message densities. That Felderman's model estimates a lower speedup than our plots for larger message densities can be understood in terms of lookahead. His model has no lookahead (the timestamp of the event is the same as the virtual clock of the sender). This is a characteristic of self-initiating models. If the event just processed is rolled back, there is a good chance that the message which was sent would also have been processed (because it has a low timestamp) leading to cascaded rollbacks. On the other hand, in our model the message has a timestamp which is computed by adding a random variable to the virtual clock of the processor. If the event just processed is rolled back, there is a good chance that the message which was sent would not have been processed (because it has a lookahead corresponding to the random variable added). Cancellation of the message will probably not result in a cascaded rollback.

V. Conclusions and Future Work

We have derived and evaluated a discrete state, continuous time Markov model for Time Warp. We have determined the dependence of fraction of useful events and speedup on the message population when n processes limit each others' rate of progress by passing messages between each other. These messages may cause rollbacks and waste some of the work that a process has accomplished. In addition, we have derived expressions for several measures which characterize the dynamics of Time Warp, such as the expected number of processed uncommitted events at a process, the expected number of these events which will be rolled back, the rollback probability as a function of the number of processed uncommitted events, the pdf of the process's virtual clock time, and the pmf for the number of processed uncommitted events at a process. We have approximately determined the fraction of time a processor is expected to be idle and have used it to arrive at a better estimate of speedup. These have been determined for exponential task times, negligible rollback time and communication delay, and unbounded buffer, homogeneous processes.

The principal contribution is the analysis of the n process case with a straight-forward Markovian model. The close agreement between the analytical and experimental results demonstrate the validity of the approximations. Our model has successfully analyzed cascaded rollbacks where complex earlier analyses had stumbled. The difficulty in analyzing cascaded rollbacks was perhaps the single most important factor which prevented extensions to the earlier two-processor analyses. In addition, to our knowledge this is the first analysis to be backed by performance measurements of a Time Warp prototype.

Possible generalizations include the following:

- Extending the model to heterogeneous processors
- Extending the analysis to general (nonexponential) time-stamp distributions
- Incorporating overheads such as rollback costs, communication delays, and state saving costs. Use could be made of results presented in [8] to construct an analytical model where messages (antimessages) correspond to positive (negative) customers
- Investigating the effect of finite buffer on the performance of Time Warp.

Appendix A
Derivation of $L_{j,i}$

In this appendix we derive $L_{j,i}$—the probability that the sum of j independent random variables is less than the sum of i independent random variables, all of which are exponentially distributed with the same rate λ. Alternatively, $L_{j,i}$ can also be viewed as the probability that Erlang with j phases is less than the Erlang with i phases. Let $t_1(t_2)$ denote the random sum of $j(i)$ random variables. Then ($\mathcal{L}^{-1}$ is the inverse Laplace transform):

$$L_{0,i} = 1, \qquad i = 1, 2, \cdots \tag{26}$$

$$L_{1,i} = 1 - 1/2^i, \qquad i = 1, 2, \cdots \tag{27}$$

$$L_{j,i} = \int_0^\infty \mathcal{L}^{-1}\left(\frac{\lambda}{\lambda+s}\right)^i \left[\int_0^{t_2} \left(\mathcal{L}^{-1}\left(\frac{\lambda}{\lambda+s}\right)^j\right) dt_1\right] dt_2, \qquad j, i = 1, 2 \cdots$$

$$= \int_0^\infty \frac{\lambda^i t_2^{i-1} e^{-\lambda t_2}}{(i-1)!} \left[\int_0^{t_2} \frac{\lambda^j t_1^{j-1} e^{-\lambda t_1}}{(j-1)!} dt_1\right] dt_2$$

$$L_{j,i} = 1 - \sum_{k=1}^{j} \binom{i+k-2}{k-1} \left(\frac{1}{2}^{i+k-1}\right). \qquad \text{for } j, i = 1, 2 \cdots. \tag{28}$$

Appendix B
Derivation of $C_{j,i}$

In this appendix we derive $C_{j,i}$—the probability that the sum of j random variables with rate α, and a random variable with rate β, is less than the sum of $i+1$ random variables with rate α. All the random variables are independent and exponentially distributed. Let t denote the sum of j random variables with rate α, and one random variable with rate β. Similarly, let t' denote the sum of $(i+1)$ random variables with rate α. Then

$$C_{j,i} = L_{j+1,i+1}, \qquad \text{for } \alpha = \beta \tag{29}$$

$$C_{0,i} = \int_0^\infty \frac{\alpha^{i+1} t'^i e^{-\alpha t'}}{i!} \left[\int_0^{t'} \beta e^{-\beta t} dt\right] dt'$$

$$= 1 - \left(\frac{\gamma}{\gamma+1}\right)^{i+1}, \qquad \text{for } i = 0, 1, 2, \cdots \tag{30}$$

where $\gamma = \alpha/\beta$.

$$C_{j,i} = \int_0^\infty \mathcal{L}^{-1}\left(\frac{\alpha}{\alpha+s}\right)^{i+1} \left[\int_0^{t'} \cdot \left(\mathcal{L}^{-1}\left(\left(\frac{\alpha}{\alpha+s}\right)\frac{\beta}{\beta+s}\right)\right) dt\right] dt', \qquad \text{for } \alpha \neq \beta$$

$$= \int_{t_1=0}^\infty \int_{t_2=0}^\infty \int_{t_3=t_1+t_2}^\infty \frac{\alpha}{i!}(at_3)^i e^{\alpha t_3} \beta e^{-\beta t_2} \frac{\alpha}{(j-1)!} \cdot (\alpha t_1)^{j-1} e^{-\alpha t_1} dt_3\, dt_2\, dt_1$$

$$= \int_{t_1=0}^\infty \int_{t_2=0}^\infty \left(\sum_{k=0}^{i} \frac{\alpha^k (t_1+t_2)^k}{k!} e^{-\alpha(t_1+t_2)}\right) \cdot \beta e^{-\beta t_2} \frac{\alpha}{(j-1)!} (\alpha t_1)^{j-1} e^{-\alpha t_1} dt_2\, dt_1$$

$$= \sum_{k=0}^{i} \int_{t_1=0}^\infty \frac{\alpha^{j+k}\beta}{(j-1)!} t_1^{j-1} e^{-2\alpha t_1} \int_{t_2=0}^\infty \cdot \frac{(t_1+t_2)^k}{k!} e^{-(\alpha+\beta)t_2} dt_2\, dt_1$$

$$= \sum_{k=0}^{i} \int_{t_1=0}^\infty \frac{\alpha^{j+k}\beta}{(j-1)!} \cdot t_1^{j-1} e^{-2\alpha t_1} \cdot \left\{\frac{1}{(\alpha+\beta)^{k+1}} \sum_{l=0}^{k} \frac{[(\alpha+\beta)t_1]^l}{l!}\right\} dt_1$$

$$= \sum_{k=0}^{i} \sum_{l=0}^{k} \frac{a^{j+k}\beta(j+l-1)!(\alpha+\beta)^l}{(j-1)!\, l!(\alpha+\beta)^{k+1}(2\alpha)^{j+l}}$$

$$= \sum_{k=0}^{i} \sum_{l=0}^{k} \binom{j+l-1}{j-1} \left(\frac{1}{2}\right)^{j+l} \frac{\gamma^{k-l}}{(1+\gamma)^{k-l+1}} \tag{31}$$

where $\gamma = \alpha/\beta$; $j = 1, 2, \cdots$; $i = 0, 1, \cdots$.

Acknowledgment

The authors would like to thank D. Nicol, L. Kleinrock, R. E. Felderman, and J. Lin for their insightful comments and criticisms on an earlier version of this paper.

References

[1] K, M. Chandy and J. Misra, "Asynchronous distributed simulation via a sequence of parallel computations," *Commun. ACM,* vol. 24, no. 11, pp. 198–206, Nov. 1981.

[2] R. E. Felderman and L. Kleinrock, "An upper bound on the improvement of asynchronous versus synchronous distributed processing," in *Proc. SCS Multiconf. Distributed Simulation,* Jan. 1990, vol. 22, no. 1, pp. 131–136.

[3] R. E. Felderman, "Speedup for large number of processors," private communication, Dec. 1990.

[4] R. E. Felderman and L. Kleinrock, "Two processor time warp analysis: some results on a unifying approach," *Proc. SCS Workshop on Parallel and Distributed Simulation,* Jan. 1991, vol. 23, no. 1, pp. 3–10.

[5] R. M. Fujimoto, "Time warp on a shared memory multiprocessor," *Trans. Soc. Comput. Simul.,* vol. 6, no. 3, pp. 211–239, July 1989.

[6] R. M. Fujimoto, "Performance of time warp under synthetic workloads," in *Proc. SCS Multiconf. Distributed Simulation,* Jan. 1990, vol. 22, no. 1, pp. 23–28.

[7] R. M. Fujimoto, "Parallel discrete event simulation," *Commun. ACM,* vol. 33, no. 10, pp. 30–53, Oct. 1990.

[8] E. Gelenbe, "Product form networks with negative and positive customers," *J. Appl. Prob.,* to be published.

[9] A. G. Greenberg, B. D. Lubachevsky, and I. Mitrani, "Unboundedly parallel simulations via recurrence relations," *Proc. 1990 ACM SIGMETRICS Conf. on Measure. and Model. Comput. Syst.,* May 1990, vol. 18, no. 1, pp. 11–12.

[10] D. R. Jefferson, "Virtual time," *ACM Trans. Program. Languages and Syst.,* vol. 7, pp. 404–425, 1985.

[11] D. Jefferson and A. Witkowski, "An approach to performance analysis of timestamp-driven synchronization mechanism," in *Proc. 3rd ACM Ann. Symp. Principles Distributed Comput.,* 1984.

[12] L. Kleinrock, "On distributed systems performance," *Comput. Networks ISDN J.,* vol. 20, nos. 1–5, pp. 209–216, Dec. 1990.

[13] L. Lamport, "Times, clocks, and the ordering of events in a distributed system," *Commun. ACM,* vol. 21, no. 7, pp. 558–565, 1978.

[14] S. Lavenberg, R. Muntz, and B. Samadi, "Performance analysis of a rollback method for distributed simulation," *Performance '83.* Amsterdam: North-Holland, 1983, pp. 117–132

[15] Y. B. Lin and E. D. Lazowska, "Optimality considerations for "time warp" parallel simulation," in *Proc. 1990 SCS Multiconf. Distributed Simulation,* Jan. 1990, pp. 29–34.

[16] R. J. Lipton and D. W. Mizell, "Time warp vs. Chandy-Misra: a worst-case comparison," in *Proc. SCS Multiconf. Distributed Simulation,* Jan. 1990, vol. 22, no. 1, pp. 137–143.

[17] B. Lubachevsky, A. Shwartz, and A. Weiss, "Rollback sometimes works . . . if filtered," in *Proc. 1989 Winter Simulation Conf.,* Dec. 1989, pp. 630–639.

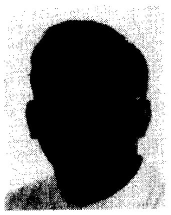
[18] V. Madisetti, J. Walrand, and D. Messerschmitt, "Synchronization in message passing computers: models, algorithms, and analysis," in *Proc. SCS Multiconf. Distributed Simulation,* Jan. 1990, vol. 22, no. 1, pp. 35–48.

[19] J. Misra, "Distributed discrete-event simulation," *Comput. Surveys,* vol. 18, no. 1, pp. 39–65, 1986.

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

[21] D. M. Nicol, "Performance bounds on parallel self-initiating discrete-event simulations," *ACM Trans. Model. Comput. Simulation,* vol. 1, no. 1, pp. 24–50, Jan. 1991.

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



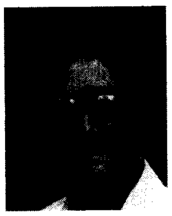
Anurag Gupta was born in Bareilly, India, in 1966. He received the Bachelor of Technology degree (Hons.) in computer science and engineering from the Indian Institute of Technology, Kharagpur, in 1988, the M.S. degree in computer science from the University of Rhode Island in 1989, and is currently working towards the Ph.D. degree in computer science at the College of Computing of the Georgia Institute of Technology, where he has been active in the Time Warp project.

He spent the Summer of 1991 at the DEC Systems Research Center in Palo Alto, CA, working on the AUTONET project. His research interest is in performance evaluation (modeling, analysis, simulation, and optimization) of computer systems.

Ian F. Akyildiz (M'85–SM'89) received the B.S., M.S., and Doctor of Engineering degrees in computer engineering from the University of Erlangen–Nuremberg, Germany, in 1978, 1981, and 1984, respectively.

Currently, he is an Associate Professor in the College of Computing, Georgia Institute of Technology, Atlanta. He is an Associate Editor for *Computer Networks and ISDN Journal.* His research interests are in performance evaluation, parallel simulation, computer networks, distributed systems, and computer security.

Dr. Akyildiz is an ACM Lecturer and a member of the ACM (Sigmetrics, Sigops and Sigcomm).



Richard M. Fujimoto (S'78–M'83) received the B.S. degrees in computer science and computer engineering from the University of Illinois, Urbana in 1977 and 1978, and the M.S. and Ph.D. degrees from the University of California, Berkeley in 1980 and 1983, respectively.

He is an Associate Professor in the College of Computing at the Georgia Institute of Technology, Atlanta. Prior to this, he was an Assistant Professor in the Computer Science Department of the University of Utah. His current research interests include computer architecture, parallel processing, and simulation. He is currently an Area Editor for *ACM Transactions on Modeling and Computer Simulation.*