Analysis of Optimistic Window-based Synchronization [*]

Phillip M. Dickens
ICASE

David M. Nicol
College of William and Mary

Paul F. Reynolds, Jr.
University of Virginia

J. M. Duva
University of Virginia

April 11, 1994

Abstract

This paper studies an analytic model of parallel discrete-event simulation, comparing the costs and benefits of extending optimistic processing to the YAWNS synchronization protocol. The basic model makes standard assumptions about workload and routing; we develop methods for computing performance as a function of the degree of optimism allowed, overhead costs of state-saving, rollback, and barrier synchronization, and LP aggregation. This allows an approximation-based analysis of the range of situations under which optimism is a beneficial extension to YAWNS. We find that limited optimism is beneficial if the processor load is sparse, but that aggregating LPs onto processors improves YAWNS relative performance.

[*]This work was supported by the National Aeronautics and Space Administration under NAS1-19480 while Dickens and Nicol were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Hampton, VA 23681. Nicol's work was additionally supported by Carleton College while he was resident as a visiting research associate, and by NSF grant CCR-9201195.

1 Introduction

Discrete-event simulations model physical systems. The literature on parallel discrete-event simulation (PDES) usually views a physical system as a set of communicating physical processes, each of which is represented in the simulation by a logical process (LP). LPs communicate through time-stamped messages reflecting changes to the system state. A time-stamp reflects an instant where a state change occurs in the physical process model.

Parallel discrete event simulation poses difficult synchronization problems, due to the underlying sense of logical time. Each LP maintains its own logical clock representing the time up to which the corresponding physical process has been simulated. The fundamental problem is to determine when an LP may execute a known future event, and in so doing advance its logical clock. If an LP advances its logical clock too far ahead of any other LP in the system it may receive a message with a time-stamp in its logical past, called a straggler. The threat of stragglers is dealt with by saving the simulation state periodically, and rolling back as appropriate when a straggler arrives . Messages sent at times ahead of the straggler's time-stamp must be undone. Fundamental problems of PDES are reviewed in Misra (1986), Fujimoto (1990), Righter and Walrand (1989). Nicol and Fujimoto (1994) give a more current state-of-the-art review.

Most PDES synchronization protocols fall into two basic categories. *Conservative* protocols (e.g. Chandy and Misra 1979, Bryant 1979, Peacock, Wong and Manning 1979, Lubachevsky 1988, Chandy and Sherman 1989 and Nicol 1993) do not allow an LP to process an event with time-stamp t if one is unable to assert that it will not receive another event with time-stamp less than t at some point in the future. *Optimistic* protocols (e.g. Time Warp, Jefferson (1985)) allow an LP to process an event before it is known for certain that the LP will not later need to process an event with earlier time-stamp. Causality errors are corrected through a rollback mechanism. A more careful taxonomy of protocol characteristics is detailed in Reynolds (1988); in keeping with standard (but imprecise) practice, we will speak in terms of conservative and optimistic protocols.

The earliest synchronization protocols are asynchronous—an LP synchronizes solely on the basis of interactions with LPs with which it directly communicates. Recently more synchronous protocols have attracted interest. While details vary, the basic idea is to incorporate barrier synchronizations and global reductions on functions of future simulation times. Examples include Moving Time Window (Sokol *et al.* 1988), Conservative Time Windows (Ayani and Rajaei, 1992), Conditional Events (Chandy and Sherman, 1989), Bounded Lag (Lubachevsky, 1988), Synchronous Relaxation (Eick *et al.*, 1993), Bounded Time Warp (Turner and Xu, 1992), Breathing Time Buckets (Steinman, 1991), and YAWNS(Nicol, 1993). The advantage to a conservative protocol is that synchronization information moves quickly through the system, lowering overhead costs. This efficiency usually comes at the

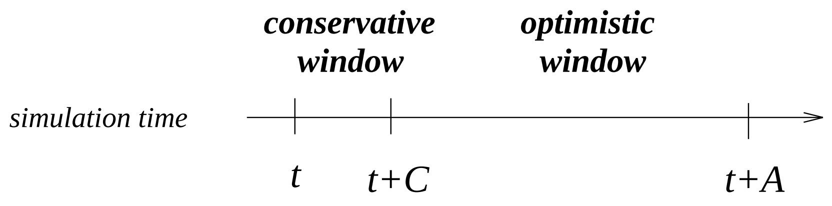

Figure 1: Extended YAWNS window is comprised of one conservative and one optimistic subwindow.

price of more pessimistic synchronization, e.g., an LP A may block against the threat of a receiving a message at time t, whereas the threatened message is actually from LP B to LP C. The global mechanisms allow for efficient computation of simulation times, like t, but do not handle routing details well. The advantage to an optimistic protocol is the elimination of a separate GVT (Global Virtual Time) calculation, and the reduction of the risk of cascading rollbacks. As for the conservative methods, the price paid is the reduction of asynchrony, and more limited opportunities for parallelism.

Our interest is in the conservative YAWNS protocol, and in determining conditions under which it makes sense to extend it by incorporating optimism. YAWNS conservatively constructs a window of simulation time within which events on different processors may be concurrently simulated (details follow in Section 2). This *conservative window* tends to be small. However, under the YAWNS mechanism, an LP that executes an event outside of the window risks receiving a straggler message. We extend optimism to YAWNS by appending an *optimistic window* to the conservative window; when an LP executes events in the optimistic window it must be prepared to deal with straggler messages. The advantage is to increase the number of events processed per window, in hopes of lowering the amortized cost of of computing and synchronizing at the window. The cost of the extension is due to state-saving, rollback, and additional global synchronization. The basic form of the extension is illustrated in Figure 1—at simulation time t all LPs use the standard YAWNS mechanism to compute the conservative window $[t, t+C)$, but also append an optimistic window $[t+C, t+A)$. Processors synchronize globally at simulation time $t+A$. For our purposes we take A as a user-specified parameter that governs the degree to which optimistic execution is permitted. We will call the method YOW (YAWNS Optimistic Window).

We find that there is a best optimistic window size that is much larger than YAWNS's. We derive formulas for YAWNS' and YOW's performance as a function of synchronization, state-saving, and event-reprocessing costs. Using these we determine that when the problem is sparse—one fine-grained LP per processor—then asymptotically (as the number of LPs increases) then YOW prevails. However, if we fix the size of the architecture and aggregate LPs onto processors, then YAWNS can prevail.

The remainder of the paper is organized as follows. Section 2 describes our analytic model and its relationship to others in the literature. Section 3 develops methods for approximating the probability distribution of an LPs workload, included reprocessed messages due to roll backs. Section 4 applies those approximations to compare conservative YAWNS with its optimistic extension, and Section 5 presents our conclusions.

2 Model

Our analysis is of a parallelized queueing network simulation. LPs represent servers, and events occur when jobs either enter service, or are received by a queue. A job's random service time increases its LP's simulation clock by the service amount. Event processing cannot be interrupted; furthermore a job's post-service destination is presumed to be known at the time it enters service. The destination is chosen uniformly at random from the set of all LPs. We do assume that the data content and next destination of a serviced job depend upon the contents and times of all jobs received by the LP prior to the event entering service. Because of this, a message reporting the job's arrival at its new destination is sent to its recipient at the time the job enters service. This is called *pre-sending* the job, and is an important aspect of both YAWNS and Time Warp. A message has both a *send-time* and *receive-time*, corresponding to the service-entry and service departure times. Service time (reflecting an advancement in simulation time) is also random, and is exponentially distributed with rate μ_s. We assume that the cost of processing a service-entry event or a job arrival event is unity; our expression of physical execution times will be in these units.

While simple, models like there are the basis for several analytic studies. This model is similar to the one studied by Gupta, Akildiz and Fujimoto (Gupta *et al.,1991*) (which we'll refer to as GAF) in their study of asynchronous Time Warp. The main differences are that we assume unit cost for executing an event and the GAF model assumes an exponentially distributed execution cost; that our model is basically that of a queueing system with single servers and a non-preemptive queuing discipline whereas the GAF model is of a queuing system with infinite servers; our model indirectly reflects the effects of communication delay, and the GAF model assumes instantaneous communication. These differences are significant enough to prevent us from quantitively comparing our model results to GAF's. We do note that GAF's assumption of random event costs should tend to worsen performance over our model, but the instantaneous communication and infinite servers should tend to improve it over ours. Furthermore, one increases the available parallelism in the GAF model by increasing the number of messages; in our model one must increase the number of LPs.

Our model is also loosely related to the self-initiating model studied by Nicol (1991), and is subsumed by Nicol's message-initiating model in his study of YAWNS (Nicol, 1993). The former model concentrates on the effects of fan-outs greater than one, and ignores the effects of rollback; the latter model provides the analysis of YAWNS that we use in this paper. The bonding model of Eick *et al.* (1993) is closely related to ours, in that it essentially describes the behavior of a parallelized queuing simulation identical to ours *except* that a message describing a job's departure is sent only at the simulation instant when the job departs—we assume the message is sent at the time the job enters service. Another difference is that we assume that a re-executed event may send its subsequent message to a different LP than before, whereas the bonding model assumes it is directed to the same LP. Finally, the randomly uniform routing assumption is shared by the model studied by Felderman and Kleinrock (1991), who make different assumptions concerning time-stamp advancement and event execution time.

Our analysis is unique in several ways. First, nearly all of the afore-mentioned models regard communication, state-saving, and synchronization as negligible. We believe that these same costs largely define which synchronization approach is best suited for a problem, and so should be explicitly incorporated in the model. Secondly, our analysis is of an optimistic window-based scheme where performance depends on the level of optimism; in this regard only Eick *et al.'s* model is similar. Our analytic approach is different, but can also be extended to the Eick *et al.* model. While we apply the model to the problem of extending YAWNS, the approach applies more generally to the analysis of window-based optimistic protocols. Finally, only the analysis in Nicol (1993) considers the beneficial effects of aggregating LPs; as we shall see, this consideration can make it more advantageous to fore-go optimism in a sufficiently aggregated case, whereas optimism is better in the non-aggregated case.

Our analytic approach is computational and is based on simplifying approximations. We develop an intuitive approximation to the probability distribution of the number of events processed by an LP while executing a window. The workload distribution includes reprocessed events induced by rollbacks. With this distribution as a basis we add overhead costs, and compute the average execution cost (in real time) per unit simulation time.

Before proceeding to the analysis, it is useful to review the YAWNS mechanism. Presume that all LPs have executed all events up to simulation time t. Under the assumptions that permit pre-sending messages, each LP i can examine its state and predict the departure time $d_i(t)$ of the next job to receive service, excluding the one receiving service at t, assuming no further message arrivals at i prior to that job entering service. This sort of lookahead is called *conditional knowledge* by Chandy and Sherman (1989), because the validity of $d_i(t)$ is conditional. Using standard minimum reduction techniques, the LPs can quickly compute $w(t) = \min_i\{d_i(t)\}$; the conservative YAWNS window is $[t, w(t))$. By construction, no job entering service during the $[t, w(t))$, window also departs service. Coupling this feature with message pre-sending, no message generated by an event in $[t, w(t))$ has a receive time in $[t, w(t))$.

We extend optimism to YAWNS by requiring that LPs synchronize at the upper edge of the optimistic window. The same min-reduction as before is used to synchronize, only now the result $w(t)$ indicates the upper edge of a "safe region" into which no straggler message will ever venture. Only one global synchronization is needed per window—a min-reduction serves both to synchronize the processors at some time t, and to compute $w(t)$. The processors know to synchronize again at time $t + A$. No state-saving need be performed prior to any event in the conservative window.

One essential difference between YOW and Bounded Time Warp (Turner 1992) is the computation and exploitation of the conservative window. Another is the proposed synchronization mechanism. Special care must be taken when synchronizing at $t + A$ since an LP may simulate up to that time but then be rolled back. Bounded Time Warp proposes an essentially linear time (in the number of LPs) termination detection mechanism. More efficient methods can be supported in hardware (Reynolds *et al.* 1993), or using special algorithms in software (Nicol 1993a). Our model presumes Nicol's software solution.

We assume that an event at LP i which is reprocessed due to rollback randomly selects a new destination with each reprocessing. This reflects the assumption that a message's content and destination is a sensitive function of the complete message history at LP i up to the time where the job enters service. Thus two messages are generated upon reprocessing an event, an anti-message to cancel its previous routing and a new routing message sent to another (probably different) LP. Like other analyses of Time Warp, we assume that the anti-message exacts no computational cost at either the sender or receiver. However, our model will not assume that an anti-message is received instantaneously. We do not model the message delay directly, but rather model the effects of such a delay.

3 Analysis

Within any given window of width A, an LP will execute (and re-execute) a random number of events, say W. This random variable (like all others in our analysis) depends on A, but this dependence will not be expressed in the notation. Our initial goal is to determine the probability distribution of W; note that this distribution is the same for all LPs under the uniformity assumptions made. Given the distribution, we can add overhead and execution costs, and determine the mean time μ_A required to complete the window by the processor requiring the longest time to do so. μ_A/A serves as our metric, measuring the average execution time required per unit simulation time.

We focus on "generations" of messages, a notion which arises as follows. Imagine that LPs synchronize at t, and then each executes all known events in the window without paying any attention to any possible communications. The set of messages sent during the first sweep with time-stamps in $[t, t + A)$ are in generation 1. Imagine now that an LP gathers up all the generation 1 messages sent to it, and processes them. These must cause rollback, anti-messages, and new routing messages. The set of all anti-messages and new routing messages are in generation 2. In general, a message is in generation $i + 1$ if it is the direct result of a rollback caused by a generation i message. We denote the random number of generation i messages received by an LP as G_i, and denote by R_i the random number of events processed as a result of receiving generation i messages.

We assume that A is small enough and the message density is high enough to ignore the possibility of a job received in $[t, t + A)$ going into service in that same window. Under this assumption, the number and times of all service-entry events during $[t, t + A)$ are known after the LPs synchronize at t, and remain unaltered (except for destination and content) while processing $[t, t + A)$. The number of such events at a given LP is a random variable S that is Poisson distributed with mean $A\mu_s$. All other events are job arrivals, of which there are J, which is also Poisson with mean $A\mu_s$.

Event reprocessing costs depend on how quickly the parallel simulator receives and reacts to straggler messages. For example, the analysis of Gupta *et al.* (1991) assumes zero message transmission delay, and that rollback occurs immediately following the complete processing of whatever event is being served at the instant the straggler message arrives. If two or more stragglers arrive during that processing time, the reprocessing effect is as though only the straggler with least time stamp was received, others exact no additional cost. But now consider the effect a communication delay may have on the algorithm. If A is small enough an LP will have few events in a window; in the time it takes a message to travel between processors, the recipient LP will already be ready to synchronize at time $t + A$. Even if communication is faster it is frequently the case (and we have observed empirically on actual applications) that the cost of probing for new messages after each event is prohibitively high on distributed memory architectures, because such a probe involves a system call. To model this effect, we assume that if a straggler message is received at some time $s \in [t, t + A)$ then the effect of that straggler is to re-execute all events at that LP from s to $t + A$, and to send anti-messages after all messages generated previously by those events. If an LP receives k generation i stragglers, we assume that each is processed serially, incurring k separate recomputation costs. This assumption is reasonable so long as an LP has only a few events in a window.

If we define generation 0 messages as corresponding to the service entry events and job arrival events, we write $R_0 = S + J$, and express the total number of events processed in the window by

$$W = \sum_{i=0}^{\infty} R_i.$$

The distribution of each reprocessing cost R_i depends on the number and time-stamps of generation i messages. The actual distributions for these messages are untractably complicated, so we will approximate. For generations $i > 1$, we assume that G_i has a Poisson distribution. Such an assumption is standard, since given a total number N generation $i-1$ messages, the number arriving at an LP is binomial $B(N, 1/P)$, P being the number of LPs. Binomials with large N and small probabilities of success are frequently modeled as Poisson.

We also approximate the distributional form of the random arrival time of a generation i message. Each such message corresponds to a service-entry event in some LP; the arrival time is the service-entry time plus an exponential. Each service entry event has some *rank* reflecting whether it is the first, second, or so on service entry event in $[t, t+A)$, on its LP. The arrival time distribution of the message sent by the i^{th} service entry event following time t is t plus the convolution of $i+1$ i.i.d. exponentials, i.e., an Erlang-$(i+1)$; we say that the arrival message has rank $i+1$. We will have occasion to condition on the service-entry event lying in $[t, t+A)$, in which case the message's arrival time distribution is suitably modified. In order for such a message m to be sent (in generations > 0), the i^{th} service entry event must be reprocessed, implying the arrival of an earlier straggler—information that alters m's arrival time distribution. Our model does not attempt to capture this distributional dependency. Under our simplifying assumption then, every generation i arrival message in $[t, t+A)$ has a time-stamp whose distribution is t, plus some Erlang conditioned on being less than A. Let $a_{i,k}$ denote the mean fraction of generation i messages that have rank k. Letting $\tilde{f}_k(s)$ be the density function for an Erlang-k conditioned on being less than A, we approximate the arrival time density function of an arbitrary generation i message as the mixture $t + \sum_{k=2}^{\infty} a_{i,k} \tilde{f}_k(s)$.

Table 1 summarizes our notation. All random quantities are LP-oriented, rather than system-oriented.

It remains to determine weighting factors $\{a_{i,k}\}$ and the distributions for W, G_i, and R_i. The approach is to condition on $S + J = k$, and determine distributions for G_i, R_i, and W suitably conditioned, call them $G_i(k)$, and $R_i(k)$, and $W(k)$. Assuming that all arrival messages are independent of each other, we compute $W(k) = \sum_{i=0}^{\infty} R_i(k)$, because the individual random variables in the convolution will be independent. It is straightforward then to uncondition on $S + J$ (since S and J are independent and Poisson). The values for $E[G_i]$ and $\{a_{i,k}\}$ can be built up with increasing i, as will be shown.

Let us first consider $E[G_1]$. A generation 1 message arises whenever a service-entry event in $[t, t+A)$ sends an arrival message with time-stamp less than $t+A$ (all other arrival events were sent by service entry events in previous windows). If we condition on $S = k$ service-

W	Total events processed in a window
S	Service entry events in a window
J	Job arrival events
μ_s	Service rate for queue server
G_i	Generation i arrival messages received
R_i	Events reprocessed by all generation i arrivals
r_i	Events reprocessed by single generation i arrival
$a_{i,j}$	Fraction of generation i arrivals with rank j
$\tilde{f}_j(s)$	Density function of Erlang-j conditioned on $s \leq A$
$F_j(s)$	Cummulative distribution function for Erlang-j
$B(n,p)$	A binomial random variable with parameters n and p
$H_j(s)$	Cummulative distribution function of Erlang-j conditioned on sum of first $(j-1)$ stages being less than A

Table 1: Summary of Notation

entry events in $[t, t+A)$, the joint distribution of their times in $[t, t+A)$ is identical to that of k independent $[t, t+A]$ uniform random variables (Ross 1983, pg. 37). Choosing one of these k uniformly at random, the probability that its arrival message lies outside of $[t, t+A)$ is given by

$$\begin{aligned}\Pr\{\text{Arrival message time for service entry event} > t+A \mid S=k\} &= \int_t^{t+A} \frac{1}{A} e^{-\mu_s v}\, dv \\ &= \frac{1.0 - e^{-\mu_s A}}{A}.\end{aligned}$$

This leads to the observation that the mean number of arrival messages generated in $[t, t+A)$ that fall outside of $[t, t+A)$ is $1.0 - e^{-\mu_s A}$. Since the mean total number of arrival messages generated in $[t, t+A)$ is $\mu_s A$, we obtain

$$E[G_1] = \mu_s A - (1.0 - e^{-\mu_s A}).$$

Values for the $\{a_{1,k}\}$ are also easily derived. For an arrival message to be of rank j it is necessary that the Erlang associated with its arrival time $t+a$ be less than $t+A$. From Bayes Theorem we obtain

$$\begin{aligned} a_{1,j} &= \Pr\{\text{A generation 1 arrival message in } [t, t+A) \text{ has rank } j \geq 2\} \\ &= \Pr\{\ a \sim \text{Erlang-}j \mid a < A\ \} \\ &= \frac{F_j(A)}{\sum_{k=2}^{\infty} F_k(A)} \quad \text{for } j \geq 2, \end{aligned}$$

where F_k is the cumulative distribution of an Erlang-k with rate parameter μ_s.

We now turn to the analysis for higher generations. Suppose $E[G_i]$ and the values $\{a_{i,j}\}$ are known for generation i; condition on $S+J=k$ and consider the distribution of $R_i(k)$. Under our assumptions, an arrival at time $t+v \in [t, t+A)$ will cause the reprocessing of every known arrival event and service-entry event with time-stamp between $t+v$ and $t+A$. Given $S+J=k$ we view the placement in time of events on $[t, t+A)$ as that of k uniforms on $[t, t+A]$ (Ross 1983, pg.37). As a consequence, the number of events reprocessed by a rollback-inducing arrival at time $t+v$ has the distribution of a Binomial $B(k, (A-v)/A)$, representing the sum of k Bernoullis with success probability $(A-v)/A$. Coupling this fact with the assumed distributional form of generation i messages we compute

$$\Pr\{n \text{ events reprocessed by generation } i \text{ message} \mid J=k\} = \int_{v=0}^{A} \sum_{j=2}^{\infty} a_{i,j} \tilde{f}_j(v) \Pr\{B(k,(A-v)/A)=n\} dv. \quad (1)$$

Equation (1) approximates the distribution of random variable $r_i(k)$, the random number of events reprocessed by a single generation i message, conditioned on $S+J=k$. We ignore here the fact that the arrival message is itself an arrival event, and that the set of known arrival events is continuously in flux through successive generations. Accepting this we compute the distribution of $R_i(k)$ as the random convolution of M independent instances of $r_i(k)$, M being Poisson with rate $E[G_i]$.

Values $\{a_{i,j}\}$ are computed in a similar fashion. If we condition on a generation i arrival at time $t+v$ and condition on there being m service-entry events in $[t, t+A)$, then the number of these falling between $t+v$ and $t+A$ is binomial. The probability that a generation $i+1$ message of rank j is generated by this arrival is zero if there aren't enough service-entry events, i.e., if $j-1>m$. Otherwise it is the probability that the $(j-1)^{st}$ service-entry event occurs after v, and that the message it generates falls within $[t, t+A)$. This gives

$$\begin{aligned} b_{i+1,j}(m) &= \Pr\{\text{generation } i \text{ message creates a rank } j \text{ generation } i+1 \text{ message} \mid S=m\} \\ &= H_j(A) \times \int_{v=0}^{A} \sum_{j=2}^{\infty} a_{i,j} \tilde{f}_j(v) \Pr\{B(m, v/A) > j-1\} dv \end{aligned} \quad (2)$$

where we recall that H_j is the cumulative distribution function of an Erlang-j conditioned on the sum of its first $j-1$ exponential stages being less than A. $H_j(A)$ gives us the probability that a reprocessed rank-$(j-1)$ service-entry event produces a message in the next generation.

Figure 2 helps to explain these ideas. A situation with $S=5$ is shown, where an arrival message at time $t+v$ falls ahead of the first three service entry events. The service entry events ahead of the arrival have ranks 4 and 5 respectively. Arcs illustrate the send/receive time difference between the messages sent by the reprocessed events; the rank 4 event message

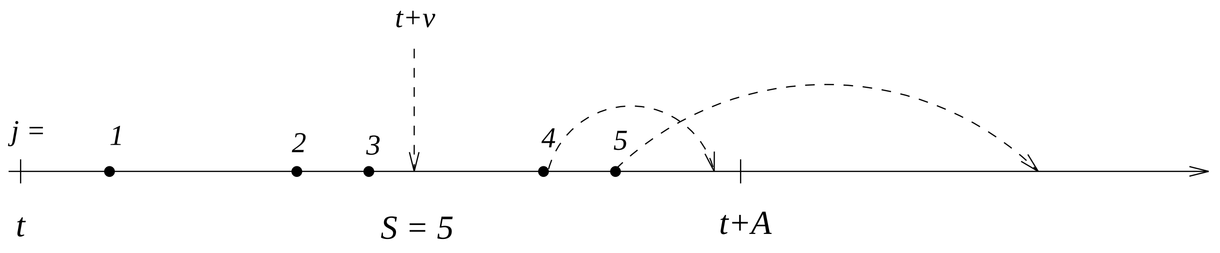

Figure 2: Reprocessing of a rank 4 service-entry event generates a rank 5 message for the next generation.

falls within the window, the rank 5 event message does not. In order for there to be a rank 5 message generated, the 4th ranked service event must lie to the right of v, as must the receive time of its message. The distribution of that receive time is an exponential added to the distribution of 4th service event, the latter of which is a conditional Erlang-4.

For each rank j let $b_{i+1,j}$ be the result of unconditioning equation (2) on S. Then, recalling that each reprocessed service-entry event generates two messages (assumed to have the same time-stamp), the mean number of generation $i+1$ messages with rank j is $2 \times E[G_i] \times b_{i+1,j}$, and the coefficients $\{a_{i+1,j}\}$ are given by

$$a_{i+1,j} = E[G_i] \left(\frac{b_{i+1,j}}{\sum_{k=2}^{\infty} b_{i+1,k}} \right).$$

Finally, the mean number of arrival messages in the next generation is simply

$$E[G_{i+1}] = 2 \sum_{k=2}^{\infty} b_{i+1,k}.$$

Using these recursions one may, for every $S+J=k$, compute the distribution of $R_i(k)$, for all generations $i = 1, 2, \ldots$. Conditioned on $S+J=k$, the random variables $R_0(k), R_1(k), \ldots$ may be taken to be independent (because the processes driving them are highly randomized arrivals from elsewhere), whence we may compute the distribution of the convolution $W(k) = \sum_{i=0}^{\infty} R_i(k)$. Finally, knowing this distribution for each $S+J=k$, we compute the distribution of W by unconditioning on $S+J$ (known to be Poisson).

Of course, any computer program calculating these distributions must truncate the infinite sums. Taking $\mu_s = 1$, we have found that summing over the first twelve generations yields accurate numbers when $A \in (0, 2\mu_s)$.

The distribution of W describes the workload of a single LP, in terms of the numbers of events processed. With large numbers of LPs and the randomizing message routing, we may treat the LP workloads as being independent random variables. Under this assumption it is straightforward to express the expected maximum workload among N LPs. Letting $M_N(A)$ be the maximum workload, we know that for every non-negative integer w

$$\Pr\{M_N(A) \leq w\} = \Pr\{W \leq w\}^N$$

so that

$$\begin{aligned} E[M_N(A)] &= \sum_{w=0}^{\infty} \Pr\{M_N(A) > w\} \\ &= \sum_{w=0}^{\infty} (1.0 - \Pr\{W \leq w\}^N). \end{aligned}$$

Numerical problems may arise computing y^x when y is small and x is large; a good approximation for $E[M_N(A)]$ is the so-called *characteristic maximum*, used for instance in Eick *et al.* (1993). Given N, the characteristic maximum of W is the smallest value w_c such that $\Pr\{W > w_c\} < 1/N$. Since W is discrete, we further refine the estimate with linear interpolation of $W's$ cumulative distribution function between w_c and $w_c - 1$, in essence creating a continuous version $\tilde{W}$ and solving for $\tilde{w}_c$ such that $\Pr\{\tilde{W} > \tilde{w}_c\} = 1/N$. $\tilde{w}_c$ estimates $E[M_N(A)]$.

The utility of our model is illustrated by Figure 3, where we compare model predictions of $E[M_{64}(A)]$ and $E[M_{1024}(A)]$ with measurements (from simulation of our model) for varying values of A. Each measurement point is estimated from one hundred window replications. As our purpose is only to ensure that the model captures general trends we omit confidence intervals. We see that the model predicts performance tolerably well over a range where the predictions span a factor of ten between smallest, and largest, although there is a breakdown at the larger end.

It is also instructive to consider how the fraction of committed events (those events that are not later reprocessed) behaves as a function of A. This is illustrated in Figure 4, where we plot the ratio of the expected maximum committed workload on a processor to the expected maximum total workload, for 64 and 1024 LPs. For both curves shown, the fraction of useful work decreases linearly in A after a certain point. This suggests that under the assumptions of our model, it does not make sense to increase A indefinitely. This is explained in the section to follow.

4 Comparison with YAWNS

It is instructive to consider how $E[M_N(A)]$ behaves as a function of A. $E[M_N(A)]$ is basically the product of three terms, (i) the number of message generations required until all LPs have finished the window, (ii) the average number of rollbacks per generation, (iii) the average

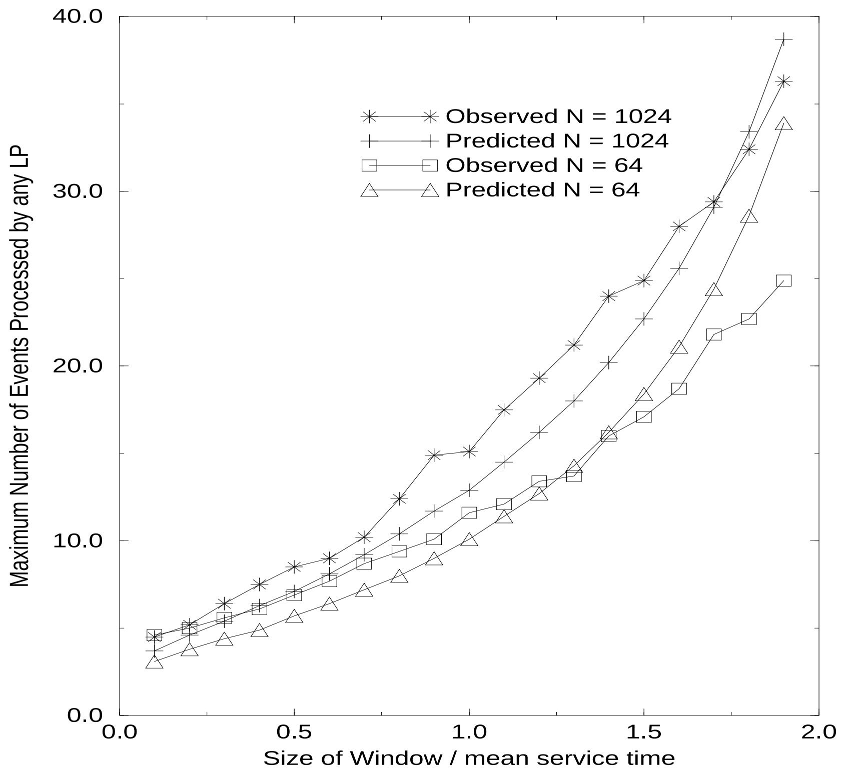

Figure 3: Comparison of observed and predicted mean maximum events processed in a window by any LP.

number of messages reprocessed per rollback. Our simulations have suggested that the number of generations grows linearly in A, an observation that agrees with the analysis of Eick *et al.* (1993). The number of messages reprocessed each rollback also increases linearly in A, for the simple reason that increasing the window size introduces new events at the top of the window to be rolled back along with the ones which were rolled back with smaller windows. The average number of rollbacks per generation is also linear in A, because each arrival message is assumed to cause the re-evaluation of all later messages. $E[M_N(A)]$ is at least a cubic function of A, so that the cost per simulation time unit $E[M_N(A)]/A$ (whose units are execution time per simulation time unit) is at least quadratic in A. This suggests that there may be some A^* minimizing this cost. Figure 5 confirms this intuition. In fact, it is interesting to note that A^* appears to be slightly less than $\mu_s = 1$. This too is in agreement with the model of Eick *et al.*, even though the models and costs are different. We conclude that $A = \mu_s$ is an excellent choice, and in the remainder presume this equality.

It turns out that the behavior of $E[M_N(\mu_s)]/\mu_s$ in N is an almost perfectly linear function of $\log N$ in the range considered, with $E[M_N(\mu_s)]/\mu_s \approx \log N + 2.9$. To incorporate the effects of state-saving, we'll assume that the per-event cost of state-saving is a factor of α, so that the cost of executing n events with attendant state-saving is αn. Note that this model does

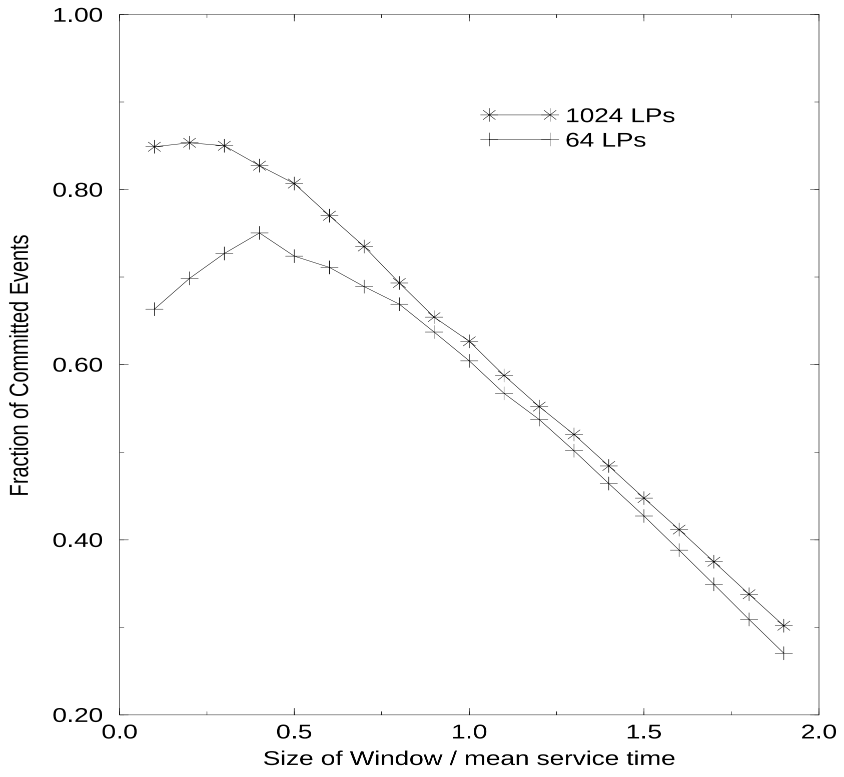

Figure 4: Fraction of committed events as a function of A, for 64 and 1024 LPs.

not presume that state is saved each event; it only presumes that the aggregate state-saving overhead amortized over events is α.

$E[M_N(A)]$ does not incorporate the cost of synchronization. To include these costs we must consider how synchronization is performed in a computation of this type. A software solution described by Nicol (1993a) has every LP engaging in synchronization activity once it finds itself apparently at the synchronization point. We could assume some synchronization cost for each and every straggler message, however this seems excessive. Instead we'll assume that the number of synchronizations are those one would incur by synchronizing at the end of each generation; empirical evidence (Nicol, 1993a) suggests that each such synchronization costs roughly twice that of a conventional synchronization. Our simulation studies show that a window of width $A = \mu_s$ requires 2.5 generations on average, a figure that is relatively insensitive to the number of LPs. Taking B as the execution cost of a conventional barrier synchronization the overall execution cost per unit simulation time given N LPs is

$$C_{optim}(N) \approx \alpha(\log_2 N + 2.9) + 5B. \tag{3}$$

Note that our assumed synchronization cost structure does not affect the optimality of $A^* = \mu_s$, since synchronization costs then grow linearly in A. Also note that B shows

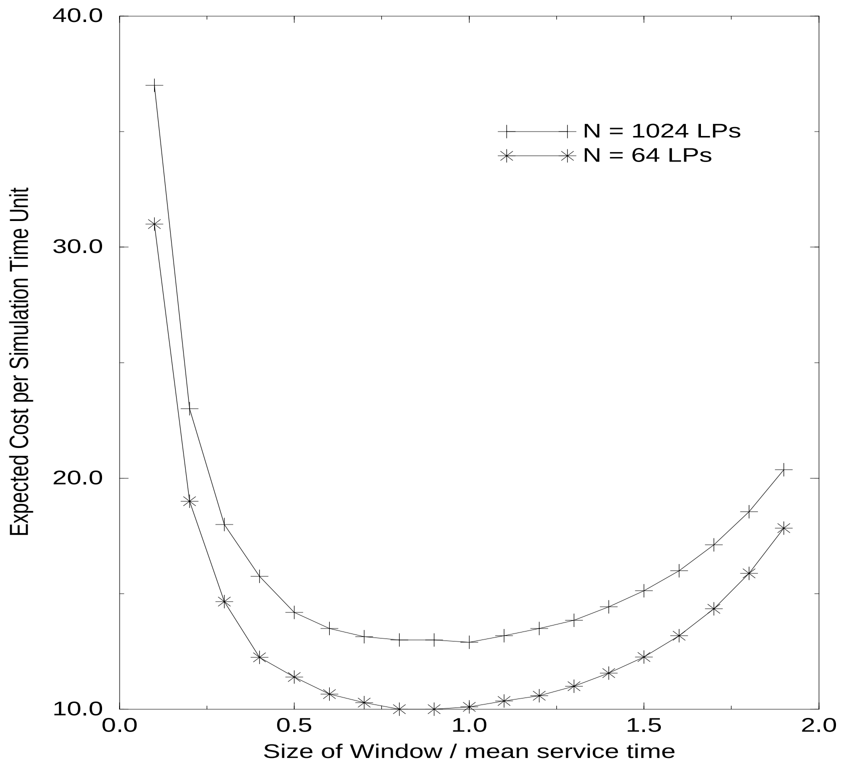

Figure 5: $E[M_N(A)]/A$ as a function of A, for 64 and 1024 LPs.

no dependence on N. Asymptotically it must grow with $\log_2 N$, however we presume that the cost of executing an event is large enough to overshadow this dependence before N becomes extremely large.

Now consider YAWNS. Nicol (1993) established that the average width of the conservative window is at least $\mu_s\sqrt{\pi/(2N)} \approx 1.25\mu_s/\sqrt{N}$. In windows this small, the average maximum number of events processed by any LP is no larger than 2, for large N it is much closer to 1. Including the barrier synchronization, YAWN's cost per unit simulation time is no greater than

$$C_{yawns}(N) = \frac{(2+B)\sqrt{N}}{1.25}. \tag{4}$$

One consequence of $A^* \approx \mu_s$ is that for large N there is relatively little advantage to avoid state-saving within the YAWNS conservative window, because the optimistic window is so much larger. For instance, if $N = 100$, then only about 12% of a window avoids state-saving. It costs very little to compute the conservative window, and so if convenient ought to be done. However, the performance benefits from doing so are not large.

We may use equations (3) and (4) to compare the approaches, given values for overhead costs. At a higher level we observe that YOW has an $O(\log_2 N)$ cost while YAWNS has an

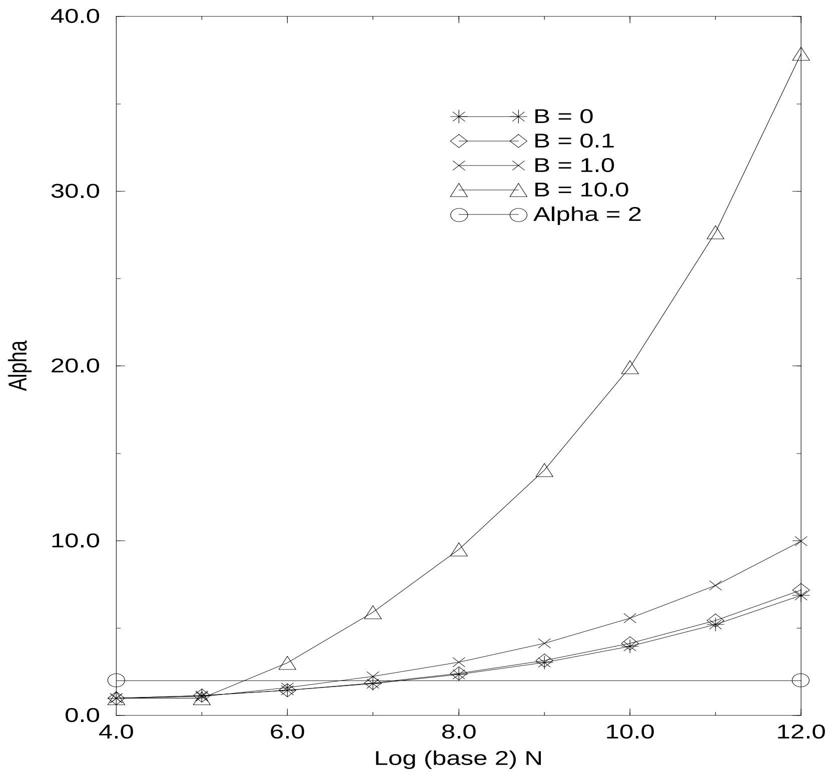

Figure 6: Function specifying LP threshold N^* after which YOW is better than YAWNS.

$O(\sqrt{N})$ cost. For sufficiently large N, the optimistic approach will always achieve a lower cost. How large must that N be? We depict this graphically in Figure 6, plotting the solution (to α) of equation $C_{optim} - C_{yawns} = 0$, as a function of $\log_2 N$ and for various values of B. Solutions $\alpha = \alpha(N, B) < 1$ are plotted as 1, since state-saving can never accelerate the cost of executing an event. For any given value of α^* and known value B, one can determine the N^* for which $\alpha^* = \alpha(N^*, B)$, and determine that YOW is better than YAWNS for all $N \geq N^*$. Imagine that state-saving doubles the cost of executing an event. Plotting the line $\alpha = 2$ we look for its intersection with the various synchronization cost curves; N's associated with the intersection define N^*. For instance, if $B = 0$ then YOW is better for $N > 128$. If $B = 1.0$ however, then YOW needs only $N > 100$, and if $B = 10.0$ it needs only $N > 40$. YAWNS is clearly impacted more strongly by increasing synchronization costs, as it synchronizes on the order of $\sqrt{N}$ times more often than YOW.

The assumptions under which we've analyzed YAWNS show that if simulation time advances by exponentially distributed amounts and if only one LP is assigned to each processor, then YAWNS has a relatively high cost. However, YAWNS performance is sensitive to both of these assumptions. If an LP's service time is bounded below by $\gamma > 0$, then the size of a YAWNS window at least γ. This seemingly minor change of assumptions defeats the assured asymptotic superiority of YOW, because it changes YAWNS $O(\sqrt{N})$ cost to $O(1/\gamma)$. The relative performance of YAWNS and YOW depend primarily then on α, B, and γ.

Next we show that by considering the effects of aggregating LPs onto processors, YAWNS again circumvents YOW's assured superiority, even if service times are exponentially distributed. The reasoning is straightforward. Let N denote the number of LPs, P denote the number of processors, and presume that each processor simulates N/P LPs. The average size of a YAWNS window is $y(N) = 1.25\mu_s/\sqrt{N}$; the number of events each LP executes in a window is Poisson with rate $2y(N)$. Since LPs are independent, the number of events a *processor* executes each window is Poisson with rate $\lambda(N) = 2.5\sqrt{N}/P$. If $M_P(\lambda)$ is the mean expected maximum of P Poissons with rate λ, then YAWNS' cost per unit simulation time per co-resident LP is

$$D_{yawns}(N) = \frac{\sqrt{N}}{1.25} \times \frac{M_P(\lambda(N)) + B}{N/P}.$$

Eick *et al.* study the asymptotics of $M_P(r)$, showing that $M_P(r) \sim \log P/\log\log P$ for small r, and $M_P(r) \sim 2r$ for $r = \Omega(\log P)$. $\lambda(N)$ increases unboundedly in N, implying that for sufficiently large N

$$\begin{aligned} D_{yawns}(N) &\sim \frac{2\lambda(N) + B}{N/P} \\ &= 4 + \frac{B}{\lambda(N)}. \end{aligned}$$

The second term vanishes as N grows, showing that YAWNS' normalized execution cost per LP is asymptotically constant.

The result above does not imply that YAWNS' normalized cost is asymptotically 4 because constants in the asymptotic analysis are missing from our expressions. However, Figure 7 plots the predicted cost (not asymptotic) as a function of $\log(N/P)$, assuming $P = 16$ and $B = 0$. It also plots the predicted performance of YOW, again assuming $A = \mu_s$, under the same values of N and P. State-saving overhead factors of $\alpha = 1, 1.2$ and 1.5 are shown. These figures are obtained by computing appropriate convolutions of W, and finding the expected maximum convolved processor load. Since aggregation may change the relative optimality for YOW of $A = \mu_s$, we also computed costs assuming other window sizes. Differences from the presented data were small. Assuming that synchronization costs contribute little to the overhead cost under high loads, it is clear now that YAWNs can do better than YOW under high degrees of aggregation, or when state-saving overhead is significant.

It should also be noted that our model assumptions work against YOW in the aggregated case. When LPs tend to communicate with other LPs on the same processor one may expect advantages due to significantly reduced communication costs. This is especially true in our

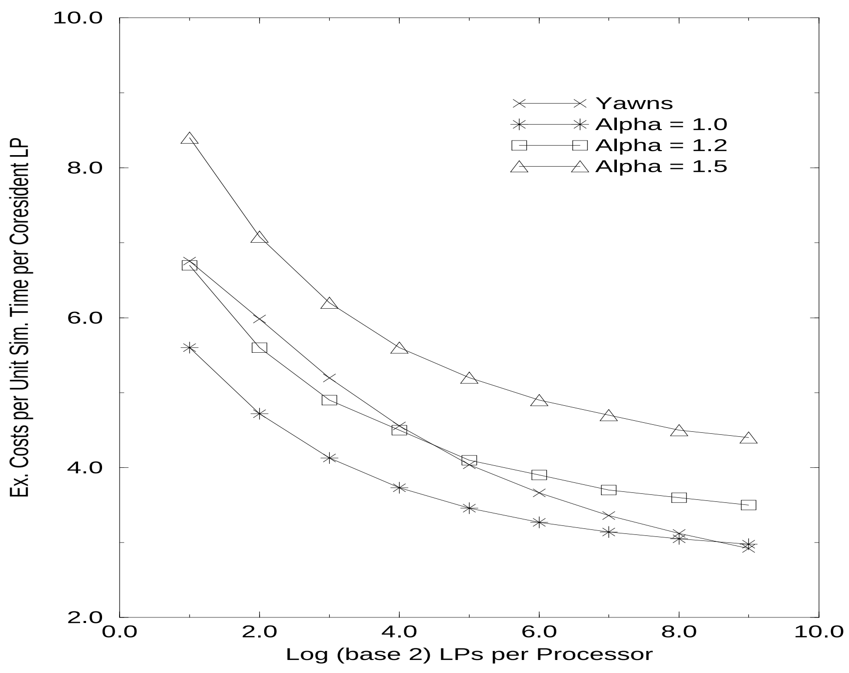

Figure 7: YAWNS and YOW normalized cost per unit simulation time under aggregation as a function of $\log(N/P)$.

model because the recomputation cost due to delayed stragglers is consequential. However, the assumption that messages are routed uniformly at random means that no such locality is present in the model. Our costing assumptions remain valid in the aggregated case so long as event processing costs are of the same order as communication and the window size is small.

5 Conclusions

We have analyzed a simple model of parallel simulation, to assess the benefit of adding optimism to an existing conservative synchronization protocol, YAWNS. Our approach is novel to the the problem area, and is relatively simple. We show how to compute approximate probability distributions of processor workload. To these distributions we add overheads due to state-saving, and synchronization. In addition, we consider the effects on performance due to aggregating many LPs onto a processor.

The extension, YOW, remains window-based; our analysis predicts that there is some optimally-size window, a prediction borne out by experiments. The window is relatively large compared to YAWNS', but is still so small that on average a logical processor executes only two events within it. Using this window size we construct equations predicting YOW's and

YAWNS' execution cost per unit simulation time, and observe that under the assumption of one LP per processor, YOW is asymptotically better than YAWNS, as the number of LPs grows. However, when we analyze performance allowing many LPs per processor we find that YAWNS does better than YOW under moderate levels of aggregation, or when state-saving costs are non-negligible.

Far-reaching quantitive conclusions are questionable for a model of this type. For both YAWNS and YOW small changes in model assumptions will significantly affect quantitative results. Qualitatively though we may infer that if actual reprocessing costs resemble those in our model and global synchronization costs aren't high, then it is likely that limiting optimism is a good thing in a window-based framework. We also conclude that if probability distributions driving simulation time advance have no lower support, then YAWNS will not do well when the problem is sparse relative to the architecture. However, this problem disappears for large problems where LPs are highly aggregated onto processors. Perhaps the strongest conclusion we offer is that performance of parallel simulations is more strongly a function of state-saving, synchronization/communication costs, problem size, and degree of aggregation than it is for the specific synchronization protocols. Synchronization methods ought to be chosen after the problem is known, and to take advantage of the problem's characteristics.

An open and important question remains, whether a window-based framework offers better performance than a completely asynchronous one. While we have not addressed this problem, we believe that extension of our analytic approach to the Gupta *et al.* model assumptions may lead to the desired comparison. We also believe a more precise treatment of the effects of communication delay is possible, which will lead to better understanding of the effect the underlying architecture has synchronization behavior.

REFERENCES

Akyldiz, I.F., L. Chen, S.R. Das, R.M. Fujimoto and R.F. Serfozo 1992. Performance Analysis of Time Warp With Limited Memory. *1992 ACM Sigmetrics Conference*

Ayani, Rassul 1989. A Parallel Simulation Scheme Based on Distances Between Objects. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, Volume 21 Number 2, 113-118. Society for Computer Simulation,

R.E. Bryant. Simulation if packet communication architecture computer systems. *MIT-LCS-TR-188*, Massachusetts Institute of Technology, 1977.

Ayani, R. and H. Rajaei 1992. Parallel Simulation Using Conservative Time Windows. *Proceedings of the 1992 Winter Simulation Conference*, pgs. 709-717, Dec. 1992.

Chandy, K.M. and J. Misra 1979. A Case Study in the Design and Verification of Distributed Programs. *IEEE Transactions on Software Engineering*, SE-5,5 May 1979, 440-452.

Chandy, K., and R. Sherman 1989. The Conditional Event Approach to Distributed Simulation. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation*, pgs. 93-99, January, 1989.

Dickens, P. and P. Reynolds 1990. SRADS with Local Rollback. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, 161-164, January, 1990,

Eick, S., Greenberg, A., Lubachevsky, B. and Alan Weiss, 1993. Synchronous Relaxation for Parallel Simulations with Applications to Circuit-Switched Networks. *ACM Transactions on Modeling and Computer Simulation*, Volume 3 Number 4, pgs. 287-314, Oct. 1993.

Felderman, R. and L. Kleinrock 1991. Bounds and Approximations for Self-Initiating Distributed Simulation Without Lookahead. *ACM Transactions on Modeling and Computer Simulation*, Vol 1, No 4, Oct 1991, pp 386-406.

Fujimoto, R. 1990. Parallel Discrete Event Simulation. *Communications of the ACM*, Volume 33, Number 10, October 1990, 30-53.

Gupta, A., I. Akyldiz and R. Fujimoto 1991. Performance Analysis of Time Warp With Multiple Homogeneous Processors. *IEEE Transactions on Software Engineering.* Volume 17, No. 10 pgs. 1013-1027, Oct. 1991.

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

Lubachevsky B. 1988. Bounded Lag Distributed Discrete Event Simulation. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, pgs. 183-191, January, 1988.

Lubachevsky B., A. Shwartz and A. Weiss 1989. Rollback Sometimes Works... If Filtered. *Proceedings of the 1989 Winter Simulation Conference.* 630-639, December, 1989.

Lubachevsky, B. 1989a. Scalability of the Bounded Lag Distributed Event Simulation. *Proceedings of the 1989 SCS Multiconference on Distributed Simulation* 100-105, January, 1989.

Madisetti, V., D. Hardaker and R. Fujimoto 1992. The MINDIX Operating System for Parallel Simulation. *Distributed Simulation*, SCS Simulation Series, Vol. 24, Num. 3, pgs. 65-74, Jan. 1992.

Misra, J. 1986. Distributed Discrete-Event Simulation. *Computing Surveys*, Vol. 18, pgs. 39 - 64, March 1986.

Nicol, D. 1991. Performance Bounds on Parallel Self-Initiating Discrete Event Simulations. *ACM Transactions on Modeling and Computer Simulation*, Volume 1, No.1, 1991.

Nicol, D.M. and R. Fujimoto 1994. Parallel Simulation Today. *ICASE Technical Report # 92-62*. To Appear in *Annals of Operations Research*, Nov. 1994.

Nicol, D. 1993. The Cost of Conservative Synchronization in Parallel Discrete Event Simulation. *Journal of the ACM*, Vol. 40, Num. 7, pgs. 304-333, April, 1993.

Nicol, D.M., 1993a. Global Synchronization for Optimistic Parallel Discrete Event Simulation. *Proceedings of the 7th Workshop on Parallel and Distributed Simulation*, pgs. 27-34., May, 1993.

Peacock, J.K, J.W. Wong and E.G. Manning 1979. Distributed Simulation Using a Network of Processors. *Computer Networks* (1979), 44-56, North-Holland Publishing.

Reiher, P., R. Fujimoto, S. Bellenot, and D. Jefferson 1990. Cancellation Strategies in Optimistic Execution Systems. *Proceedings of the 1990 SCS Multiconference on Distributed Simulation*, January, 1990.

Reynolds, P. 1988. A Spectrum of Options for Parallel Simulation. *Proceedings of the 1988 Winter Simulation Conference,*, pgs. 325-332, Jan. 1988.

Reynolds, P., Pancerella, C. and S. Srinivasan 1993. Design and Performance Analysis of Hardware Support for Parallel Simulations. *Journal of Parallel and Distributed Computing*, Volume 18, No. 4, August 1993, pgs. 435-453.

Righter, R and J. Walrand 1989. Distributed Simulation of Discrete Event Systems. *Proceedings of the IEEE, Vol. 77, No. 1 Jan. 1989.*

Ross, S. 1983. Stochastic Processes. *Wiley Series in Probability and Mathematical Statistics.* Published by John Wiley and Sons, Inc., 1983.

Sokol, L., D. Briscoe and A. Wieland 1988. MTW: A Strategy for Scheduling Discrete Simulation Events for Concurrent Execution. *Proceedings of the 1988 SCS Multiconference on Distributed Simulation*, pgs. 169-173, Jan. 1988.

Steinman, J. 1991. SPEEDES: Synchronous Parallel Environment for Emulation and Discrete Event Simulation. *Proceedings of the SCS Western Multiconference on Advances in Parallel and Distributed Simulation*, Volume 23, No. 1, pgs. 95-103.

Steinman, J. 1992. SPEEDES: A Unified Approach to Parallel Simulation. *Proceedings of the 6th Workshop on Parallel and Distributed Simulation*, SCS Simulation Series, Vol. 24, No. 3, pgs. 75-84, Jan. 1992.

Turner, S. and M. Xu 1992. Performance Evaluation of the Bounded Time Warp Algorithm. *Distributed Simulation*, SCS Simulation Series, Vol. 24, Num. 3, pgs. 117-126, Jan. 1992.