Analysis of Bounded Time Warp and Comparison with YAWNS

Phillip M. Dickens
Illinois Institute of Technology
and
David M. Nicol
College of William and Mary
and
Paul F. Reynolds, Jr.
University of Virginia
and
J. M. Duva
University of Virginia

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), Mail Stop 132C, 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.
Name: Phillip M. Dickens
Address: Department of Computer Science and Applied Mathematics, Illinois Institute of Technology, 235C Stuart Building, 10 West 31st. St. Chicago, Ill. 60616
Affiliation: Illinois Institute of Technology
Name: David M. Nicol
Address: Department of Computer Science, 6211 Sudikoff Laboratory, Dartmouth College, Hanover, NH 03745-9725
Affiliation: Dartmouth College
Name: Paul F. Reynolds, Jr.
Address: Department of Computer Science, Olsson Hall, 214, University of Virginia, Charlottesville, Va. 22903
Affiliation: University of Virginia
Name: J. M. Duva
Address: Department of Applied Mathematics, Olsson Hall, University of Virginia, Charlottesville, Va. 22903
Affiliation: University of Viirginia

This paper studies an analytic model of parallel discrete-event simulation, comparing the YAWNS conservative synchronization protocol with Bounded Time Warp. The assumed simulation problem is a heavily loaded queueing network where the probability of an idle server is close to zero. We model workload and job routing in standard ways, then develop and validate methods for computing approximated performance measures as a function of the degree of optimism allowed, overhead costs of state-saving, rollback, and barrier synchronization, and workload aggregation. We find that Bounded Time Warp is superior when the number of servers per physical processor is low (i.e. sparse load), but that aggregating workload improves YAWNS relative performance.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*modeling techniques, performance attributes*; I.6.0 [**Simulation and Modeling**]: General

General Terms: Algorithms, Performance

Additional Key Words and Phrases: Parallel Simulation, synchronization Protocol

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], and [Righter and Walrand 1989]. Nicol and Fujimoto [Nicol and Fujimoto 1994] give a more current state-of-the-art review.

Most PDES synchronization protocols fall into two basic categories (although a more detailed taxonomy is given in [Reynolds 1988]). *Conservative* protocols (e.g. [Chandy and Misra 1979], [Bryant 1977], [Peacock et al. 1979], [Lubachevsky 1988], [Chandy and Sherman 1989], and [Nicol 1993a]) 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.

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 Rajae 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 1993a]). The advantage to a conservative protocol is that synchronization information moves quickly through the system, lowering overhead costs. This efficiency usually comes at the 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. 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 its performance relative to optimistic techniques. For this comparison the Bounded Time Warp (BTW) protocol suits our purposes well because it, like YAWNS, is an iterative algorithm requiring a global synchronization at the end of each iteration. It is worthwhile to note the differences between these two approaches. Both protocols consist of three phases where in the first phase the LPs cooperatively define a global window of simulation time, in the second phase the LPs concurrently process their events with time-stamps falling within this window, and in the third phase the LPs engage in a barrier synchronization. The primary difference is the size of the global window of simulation time. YAWNS creates windows small enough to guarantee that all processing within the window will be correct. BTW defines larger windows, and allows the LPs to optimistically process all events within this larger window.

By comparing YAWNS and BTW side-by-side we will better understand the costs and benefits of employing optimism in a window-based framework. By constructing very small windows YAWNS avoids the overheads of optimism, at the cost of reduced parallelism and larger frequency of barrier synchronization. Constructing larger windows, BTW enjoys a larger degree of potential parallelism and a lower frequency of barrier synchronization, but pays for that with state-saving and rollback overheads.

YAWNS was analyzed elsewhere ([Nicol 1993a]). The present analysis of BTW develops an approximated form of the probability distribution of the number of events an LP executes within a window, and quantifies the distribution using numerical techniques. This style of approach is standard in many areas of modeling, but is relatively novel among analyses of parallel simulation. It allows one to model more complex phenomena than would otherwise be possible. One crucial point is that the approximation is validated via simulation of a BTW computation. Another crucial point is that our model incorporates a delay between the instant when an anti-message is sent, and the instant it is received. This point has a profound impact on the conclusions of the analysis.

Our model predicts that BTW's optimal window size is much larger than YAWNS's, but surprisingly that only one or two events are processed (on average) by an LP within this optimally sized window. To facilitate comparison we derive formulas for

YAWNS' and BTW'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) BTW prevails. However, if we fix the size of the architecture and aggregate LPs onto processors, then YAWNS can prevail.

The contribution of this paper is to present a validated approach to analytically approximating the behavior of a complex synchronization protocol. Unlike most previous analyses of parallel simulation protocols, the approach is general enough to incorporate numerous overhead costs, and to include the effects of workload aggregation. Using this approach we compare YAWNS and BTW, and identify (as a function of overhead costs) situations where it is more advantageous to use optimism and situations where it is more advantageous to remain conservative. A reader uninterested in the mathematical details may well still be interested in the qualitative conclusions those details produce.

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 LP's workload, including reprocessed messages due to roll backs. Section 4 applies those approximations to compare YAWNS and BTW, and Section 6 presents our conclusions.

2. MODEL

Our analysis is of a parallelized queueing network simulation, where all simulated servers are under heavy load. LPs represent servers, and events occur when jobs either enter service, or are received by a queue. The servers use non-preemptive scheduling, and 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 allow for the data content and next destination of a serviced job to depend upon the contents and times of all jobs received by the LP prior to the time when that job enters 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. 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, Akyldiz 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 use unit cost for executing an event and the GAF model uses 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 quantitatively comparing our model results to GAF's. We do note that GAF's assumption of exponentially distributed event execution 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 ([Nicol 1991]), and is subsumed by Nicol's message-initiating model in his study of YAWNS ([Nicol 1993a]). 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.* ([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. This assumption is in keeping with normal practice in serial simulations; however, parallel simulations always pre-send messages if they can, for it increases the parallelism. Another difference is that in our model, a re-executed event chooses a new destination for its message uniformly at random, whereas the bonding model assumes it is directed to the same LP as before. Neither model is particularly realistic in this regard; it is not a critical facet of our model. Finally, the randomly uniform routing assumption is shared by the model studied by Felderman and Kleinrock [Felderman and Kleinrock 1991], who model time-stamp advancement and event execution time differently.

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. Finally, only the analysis in Nicol ([Nicol 1993a]) 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 ([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 $[t, w(t))$.

BTW is similar in the sense that it requires all LPs to synchronize at the upper edge of the optimistic window, the time of that window is understood by all processors. Whereas the original BTW algorithm proposed a window synchronization mechanism whose cost is linear in the number of LPs, we model the use of an algorithm with logarithmic cost, such as those described in [Nicol 1993b] or [Steinman 1992].

Every time an event is processed (whether initially or due to a rollback) it chooses a destination for its message, at random, regardless of the previous behavior of that event in the window. This allows a message's content and destination to be a sensitive function of the complete message history at LP i up to the time where the job enters service. This feature places BTW at a disadvantage in that lazy cancellation is ineffective. 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 neglect the cost of processing the anti-message at both the sender and receiver. However, in our model an anti-message is not recognized instantaneously by its recipient, but only after the recipient has processed all known events in its window. This feature makes sense when the cost of probing for a new message is high enough to govern that activity (which is the case on current distributed memory architectures).

3. ANALYSIS

The principle challenge in modeling optimistic protocols is to capture the effects of rollbacks, for, the occurrence of one straggler message may trigger one or more rollbacks. In our specific case we have the additional challenge of capturing the effect that BTW's global synchronization has on rollback propagation. This is a very difficult analytic problem; we make headway by use of some intuitive approximations. If a straggler arrives at time s and if the first post-rollback event at that LP has time stamp t, then for an additional rollback to be triggered the receive time of the message sent at time t must lie inside of the BTW synchronization window. The probability that this occurs depends on the relative positioning of time s within the window—the closer s is to the end of the window, the less likely it is to trigger another rollback. In fact, if we condition on knowing that a straggler message arrives at time s, under the model assumptions we can compute exactly the probabilistic effects of that straggler has on anti-message generation. Because the actual arrival time distribution for a message is intractably complex, we approximate its form and numerically compute parameter estimates of that form, thereby allowing computation of the effects of stragglers. The form is based on the intuition that the arrival time of a straggler message can be viewed as a sum of service times, because for any given message arriving at time t we can trace back a chain of service durations given to different jobs at different LPs. This approximation is novel among analyses of parallel simulation. As part of this calculation we use more standard approximations such as replacing a conditional binomial distribution with a mean-matched Poisson distribution, and assumption of independence between random variables whose correlation structure is very low due to randomizing effects of stochastic routing.

The approximated distributional form of message arrival times permits us to compute the probability distribution of the random number W of events executed (including re-executions) within a window of width A. W clearly 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 $[t, t + A)$ without receiving or being affected by any message not present at the synchronization. The set of messages sent during the first sweep with time-stamps in $[t, t + A)$ are defined to be in generation 1. As we have noted earlier, some messages sent during the first sweep will have time-stamps greater than $[t, t + A)$; these are explicitly excluded from consideration because they have no effect on processing in the present window. Each generation 1 message causes an LP to rollback and reprocess all events in the window that lie ahead of it's receive time-stamp. This reprocessing in turn generates another set of messages—ones in generation 2. Continuing in this vein, 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.

Our analysis is of a simulated non-preemptive queueing network with load so high that every server is always busy. When A is small, the job will arrive in one window, then wait for service through one or more windows. Discounting the possibility of a job going into service in the same window as it arrives, at time t an LP knows the number, the entry times, and the service times of all jobs it will place into service in window $[t, t+A)$. These jobs and their associated times remain unaltered throughout the processing of the window. However, the contents and destinations of messages the LP sends during $[t, t + A)$ are permitted to change as a function of the messages arriving in $[t, t + A)$. Observe then that the number of service entry events an LP has in $[t, t + A)$ is a random variable S that is Poisson distributed with mean $A\mu_s$. Since the routing of jobs is taken to be uniformly at random, the LP also knows of J job arrival events, where J is also Poisson with mean $A\mu_s$. S and J are independent.

Event reprocessing costs depend on how quickly the parallel simulator receives and reacts to straggler messages. For example, the analysis of Gupta *et al.* ([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 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. In our model, 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, then each is processed serially, incurring k separate recomputation costs. This aspect of the model concerns timing of message arrivals and their subsequent processing. While BTW need not behave in this way, it *may* do so as long as an LP has only a few events in a window and the communication lag is noteworthy.

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

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

Observe that the event re-executions counted by R_i with $i > 0$ are overhead. The distribution of each reprocessing cost R_i depends on the number of generation i messages. Given a total number N generation i messages in the system, the number arriving at an LP is a binomial $B(N, 1/P)$ random variable, P being the number of LPs. In principle we could carry the analysis forward retaining the binomial form. In practice there is a computational advantage to modeling the binomial with a Poisson random variable with matching mean. This approximation is standard when N is large and $1/P$ is small, which is the case for the early generations whose contributions dominate W.

As noted earlier, the distribution of a message's arrival time is too complex to handle exactly (owing to inescapable probabilistic dependencies). Our approximation of the arrival time distribution of a message notes that the 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 altered by this conditioning. 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.

If we chose a representative random generation-i arrival in $[t, t + A)$, what is its rank? The probability of the rank being k is the expected fraction $a_{i,k}$ 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

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 $s \leq A$
$F_j(s)$	Cumulative distribution function for Erlang-j
$B(n,p)$	A binomial random variable with parameters n and p
$H_j(s)$	Cumulative distribution function of Erlang-j conditioned on sum of first $(j-1)$ stages being less than A

Table 1. Summary of Notation

of an arbitrary generation i message as the mixture $t + \sum_{k=2}^{\infty} a_{i,k}\tilde{f}_k(s)$. We will show how to approximate the coefficients $\{a_{i,k}\}$.

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 $W(k)$. Under our model assumptions, the correlation between message arrival times at an LP are very slight; taking them to be independent 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.

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-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 $R_i(k)$. In our formulation, 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 again may view the placement in time of events on $[t,t+A)$ as that of k uniforms on $[t,t+A]$. 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 approximated 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 approximate the distribution of $R_i(k)$ as the random convolution of M independent instances of $r_i(k)$, M being Poisson with rate $E[G_i]$ (M's true distribution is $B(G_i^{(N)},1/N)$, $G_i^{(N)}$ being an N-fold convolution).

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.

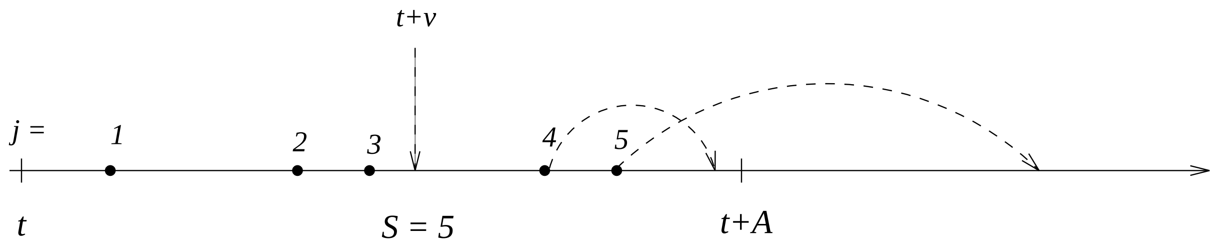

Fig. 1. Reprocessing of a rank 4 service-entry event generates a rank 5 message for the next generation.

Figure 1 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 falls within the window, the rank 5 event message does not. In order for there to be a rank 5 message generated, the 4^{th} 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 the 4^{th} 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 (with 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).

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. It is then 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.* ([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 $E[M_N(A)]$.

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 convergent numbers when $A \in (0, 2\mu_s)$. A precise complexity analysis of the computation involves enumerating the number of discrete points used to model distribution functions, a messy proposition considering that our code adaptively chooses the number of points needed. Practically speaking, the computational complexity is not large. Each data point on the curves we will illustrate required a small number of seconds of computation on a personal computer.

In the course of this research we experimented with several different forms for the approximate message arrival time distribution. The form developed here was validated against a large number of empirically observed observations, drawn from many values of N and A. Figure 2 provides a representative sample of the validation, comparing model predictions of $E[M_{64}(A)]$ and $E[M_{1024}(A)]$ with simulation-based measurements for varying values of A. The simulation is of a heavily loaded queueing network simulation, synchronized under BTW. 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 behavior 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. This is likely due to the model's overly pessimistic calculation that *every* straggler causes re-execution of *every* event ahead of within the window. With larger window sizes BTW increasingly deviates from this behavior, because recognizing two stragglers concurrently, only the one with smaller arrival time causes re-computation.

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 3, 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 within our model framework it does not make sense to

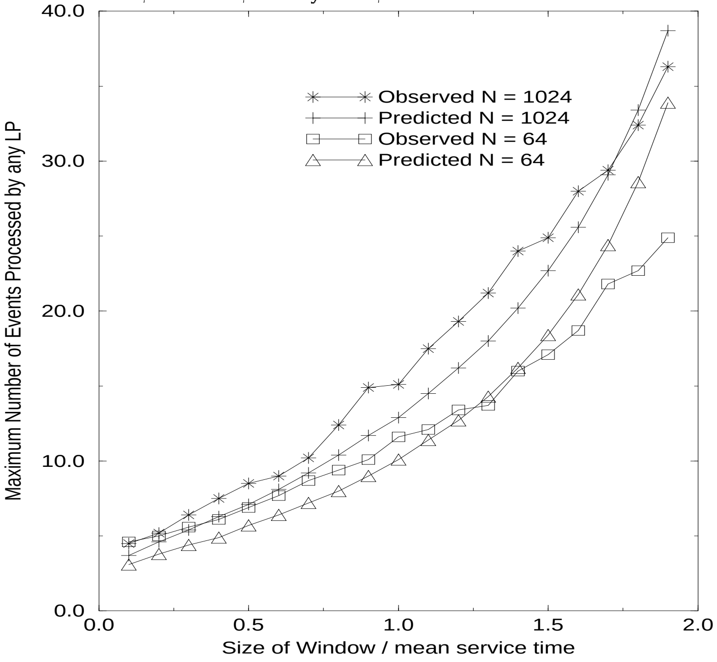

Fig. 2. Comparison of observed and predicted mean maximum events processed in a window by any LP.

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 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.* ([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 4 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 will show that it remains so in the presence of reasonable overheads), and in the remainder presume this equality.

To incorporate the effects of state-saving, we'll assume that the per-event cost of

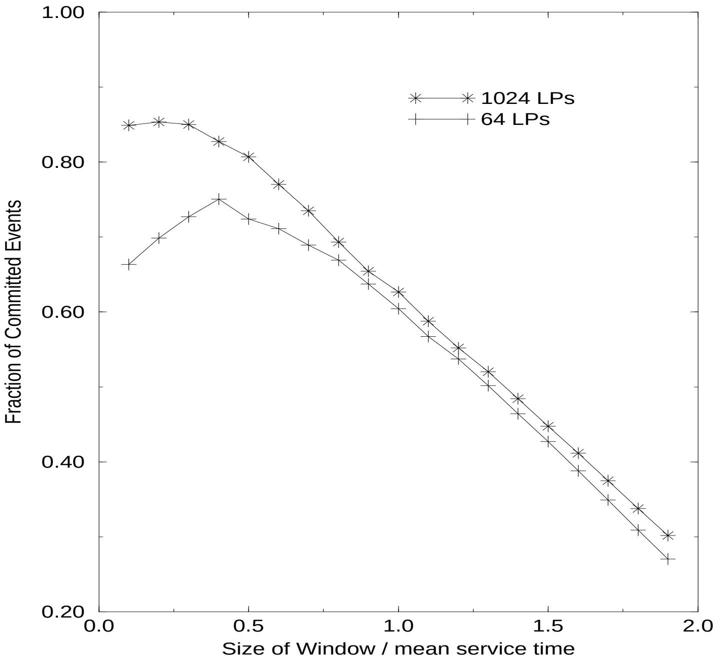

Fig. 3. Fraction of committed events as a function of A, for 64 and 1024 LPs.

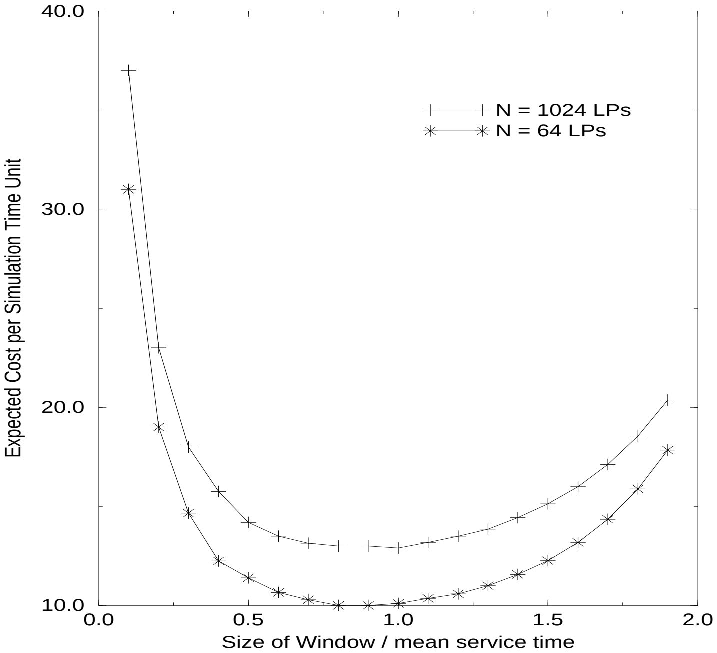

Fig. 4. $E[M_N(A)]/A$ as a function of A, for 64 and 1024 LPs.

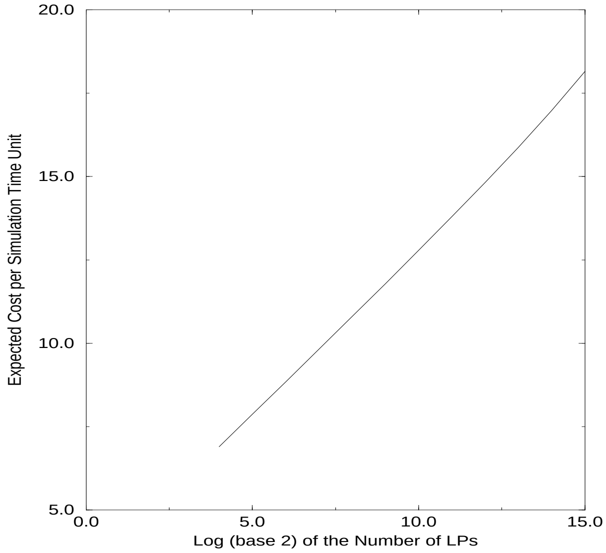

Fig. 5. $E[M_A(\mu_s)/\mu_s]$ as a function of $\log_2 N$

state-saving is a factor of α, so that the cost of executing n events with attendant state-saving is αn, $\alpha \geq 1$. Note that this model does not presume that state is saved each event; it only presumes that the aggregate state-saving overhead amortized over events is α. This is equivalent to saying that the effect of state-saving is to cause the execution to run at $1/\alpha$ the speed of an equivalent computation that does not state-save.

$E[M_N(A)]$ (and hence Figure 4) does not incorporate the cost of synchronization nor state-saving. The effect of state-saving cost α is to simply shift the curves illustrated in Figure 4 vertically by a factor of α; the optimal window size does not change. If we include a synchronization cost Z for each window, the true cost per simulation time unit is $\alpha E[M_N(A)]/A + Z/A$. Depending on the value of Z, the optimal window size may grow. However, an analysis of Figure 4 shows that Z must be very large to significantly alter the optimality of window sizes near μ_s. Having seen that $E[M_N(A)]/A$ is at least a quadratic function of A, simple calculus shows that the A minimizing $E[M_N(A)]/A + S/A$ is $O(Z^{1/3})$. More concretely we may ask for the value of Z forcing the optimal window size to be almost twice μ_s. This occurs roughly when the adjusted function is equal at the rightmost two points of a curve in Figure 4, (i.e., when $E[M_N(1.9)]/1.9+Z/1.9 = E[M_N(1.8)]/1.8+Z/1.8$; algebra yields $Z = 63.6$, a number reflecting a synchronization cost that is more than three times larger than the window's entire computation cost at that point, and is *ten* times larger than the cost of executing committed events. Overheads this large aren't practical; in the remainder we consider the case where Z is moderate (e.g. no more than the window's computation cost), and we may continue to use μ_s as a close-to-optimal window size.

In order to compare synchronization costs in YAWNS and BTW we must consider how synchronization is performed in an optimistic computation. A software solution described by Nicol ([Nicol 1991]) 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; this is in fact a reasonable way to program BTW, because following each synchronization the processors can determine whether there are further stragglers, and so decide whether to move on to the next window. Our experience with the optimistic barrier studied in ([Nicol 1993b]) is that it's cost is close to 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.

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 $E[M_N(\mu_s)]/\mu_s \approx \log N + 2.9$. Taking B as the execution cost of a conventional barrier synchronization the overall execution cost per unit simulation time given N LPs is

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

In the context of our earlier remarks concerning synchronization cost Z, this function is reasonable when $5B$ is of the order of the computation cost, say, $5B < \alpha(\log_2 N + 2.9)$.

Now consider YAWNS. Nicol ([Nicol 1993a]) 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, YAWNS' cost per unit simulation time is no greater than

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

We may use equations (3) and (4) to compare the approaches, given values for overhead costs. At a higher level we observe that BTW has an $O(\log_2 N)$ cost while YAWNS has an $O(\sqrt{N})$ cost. For sufficiently large N, BTW 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 BTW 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 BTW is better for $N > 128$. If $B = 1.0$ however, then BTW 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 BTW.

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

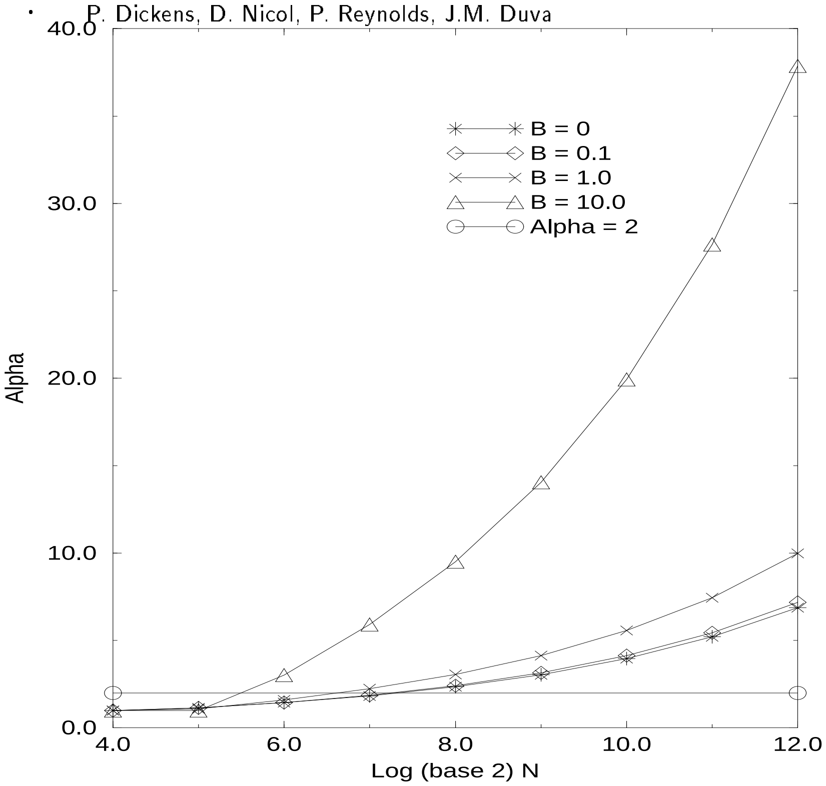

Fig. 6. Function specifying LP threshold N^* after which YOW is better than YAWNS.

seemingly minor change of assumptions defeats the assured asymptotic superiority of BTW, because it changes YAWNS $O(\sqrt{N})$ cost to $O(1/\gamma)$. The relative performance of YAWNS and BTW depend primarily then on α, B, and γ.

Next we show that by considering the effects of aggregating LPs onto processors, YAWNS again circumvents BTW'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. Prior analysis of YAWNS under our model assumptions shows that 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.*([Eick et al. 1993]) 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

$$D_{yawns}(N) \sim \frac{2\lambda(N) + B}{N/P}$$

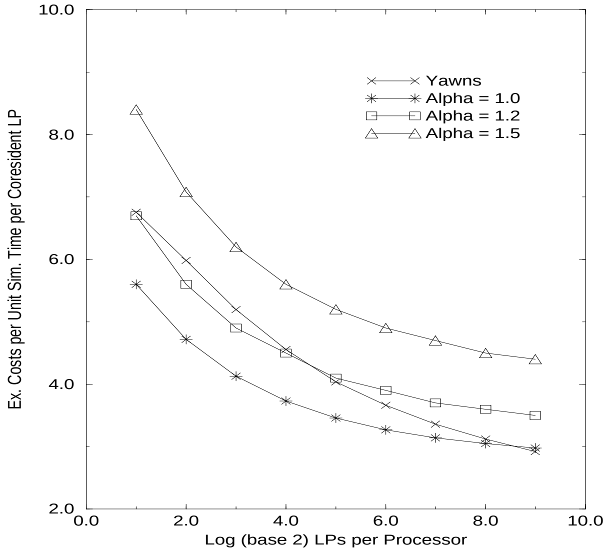

Fig. 7. YAWNS and BTW normalized cost per unit simulation time under aggregation as a function of $\log(N/P)$.

$$= 4 + \frac{B}{\lambda(N)}.$$

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 BTW, 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 BTW of $A = \mu_s$, we also computed costs assuming other window sizes. Differences from the presented data were small. When synchronization costs contribute little to the overhead cost under high loads, it is clear now that YAWNS can do better than BTW under high degrees of aggregation, or when state-saving overhead is significant.

It should also be noted that our model works against BTW 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 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. EXTENSIONS

We believe that our general analytic approach can be extended in some valuable ways. The first extension permits analysis of simulations where the routing probabilities are not uniform. Since our core analysis is of a single LP, and that analysis is driven by an assumed message arrival rate, one easily envisions constructing different workload probability distributions for LPs that have different arrival rates. One can then compute the expected maximum LP workload numerically. Note however that the assumption of independence between LP workloads may in special cases become less viable. We can anticipate the effect of such a change on the performance of YAWNS and BTW. The performance of both methods will degrade because of load imbalance in the arrival events. However, BTW's performance should suffer more, because of additional roll-back induced workload at LPs with higher-than-average arrival rates. One also imagines changes where LP message arrival rates remain the same, but the routing probabilities are changed to induce locality—consider a ring of LPs that communicate only with neighbors. So long as servers remain heavily loaded, our approach remains the same. The question of how the change affects an actual simulation is valid. Our experience with random-but-uniformly balanced routing in other contexts has been that differences exist, but they are small.

Another consideration is to relax the assumption that all servers are heavily loaded. We don't believe this changes our approach much; the number of service events in a window of size A becomes Poisson with rate $\rho\mu A$, ρ being the server utilization. We also need to account for the possibility of a job entering service upon arrival. These details appear to be manageable.

Lastly and most importantly, we believe it is possible to extend the analysis to explicitly model the communication delay between when a message is sent, and received. To accomplish this we envision modeling the arrival process at an LP as a non-homogeneous Poisson process, whose rate function depends on the communication lag. Our computational model will have to be based in physical time (given the probabilistic description at time τ, we compute its description at time $\tau + \delta$). However, such an approach offers the possibility of accounting for the phenomena that only the earliest known straggler initiates rollbacks. We hope that an extension of this type will sharpen our predictive power for larger windows.

6. CONCLUSIONS

We have analyzed a simple model of parallel simulation, to compare the performance of the conservative YAWNS synchronization protocol, and the optimistic Bounded Time Warp protocol. 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.

Our analysis predicts that BTW has 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 BTW's and

YAWNS' execution cost per unit simulation time, and observe that under the assumption of one LP per processor, BTW 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 BTW under moderate levels of aggregation, or when state-saving costs are non-negligible.

Far-reaching quantitative conclusions are questionable for a model of this type. For both YAWNS and BTW 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 on synchronization behavior.

ACKNOWLEDGMENTS

Our good friend and colleague, J. Mark Duva, passed away in the fall of 1995 before this paper was published. He will be sorely missed.

REFERENCES

Ayani, R. and Rajae, H. 1992. Parallel simulation using conservative time windows. In *Proceedings of the 1992 Winter Simulation Conference* (1992), pp. 709–717.

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

Chandy, K. and Misra, J. 1979. A case study in the design and verification of distributed programs. *EEE Transactions on Software Engineering SE-5*, 5 (May), 440–452.

Chandy, K. and Sherman, R. 1989. Conditional event approach to distributed simulation. In *Proceedings of the 1989 SCS Multiconference on Distributed Simulation* (1989), pp. 93–99.

Eick, S., Greenberg, A., Lubachevsky, B., and Weiss, A. 1993. Synchronous relaxation for parallel simulations with applications to circuit-switched networks. *ACM Transactions on Modeling and Computer Simulation 3*, 4 (October), 287–314.

Felderman, R. and Kleinrock, L. 1991. Bounds and approximations for self-initiating distributed simulation without lookahead. *ACM Transactions on Modeling and Computer Simulation 1*, 4 (October), 386–406.

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

GUPTA, A., AKYLDIZ, I., AND FUJIMOTO, R. 1991. Performance analysis of time warp with multiple homogeneous processors. *IEEE Transactions on Software Engineering 17*, 10 (October), 1013–1027.

JEFFERSON, D. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems 7*, 3, 404–425.

LUBACHEVSKY, B. 1988. Bounded lag distributed discrete event simulation. In *Proceedings of the 1988 SCS Multiconference on Distributed Simulation* (1988), pp. 183–191.

MISRA, J. 1986. Distributed discrete-event simulation. *Computing Surveys 18*, 39–64.

NICOL, D. 1991. Performance bounds on parallel self-initiating discrete event simulations. *ACM Transactions on Modeling and Computer Simulation 1*, 1.

NICOL, D. 1993a. The cost of conservative synchronization in parallel discrete event simulation. *Journal of the ACM 40*, 7 (April), 304–333.

NICOL, D. 1993b. Global synchronization for optimistic parallel discrete event simulation. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation* (1993), pp. 27–34.

NICOL, D. AND FUJIMOTO, R. 1994. Parallel simulation today. *Annals of Operations Research 53*, 249–286.

PEACOCK, J., WONG, J., AND MANNING, E. 1979. Distributed simulation using a network of processors. *Computer Networks*, 44–56.

REYNOLDS, P. 1988. A spectrum of options for parallel simulation. In *Proceedings of the 1988 Winter Simulation Conference* (1988), pp. 325–332.

RIGHTER, R. AND WALRAND, J. 1989. Distributed simulation of discrete event systems. *Proceedings of the IEEE 77*, 1 (January).

SOKOL, L., BRISCOE, D., AND WIELAND, A. 1988. Mtw: A strategy for scheduling discrete simulation events for concurrent execution. In *Proceedings of the 1988 SCS Multiconference on Distributed Simulation* (1988), pp. 169–173.

STEINMAN, J. 1991. Speedes: Synchronous parallel environment for emulation and discrete event simulation. In *Proceedings of the SCS Western Multiconference on Advances in Parallel and Distributed Simulation* (1991), pp. 95–103.

STEINMAN, J. 1992. Speedes: A unified approach to parallel simulation. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation* (1992), pp. 75–84.

TURNER, S. AND XU, M. 1992. Performance evaluation of the bounded time warp algorithm. In *Proceedings of the 6th Workshop on Parallel and Distributed Simulation* (1992), pp. 117–126.