Proceedings of the 1997 Winter Simulation Conference
ed. S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson

MINIMUM COST ADAPTIVE SYNCHRONIZATION: EXPERIMENTS WITH THE PARASOL SYSTEM

Edward Mascarenhas

Silicon Graphics Computer Systems
2011 N. Shoreline Blvd. MS 510
Mountain View, CA 94043
U.S.A.

Felipe Knop

IBM Corporation
522 South Road, MS P963
Poughkeepsie, NY 12601
U.S.A.

Vernon Rego

Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
U.S.A.

ABSTRACT

We present a novel adaptive synchronization algorithm, called the minimum average cost (MAC) algorithm, in the context of the PARASOL parallel simulation system. PARASOL is a multithreaded system for parallel simulation on shared- and distributed-memory environments, designed to support domain-specific Simulation Object Libraries. The proposed MAC algorithm is based on minimizing the cost of synchronization delay and rollback at a process, whenever its simulation driver must decide whether to either proceed optimistically or to delay processing. In the former case the risk is rollback cost, in the event of a straggler's arrival. In the latter case the risk is unnecessary delay, in the event a late-comer is not a straggler. In addition to the MAC algorithm and an optimal delay computation model, we report on some early experiments comparing the performance of MAC-based adaptive synchronization to optimistic synchronization.

1 INTRODUCTION

The problem of parallelizing discrete event simulations is known to be a challenging one. The challenge lies in moving a multiprocessor simulation forward to completion as quickly as possible in real time, while satisfying event-related synchronization constraints in simulation time. The work described in this paper is motivated by the need for a tool that will enable low-effort and practical parallel simulation. If a physical system of interest is viewed as a system of interacting *physical* processes, a simulator for such a system consists of interacting *logical* processes (LPs). Each LP progresses from one event to the next in simulation time (also called *local virtual time* or LVT), where such time is tracked by a clock that is local to the LP.

In a physical system, dynamic entities move between physical processes. In the logical system, these entities are represented by *active-transactions*: user-level threads (Mascarenhas and Rego 1996) which may move between LPs. By binding simulation time-stamps – which indicate occurrence-times of simulation events – to transactions, a flow of transactions between LPs enables LPs to communicate and thus to synchronize with one another. By synchronzing LPs, a parallel simulator can eliminate invalid simulation trajectories generated by *causality errors*. A causality error occurs at an LP if this LP finds itself in violation of the fundamental simulation rule: *events must be processed in order of non-decreasing time*. The idea is to either prevent causality errors from ever occurring (i.e., conservative synchronization), or to allow their occurrence but recover from error-related effects when such errors are detected (i.e., optimistic synchronization). The major focus of parallel simulation research, in recent years, has centered around assessment of these two synchronization methods.

In the *conservative* approach (Chandy and Misra 1979), events are executed strictly in order of their occurrence in simulation time. This approach prevents causality errors from occurring, but may lead to deadlock; an LP may wait for events that never arrive. LPs that have no messages to send to other LPs may use special "informant" messages, called *Null* messages, to prevent deadlocks (Chandy and Misra 1979). In the *optimistic* (Time Warp) (Jefferson 1985) algorithm, an LP processes events as event messages arrive from other LPs until a causality error is detected: a message arrives with a time-stamp less than the LP's LVT. Such a message – called a *straggler* – renders all simulation processing done by the LP as potentially invalid, starting from the time-stamp of the straggler. When this occurs, the computation is rolled back and restarted from an appropriate, previously saved and error-free state. *State-saving* and *rollback* mechanisms enable implicit LP synchronization. The minimum of the LVTs of all LPs and time-stamps of event messages in transit is said to define the global virtual time (GVT), so that that rollbacks

to states prior to the GVT are impossible. Repeated parallel computation of the GVT allows memory to be reclaimed (e.g., saved state associated with a time that is less than the GVT is no longer required), in a procedure known as *fossil collection*. I/O operations with time-stamps less than the GVT may be committed.

We present a novel adaptive synchronization algorithm, called the minimum average cost (MAC) algorithm, in the context of the ParaSol parallel simulation system. ParaSol is a multithreaded system for parallel simulation on shared- and distributed-memory environments, designed to support domain-specific Simulation Object Libraries. A brief overview of the ParaSol system appears in Mascarenhas et al. (1997).- Additional details on the ParaSol system can be found in Knop (1996) and Mascarenhas (1996). The proposed MAC algorithm is based on minimizing the cost of synchronization delay and rollback at a process, whenever its simulation driver must decide whether to either proceed optimistically or to delay processing. In the former case the risk is rollback cost, in the event of a straggler's arrival. In the latter case the risk is unnecessary delay, in the event a late-comer is not a straggler. In addition to the MAC algorithm and an optimal delay computation model, we report on some early experiments comparing the performance of MAC-based adaptive synchronization to optimistic synchronization.

2- RELATED WORK

Adaptive protocols can limit the uncontrolled execution of incorrect events, or limit the optimism of the optimistic protocol. Protocols that limit optimism have been classified as: window based, space based, penalty based, knowledge based, probabilistic, and state based (Srinivasan and Reynolds 1995).

Proposals for adaptive synchronization, based either on local channel-state information or on global information, have recently begun to appear. In the Local Adaptive Protocol (LAP), LPs compute a real-time blocking window (RTBW) based on the average event-arrival rates in real and virtual time (Hamnes and Tripathi 1994). An LP blocks if it finds an empty channel for which the increment in virtual time between the last event processed and the next candidate event is larger than the average virtual inter-arrival time. Adaptive control is achieved via a channel specific constant c, enabling various "degrees" of optimism. The constant c is chosen in a way that maximizes the rate of simulation time advance. In experimental work, this protocol has been shown to exhibit good performance (Hamnes and Tripathi 1994).

Ferscha and Chiola (1994) propose a probabilistic control of optimism, based on the determination of a local virtual time window. Events within the window are executed with a given probability, computed using the virtual time-stamps of arriving messages. Yet another proposed approach is to predict the time-stamp of the next message arrival on a given channel, using message-arrival history. Only available events with time-stamps smaller than the predicted time-stamp are processed (Ferscha 1995). A confidence level assigned to each estimate enables the execution of each event with a certain probability. Several different methods of predicting the time-stamp of the next arrival have been proposed. These include estimates based on the arithmetic mean, median, exponentially smoothed average, and computationally intensive auto-regressive and integrated moving-average forecasting methods. Some success was reported with these methods (Ferscha 1995, Ferscha and Chiola 1994).

Another proposed strategy for limiting optimism is based on memory management (Das and Fujimoto 1994). Here, optimism is limited in an indirect manner, by controlling the amount of memory provided to an LP. The adaptive protocol attempts to provide each LP with only as much memory as necessary for optimal performance. The idea is that giving an LP unchecked access to memory each time such memory is requested, subject to machine limitations, may hamper progress at other LPs.

3 ADAPTIVE SYNCHRONIZATION

Both conservative as well as optimistic synchronization protocols have their advantages and disadvantages (Fujimoto 1990). Which protocol is better depends on the characteristics of a given application (number of simulation objects, density of messages, etc.), input data, and the run-time environment (processor speeds, communication latency, etc.). There is a need for synchronization schemes that work well in an application-independent way. Adaptive synchronization methods, in particular, provide a framework for tailoring synchronization to the peculiarities of a given application. Based on decisions made with run-time data, such methods offer a dynamic combination of optimistic and conservative synchronization, and have been reported to exhibit poor to reasonable performance in previous studies (Das and Fujimoto 1994, Ferscha 1995, Hamnes and Tripathi 1994, Srinivasan and Reynolds 1995). Of particular importance is the fact that extreme forms of optimistic progress and conservative blocking can be avoided.

In this section, we present a new and dynamic method for adaptive synchronization. While past experimental work has generally involved specialized ex-

perimental setups for answering specific questions on synchronization schemes, we present an experimental study in the context of the ParaSol system. Indeed, the system was designed for practical use and for experimentation. The proposed method is based, in part, on some ideas presented in Ball and Hoyt (1990). In essence, the proposal is to minimize loss in either waiting for late transactions or in undoing work. Before processing a transaction, an LP examines its input channels and computes an optimal delay interval. Transaction processing is then suspended for a period based on this interval. In this way, the LP attempts to minimize the average cost of rollback and delay. During this period, certain other simulation activities may proceed: message processing, state-saving, and fossil-collection. If the delay interval computed turns out to be too small, the LP may simply spin in a busy loop.

We present a model based on a rollback cost and delay analysis, similar to that presented in Ball and Hoyt (1990), and Ferscha and Luthi (1995). Our work differs from theirs in some respects, such as in how expected cost is computed. We estimate transaction interarrival distributions (both virtual time and real time) on input channels and use these, along with other costs, to estimate rollback probability and cost. In Ferscha and Luthi's approach (1995), rollback probability is estimated by tracking rollback frequency over a discrete real-time/virtual-time plane. Though we cannot argue that virtual and real interarrival times of transactions at an LP are independent, we have some empirical evidence which shows a weak correlation in situations that we tested. When it is not reasonable to assume independence, an approximate rollback probability based on estimation of a joint distribution is warranted. Otherwise, marginal distributions – which are easier to estimate – may be used.

3.1 The Minimum Average Cost (MAC) Model

In a ParaSol simulation, each LP runs as a thread – distinct from threads which implement transactions – within a (Unix) process. A single process may host many LPs. Transactions move between LPs, accessing simulation objects and possibly modifying their state. Transactions are selected for processing based on their time-stamps. Each process runs a simulation driver which examines all LPs internal to the process, to select the LP holding the event (related to a transaction) with the smallest time-stamp. If one is found, this LP is given the CPU, and LP processing proceeds. After an LP has processed its transaction, control returns to the driver and the procedure is repeated.

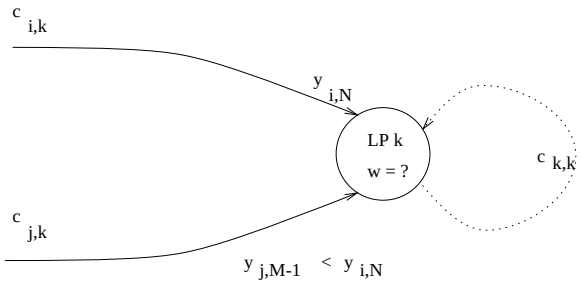


Figure 1: LP k has two input channels, of which channel $c_{j,k}$ is empty. The last transaction executed at LP k from channel $c_{j,k}$ had a time-stamp $y_{j,M-1} < y_{i,N}$. The LP is considering whether to execute a transaction from channel $c_{i,k}$ at time $y_{i,N}$ immediately or wait for real time w.

3.1.1 Conservative Synchronization

In a conservative simulation, a process's driver does not examine an LP unless the LP has determined, without potential for a causality error, its own minimum time-stamp transaction. If there is no potential for a causality error at an LP, execution in a process will also be causality-error free. For example, consider an LP k which may receive transactions from at most two other (source) LPs, say LP i and LP j. To source i it assigns an input channel $c_{i,k}$, and to source j it assigns an input channel $c_{j,k}$, as shown in Figure 1. In a conservative simulation, time-stamps on transactions arriving on any given channel are ordered in (increasing) time. LP k may also generate its own transactions, process them and send them on to other LPs. To ensure causality, a process's driver cannot obtain time-stamp information from LP k to process the next transaction unless this LP has at least one pending transaction on each of its input channels.

3.1.2 Optimistic Synchronization

In an optimistic simulation, each LP will be examined by a process's driver even though one or more of the LP's input channels is empty. For example, assume that a process hosting a single LP, say LP k, examines the LP and finds channel $c_{j,k}$ empty. Even though next-event information from LP j is currently unavailable, LP k will be scheduled to run, to process its minimum time-stamped transaction. This transaction is either an internal transaction, or one from channel $c_{i,k}$. If channel $c_{j,k}$ yields a transaction with a smaller time stamp (i.e., a straggler) after the first transaction has been accepted for processing, then

upon completion of transaction processing the system detects a causality error. The effects of the error are undone: the system sends out 'terminator' transactions (anti-transactions) to locate and destroy invalid emigrants, rolls back to a valid state and redoes some transaction processing (called 'coast-forwarding') before accommodating the straggler. After the causality error has been corrected, new emigrants may be generated.

3.1.3- Adaptive Synchronization

The optimistic approach outlined above has two drawbacks. First, it is difficult to justify optimistic progress based on arbitrary transaction availability or even to relate it directly to progress in simulation time. Stragglers cause rollbacks, and rollback costs can be high; terminator generation, rollback and coast-forward phases may further hinder simulation progress by delaying processing of new incoming transactions. Second, it can be argued that the approach ignores available data, such as rollback and coast-forward costs, and probability of straggler arrival, potentially ignoring strategies for enhancing the average rate at which the parallel simulation progresses.

In the example described above, the driver has an alternative to scheduling LP k's processing optimistically. Based on local and repeatedly updated data, the driver may suspend processing at the LP for a given window of time w. An appropriately chosen value of w will minimize the local costs associated with processing delay or rollback at LP k. With $w = 0$, this strategy is equivalent to purely optimistic processing. If w is sufficiently large to guarantee a zero probability of straggler arrival, the strategy is equivalent to conservative processing. The intent is to minimize the amount of time a process spends in either waiting for a late transaction that is not a straggler, or in undoing work caused by premature transaction processing. In the following, we propose an adaptive synchronization model based on the Minimum Average Cost (MAC) of delay and rollback processing. We also give some empirical justification for such a model.

3.1.4- MAC Adaptive Synchronization: The Two Source Case

Assume that at virtual time t, the driver examines LP k and finds channel $c_{j,k}$ empty, and channel $c_{i,k}$ offering a transaction. Further, assume that the latter transaction is the only available transaction at LP k, and that it contains the N-th (virtual) time-stamp $y_{i,N}$ arriving on $c_{i,k}$. Operating with an optimistic protocol, LP k is allowed to process the transaction on $c_{i,k}$ without delay. A rollback will ensue only if a straggler arrives from LP j during or after processing of LP i's transaction. Suppose that the next transaction to arrive on $c_{j,k}$ is its M-th transaction, and this arrival occurs after time t. Then this arriving transaction is a straggler if and only if the transaction with time-stamp $y_{i,N}$ has started or completed processing, and the new arrival has a time-stamp $y_{j,M} < y_{i,N}$.

Transactions arriving at LP k from each source LP j arrive in sequence. For each input channel $c_{j,k}$, let the sequence of transaction (random) arrival times be denoted by $\{X_{j,n}; n \geq 0\}$, and let the sequence of (random) time-stamps on these transactions be denoted by $\{Y_{j,n}; n \geq 0\}$. Define $R_{j,n} \stackrel{\Delta}{=} X_{j,n} - X_{j,n-1}$ and $T_{j,n} \stackrel{\Delta}{=} Y_{j,n} - Y_{j,n-1}$ to be random variables denoting the interarrival time and time-stamp of the n-th transaction coming in on $c_{j,k}$, respectively, for any j, and $n \geq 1$. We assume that $\{R_{j,n}; n \geq 1\}$ and $\{T_{j,n}; n \geq 1\}$ are stationary sequences (Law and Kelton 1982).

Given some history $\{x_{j,n}; n < M\}$ of transaction arrival times, and also some history $\{y_{j,n}; n < M\}$ of transaction time-stamps, we would like to estimate the probability that the next (i.e., M-th) arrival on $c_{j,k}$, arriving after time t, is a straggler. We do this by estimating the distribution of random variable $T_{j,M}$ at time t, using either the complete history or some subset of the history of the time-stamp sequence. Given that $c_{j,k}$ is empty at time t, define $S_j(t)$ to be a Bernoulli random variable which is 1 if, at time t, the anticipated late-comer from LP j is a straggler, and 0 otherwise. The probability

$$P\{S_j(t) = 1\} = P\{T_{j,M} < y_{i,N} - y_{j,M-1} \mid Y_{i,N} = y_{i,N}, X_{j,M} > t\} \quad (1)$$

where $y_{i,N}$, $y_{j,M-1}$ are known, and the probability involving $T_{j,M}$ is estimated at time t.

Given only that $c_{j,k}$ is empty at time t, then under our assumption of stationarity, the conditional probability that the late-comer will arrive *after* time $(t+w)$ *and* turn out to be a straggler, is given by

$$P\{S_j(t+w) = 1\} = P\{T_{j,M} < y_{i,N} - y_{j,M-1}, R_{j,M} > t + w - x_{j,M-1} \mid R_{j,M} > t - x_{j,M-1}\} \quad (2)$$

where $R_{j,M} = X_{j,M} - x_{j,M-1}$.

If we assume that transaction time-stamps and transaction arrival times on each channel are independent, we may simplify the computation, to obtain

$$P\{S_j(t+w)=1\} = (P\{T_{j,M} < y_{i,N} - y_{j,M-1}\} * P\{R_{j,M} > t+w-x_{j,M-1}\})/ P\{R_{j,M} > t-x_{j,M-1}\} \quad (3)$$

For a fixed value of t, $P\{R_j > w+\epsilon\} < P\{R_j > w\}$ for any $\epsilon > 0$. Under independence, because the first probability in the product shown in Equation 3 is unaffected by w, the random variable $S_j(t+w)$ is stochastically decreasing in w. If the cost of rollback B_{t+w} incurred by premature processing of LP i's transaction at time $(t+w)$ is known, then the actual cost $C(t,w)$ of delaying LP k's processing from time t to time $(t+w)$ can be expressed as

$$\begin{aligned} C(t,w) &= I_{\{S_j(t+w)=0\}} * w \\ &\quad + I_{\{S_j(t+w)=1\}} * (w + B_{t+w}) \\ &= w + I_{\{S_j(t+w)=1\}} * B_{t+w} \end{aligned} \quad (4)$$

where I is an indicator function for the specified event. The expected cost $E[C(t,w)]$ is thus

$$E[C(t,w)] = w + P\{S_j(t+w)=1\} * \bar{b}_t \quad (5)$$

where $\bar{b}_t$, an estimate of $E[B_t]$, is used to approximate $E[B_{t+w}]$.

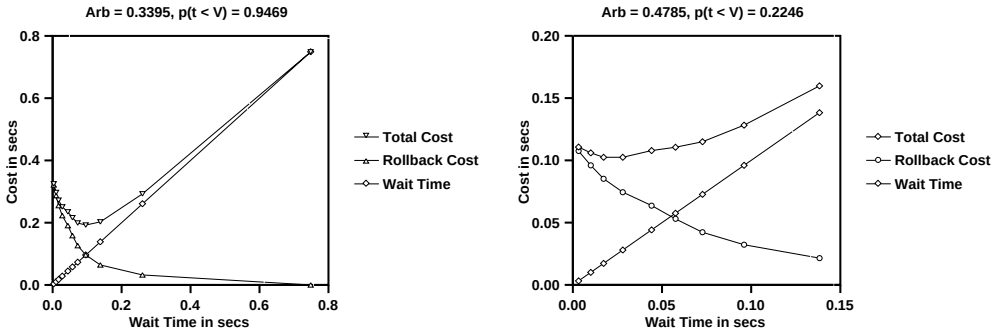
When the driver finds channel $c_{j,k}$ at LP k empty at time t, it resorts to the following strategy. Using run-time data to make estimates of probabilities and rollback cost , it determines the value of w that minimizes $E[C(t,w)]$. Transaction processing at LP k is suspended between time t and time $(t+w)$ – though other tasks, such as fossil collection, state saving, etc., may proceed – since this action minimizes the expected cost associated with delay and premature execution of transactions. Processing at LP k is resumed at time $(t+w)$, regardless of whether the late-comer does or does not arrive on $c_{j,k}$ by this time. If the late-comer does arrive by this time, uncertainty is removed, and the right transaction can be processed at time $(t+w)$. If not, the expected cost of delay and rollback is smallest at time $(t+w)$.

If channel $c_{j,k}$ at LP k remains empty while $c_{i,k}$ continues to generate transactions, the expected rollback cost component in Equation 5 increases. Suppose, for example, that the driver finds $c_{j,k}$ still empty at some time $t' > t$. If a delay interval is to be computed at time t', the value of w minimizing $E[C(t',w)]$ is used. Given that $c_{j,k}$ remains empty in $[t,t']$, $S_j(u+w)$ is stochastically increasing in u, $u \in [t,t']$, and $B_{t'} > B_t$. As a result, $C(t',w) > C(t,w)$. To compensate for this effect, the delay w at LP k tends to increase, making LPs reluctant to race ahead optimistically, minimizing risk and cost of rollback. The net result is that the LVTs of different LPs tend to remain closer to one another than in optimistic executions.

If a process's driver finds only internal channel $c_{k,k}$ empty at LP k, transaction processing proceeds at the LP without delay. This is because LP k cannot generate transactions that will effect its own past. Similarly, if no channels are found empty, transaction processing proceeds without delay, since the minimum time-stamp event can be determined. In Figures 2(a) and (b) are shown typical behaviors of the expected cost $E[C(t,w)]$ computed at some fixed time instant t at which the driver seeks a delay interval. The expected cost is graphed as a function of w. At time t, the driver estimates $\bar{b}_t$ and the virtual time component of the probability product in Equation 3. Once this is done, an estimate of the probability involving real time, in the same equation, is computed for select values of w. Note that the latter probability decreases with increasing w, to ultimately reach 0; this defines an upper bound on delay. The virtual time probability obtained for Figure 2(a) is higher than that obtained for Figure 2(b), because of which Figure 2(a) yields a larger minimum cost delay interval than the latter.

3.1.5- MAC Adaptive Synchronization: The General Case

In general, LP k may be fed transactions from several sources, and on examining this LP, the driver may find r of its channels empty, where $r \geq 1$. Assume that channel $c_{i,k}$ is found to have the minimum time-stamp transaction, and that input channels from LPs in the set $E = \{j_1, j_2, j_3, \ldots, j_r\}$ are all found empty. The driver must determine an optimal delay interval that accounts for potential stragglers from *all* LPs in E. One strategy is to first compute an optimal delay interval w_{i,j_ℓ}, by pairing $c_{i,k}$ and $c_{j_\ell,k}$ and applying the two-source analysis described above, for each $\ell \in E$. That is, w_{i,j_ℓ} is the delay interval obtained given that $c_{j_\ell,k}$ is the only empty channel, and $c_{i,k}$ offers a transaction with the smallest time-stamp. Then, LP k can be made to delay processing for time $w = \max_{\ell \in E}\ w_{i,j_\ell}$ (this scheme was followed in Hamnes and Tripathi (1994)). This strategy will not give a correct value for the delay – a delay that will minimize the cost – and the computational expense is too large. An exact expected cost function that simultaneously accounts for all empty channels can be developed. Assume, as done above, that $c_{i,k}$ offers the minimum time-stamp transaction. Define $S_{i,E}(t)$ to be a Bernoulli random variable which is 1



(a) (b)

Figure 2: The variation in the total costs of waiting and expected rollback, for a closed queuing network is shown. Arb is the average rollback cost and $p(t < V)$ is the virtual time related probability.

if, at time t, an anticipated late-comer from any LP in E is a straggler, and 0 otherwise. Then the probability that a straggler does not arrive at LP k in the interval $[t, w)$ is given by

$$P\{S_{i,E}(t+w) = 0\} = P\{S_{i,j_1}(t+w) = 0, \ldots, S_{i,j_r}(t+w) = 0\}. \quad (6)$$

If stragglers at LP k arrive independently, from the different source LPs in E, then

$$P\{S_{i,E}(t+w) = 0\} = \prod_{\ell \in E} P\{S_{i,j\ell}(t+w) = 0\} \quad (7)$$

is obtained as a simple product of known probabilities. The probability $P\{S_{i,E}(t+w) = 1\}$ can now be used in Equation 5 to obtain the total expected cost, and finally an optimal delay interval $w_{i,E}$ which minimizes this cost.

4 PERFORMANCE

The execution environment consisted of a cluster of SPARCstation 5 workstations connected over an Ethernet. The nodes in this environment are SPARC processors running at 70MHz, and each workstation has a memory of 32 MB. For convenience we identify this execution environment as CLUS.

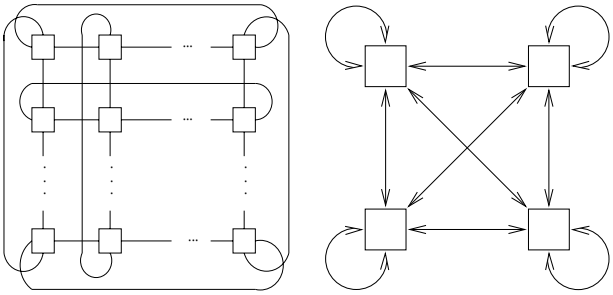
To test the new methods, we chose three different examples of *closed* queuing systems. Queuing networks are difficult to simulate in parallel because they exhibit low computation granularity and high node interaction. The applications consist of Facilities (a Facility consists of a server and a queue, from the queuing domain), each initialized with a given number of jobs. Upon service completion at a Facility, a job moves on to some other Facility in the network, depending on probabilities and paths specified in the model. In these examples, we assume that the service discipline at each Facility is FIFO. Upon leaving a Facility, a job selects a destination Facility based on a uniform distribution. The size of the configuration – the number of Facilities in the model is input from a file – is a control parameter and is fixed for a run. The example configurations are:

CQN. This is a closed queuing network configuration, shown in Mascarenhas et al. (1997).

TORUS. A torus consists of Facilities arranged in a two dimensional mesh, as shown in Figure 3 (a). Each Facility has four outgoing and four incoming channels. The probability of a job leaving a Facility on a given outgoing channel can be defined through an input parameter file. Thus, to reduce the number of channels some outgoing probabilities may be set to zero. Unless mentioned otherwise, we use a branching probability of 0.25 on each of 4 outgoing channels.

COMP. This defines a completely connected queuing network, shown in Figure 3 (b) (for the 4 Facility case). After service each job can be routed to any one of the Facilities in the system. This application is "difficult" for optimistic parallel simulation because the occurrence of rollback is high.

Besides the application type and network size, other parameters that can be varied are *transaction density*, *average service time*, *service distribution*, and *run length*. The transaction density (denoted by TD in figures) is the ratio of the total number of jobs in the system to the total number of Facilities in the system. Service time distributions can be changed. In our examples we use a biased exponential service time distribution, given by $service\ time = rT + exp((1-r)(T + factor * lpid))$, where $T = 10$ is the average service time, and $r = 0.01$. The granularity $factor$ in the service time expression allows us to vary the mean service time across LPs. Facilities hosted by a LP with a larger *lpid* will have a larger average service time, if $factor$ is non-zero.



(a)- (b)

Figure 3: Torus and Complete Queuing Networks

Unless mentioned otherwise, the value of *factor* is set to zero, i.e., average service times are same at all facilities.

The measurement metric chosen is the *execution time* of the simulation, measured by the cost and statistics module in the ParaSol kernel. Components of the total execution time are also used when appropriate. Each simulation run is terminated when the GVT exceeds a specified value.

To have some degree of confidence in our results we repeated experiments on the CLUS environment at least 10 times. The standard deviations were low. For example, for the experiments reported next, the standard deviations in execution time were less than 1% in the CLUS environment.

Models were partitioned equally across processors. Objects in the model were assigned to processes in a round robin fashion, moving from left to right across the queuing network. The number of LPs per process was set to one, except in the case of CQN, where no more than one complete row of Facilities in a switch was assigned to one LP.

We performed experiments comparing our adaptive model with the optimistic model in the CLUS environment. The rollback overhead is high in this environment, and the potential for improvement over the optimistic execution is high. We use rollback costs, execution time, and adaptive synchronization costs to compare performance.

Figure 4 shows the comparison between the adaptive and optimistic synchronization methods on a cluster of 4 workstations. It is clear from the figures that the adaptive methods are successful in reducing the costs of rollback in all cases. The number of rollbacks is also reduced. The reduction in rollback cost is about 25% in most cases. When rollback overheads are high as in the COMP application, the reduction is as much as 57%. The effect of a reduction in rollback overhead is offset to some extent by the cost of adaptive synchronization. As a percentage of the total time taken by the optimistic runs this is in the range of 1 to 2 % for the TORUS and COMP networks and between 4 – 12% for the CQN network. The graphs also show that the time spent in adaptive synchronization as a proportion of the rollback cost is larger for the CQN application than the TORUS and the COMP. One of the reasons the CQN application does not rollback as often is because of the lower connectivity among servers. For this reason we observe minor or no improvement in the execution time for an adaptive run of the CQN network.

5- CONCLUSION

In this paper we have presented a cost based adaptive model for reducing overheads of synchronization in ParaSol. Our adaptive synchronization model is able to reduce rollback costs, on an average, by about 25% for the applications that were studied. An important aspect of our adaptive synchronization model is that it makes few assumptions. We assume stationarity of the interarrival time distributions of transactions in real-time and virtual-time. As a model simplification we also assumed that these distributions are independent of each other. Additional experimentation in other execution environments with other simulations are required to study the performance of the adaptive model. The model can also be improved to take into account the effect of antimessages and use global information when it is available at low cost.

ACKNOWLEDGMENTS

This research was supported in part by ONR-9310233, BMDO-34798-MA, and NSF-CCR 9311862. The second author was supported in part by CNPq-Brazil, grant 260059 /91.9.

REFERENCES

Ball, D., and S. Hoyt. 1990. The adaptive time-warp concurrency control algorithm. In *Proceedings of the SCS MultiConference on Distributed Simulation*, 174–177.

Chandy, K. M., and J. Misra. 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, 5(5):440–452.

Das, S. R., and R. M. Fujimoto. 1994. An adaptive memory management protocol for time warp parallel simulation. In *Proceedings of the 1994 ACM Sigmetrics Conference on Measurement and Modelling of Computer Systems*, 201–210.

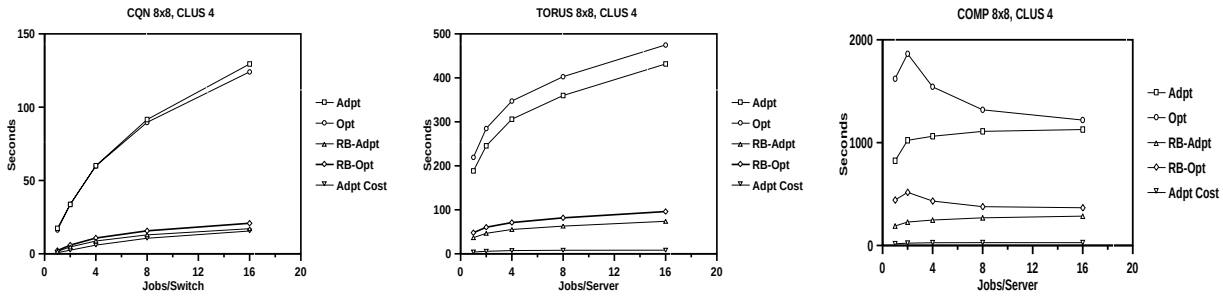


Figure 4: Performance of CQN, TORUS and COMP as the Transaction Density is Varied

Ferscha, A. 1995. Probabilistic adaptive direct optimism control in time warp. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, 120–129.

Ferscha, A., and G. Chiola. 1994. Self adaptive logical processes: The probabilistic distributed simulation protocol. In *Proceedings of the 27th Annual Simulation Symposium*, pages 78–88.

Ferscha, A., and J. Luthi. 1995. Estimating rollback overhead for optimism control in time warp. In *Proceedings of the 28th Annual Simulation Symposium*, 2–12.

Fujimoto, R. 1990. Parallel discrete event simulation. *CACM*, 33(10):30–53.

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

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

Knop, F. 1996. Software Architectures for Fault-Tolerant Replications and Multithreaded Decompositions: Experiments with Practical Parallel Simulation. PhD thesis, Department of Computer Sciences, Purdue University.

Law, A. M., and W. D. Kelton. 1982. *Simulation Modeling and Analysis*. McGraw-Hill.

Mascarenhas, E. 1996. A System for Multithreaded Parallel Simulation and Computation with Migrant Threads and Objects. PhD thesis, Department of Computer Sciences, Purdue University.

Mascarenhas, E., and V. Rego. 1996. Ariadne: Architecture of a portable threads system supporting thread migration. *Software-Practice and Experience*, 26(3):327–356.

Mascarenhas, E., F. Knop, R. Pasquini, and V. Rego. 1997. Checkpoint and recovery methods in the ParaSol simulation system. In *Proceedings of the Winter Simulation Conference*.

Srinivasan, S., and P. F. Reynolds Jr. 1995. NPSI adaptive synchronizaton algorithms for PDES. In *Proceedings of the Winter Simulation Conference*, 658–665.

AUTHOR BIOGRAPHIES

EDWARD MASCARENHAS, is a Member of Technical Staff at Silicon Graphics Computer Systems. He received a Masters degree in Industrial Engineering from NITIE (Bombay, India), an M.S. in Computer Sciences from Purdue University in 1993, and a Ph.D. degree in Computer Sciences from Purdue University in 1996. His research interests include parallel computation, distributed simulation, and multithreaded programming environments.

FELIPE KNOP, Ph.D., Computer Sciences Department, Purdue University, (August 1996), received a Masters degree in Computer Sciences from Purdue University in 1993 and a Masters degree in Electrical Engineering from University of São Paulo, Brazil, in 1990. He joined IBM, RS/6000 Scalable POWERparallel division, in August 1996. His current research interests include parallel and distributed simulation, and multiprocessor operating systems.

VERNON REGO is a Professor of Computer Sciences at Purdue University. He received his M.Sc.(Hons) in Mathematics from B.I.T.S (Pilani, India), and an M.S. and Ph.D. in Computer Science from Michigan State University (East Lansing) in 1985. He was awarded the 1992 IEEE/Gordon Bell Prize in parallel processing research, and is an Editor of *IEEE Transactions on Computers*. His research interests include parallel simulation, parallel processing, modeling and software engineering.