THE EFFECT OF SYNCHRONIZATION REQUIREMENTS ON THE PERFORMANCE OF DISTRIBUTED SIMULATIONS

Murali S. Shanker and B. Eddy Patuwo
Department of Administrative Sciences
Kent State University, Kent, OH 44242
E-mail: *mshanker@scorpio.kent.edu*

Abstract

Recent experiments have shown that conservative methods can achieve good performance by exploiting the characteristics of the system being simulated. In this paper we focus on the interrelationship between run time and synchronization requirements of a distributed simulation. A metric that considers the effect of lookahead and the physical rate of transmission of messages, and an arrival approximation that models the effect of synchronization requirements on the run time are developed. It is shown that even when good lookahead is exploited in the system, poor run-time performance is achieved if an inefficient mapping of LPs to processors is used.

1 Introduction

With the need to do increasingly complex simulations, distributed discrete-event simulation appears to be a viable way for reducing simulation run time. Two popular approaches for distributed simulation (DS) are the conservative [Bry77, CM79, CM81, Mis86] and the optimistic paradigms [Jef85]. In both approaches, the system is modeled as a collection of logical processes (LPs) that communicate via timestamped messages. The two approaches differ in the manner in which synchronization is provided to ensure the correctness of the simulation.

In this paper we focus on the interrelationship between run time and the synchronization requirements of a conservative DS. Specifically, we consider the Chandy-Misra-Bryant approach for DS [Bry77, CM79, CM81, Mis86]. A metric that considers the effect of lookahead and the physical rate of transmission of messages, and an arrival approximation that models the effect of synchronization requirements on the run time are developed. It is shown that even when good lookahead is exploited in the system, poor run-time performance is achieved if there exists disparity in timestamp increments among different input channels to an LP. This suggests the need to find efficient mappings of LPs to processors if conservative DS are to perform well, even when lookahead properties are exploited.

The rest of the paper is organized as follows. The next section describes our research focus. Section 3 develops an arrival approximation to model the effect of synchronization requirements on run time. Section 4 contains the experimental design, and the results and conclusions are in Sections 5 and 6, respectively.

2 Research Focus

Consider the delays experienced by messages in a DS executed on a parallel-processing computer with the objective of minimizing the run time of the simulation. In addition to the execution time incurred by a message at an LP (we say that a message is executed at an LP to mean that it is really executed on the physical processor to which the LP is assigned), messages may also be delayed for the following reasons:

- Messages may be blocked to ensure that event causality is maintained. Here we say that such messages have not yet satisfied their *precedence* relationships.
- Messages that have satisfied their precedence relationships wait in a *ready* queue until the processor becomes free to execute them.
- Messages that have been executed at a processor may also wait in an *output* queue until the LP's clock value advances to the message's timestamp.

As the execution cost of a message at an LP is normally a fixed cost (assuming homogeneous processors and ignoring communication costs), an optimal assignment of LPs to processors would seek to minimize the input and output delays. The advantage of having good lookahead is that it reduces not only the output waiting time, but also the waiting time for messages *before* they can join the ready queue. It is easy to see that this reduction in waiting time can easily propagate, especially for closed networks and densely connected networks. Here we provide an arrival approximation that estimates the rate at which precedence relationships are satisfied in a DS. Using this approximation it is then possible to find more efficient assignments to improve run time by reducing the waiting time for messages. The next section describes the approximation.

3 An Approximation for Throughput in DS

Figure 1 shows a general representation of the input process at a merge LP for a logical system. Here, LP_i, $i \in \{1, 2, \cdots, n\}$, sends messages to LP_{n+1} along channel $\mathrm{C}_{i,n+1}$. The n channels, $\mathrm{C}_{i,n+1}$, $i = \{1, 2, \cdots, n\}$, merge into a *gate*. There is a single queue, called the *ready* queue, connecting the gate to the merge node LP_{n+1}. Messages to LP_i, $i \in \{1, 2, \cdots, n\}$, are assumed to arrive from outside the system, possibly from other LPs in the logical system.

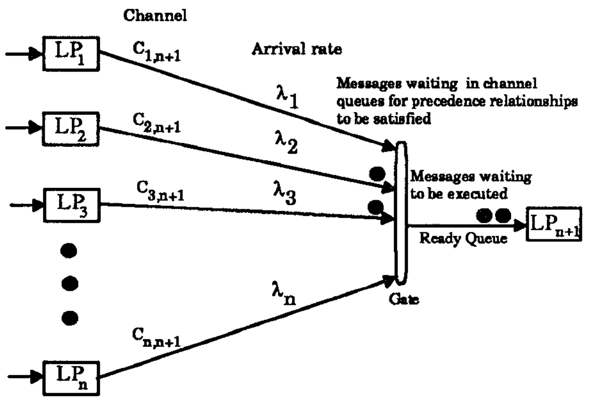

Figure 1: A schematic representation of n arrival processes to a merge LP

The *gate* ensures event causality as follows: a message in $\mathrm{C}_{i,n+1}$ passes through the gate if it is the first message there (the queue discipline is first-in first-out), and if the gate is *open* to that channel. At any point, the gate is open to only one channel: the channel that has either received, or will receive the message with the least timestamp among messages in all input channels. Once the message joins the ready queue, the gate resets to the channel that will receive the message that *should* be processed next. Thus, the gate acts to ensure that messages join the ready queue in non-decreasing timestamp order. The gate is used to differentiate between the two different input waiting periods for a message. Messages before the gate are waiting for precedence relationships to be satisfied, while messages in the ready queue are waiting for the processor to become free. Here we provide tools to reduce the waiting time for messages before the gate. Note that the order in which messages join the ready queue depends only on the timestamp of the messages.

In an optimal implementation, ignoring communication costs, there will be no waiting in queue, before or after the gate. Usually, the waiting time depends not only on the physical arrival rates of messages, but also on the *relative* timestamp of the messages. If messages do not wait in queue before the gate, then the arrival rate of messages into the ready queue is the superposition rate of the n input processes. But this would rarely be the case. In conservative methods there is usually a *penalty* to ensure proper timestamp ordering. Under such circumstances, we are interested in reducing the *penalty* through a proper assignment of LPs to processors.

To develop an approximation for the superposition of arrival processes arising in DS, we consider an equivalent interpretation for the process of Figure 1. Assume messages arrive at the rate λ_i along channel $\mathrm{C}_{i,n+1}$. Let the gate be *open* to $\mathrm{C}_{i,n+1}$ with probability p_i ($\sum_i^n p_i = 1$). If messages are waiting in that channel, the first waiting message passes through the gate into the ready queue. If there are no messages in $\mathrm{C}_{i,n+1}$, the gate remains open to that channel until a message arrives and passes through it. The gate then with probability p_j, independent of the previous state, waits for a message along $C_{j,n+1}$. This interpretation does away with timestamped messages, and the effect of precedence relationships is mimicked by the *probability gate*. In the following discussion we use the two representations interchangeably.

Extensive simulations were conducted to study the behavior of throughput from the gate. If λ_i' is the departure rate of messages from $\mathrm{C}_{i,n+1}$ through the gate, then the following relationship was derived:

$$\begin{aligned} \lambda_i' &= \frac{p_i}{(p_m/\lambda_m)} && (1) \\ \text{where } \frac{p_m}{\lambda_m} &= \mathrm{Maximum}\{p_i/\lambda_i \ \forall i \in \{1, \cdots, n\}\} \end{aligned}$$

Equation (1) allows us to predict the superposition arrival rate of messages to the ready queue. If λ' defines this rate, then

$$\lambda' = \sum_{i=1}^{n} \lambda_i' = \sum_{i=1}^{n} \frac{p_i}{(p_m/\lambda_m)} = \frac{1}{(p_m/\lambda_m)} \qquad (2)$$

Clearly the departure rate λ_i' of messages from $\mathrm{C}_{i,n+1}$ is restricted by the channel that observes p_m/λ_m (i.e., the channel that receives messages with the least timestamp increment per unit time). The actual departure rate depends on the relative magnitude of p_i with respect to p_m/λ_m, and unless $p_i/\lambda_i = p_m/\lambda_m$, $\lambda_i' < \lambda_i$, an unstable situation for $\mathrm{C}_{i,n+1}$ arises. Also, from Equation (2), the limiting factor in throughput is p_m/λ_m. To improve throughput then, we need to minimize p_m/λ_m. An upper bound on the performance is when $p_i/\lambda_i = p_j/\lambda_j \ \forall i, j \in \{1, \cdots, n\}$, in which case $\lambda_i' = \lambda_i$, $\forall i \in \{1, \cdots, n\}$. Two factors can be manipulated to increase this throughput rate: the probability of choosing from a channel, and the arrival rate of messages. The order in which messages are picked is determined by their timestamp. For most simulations, this factor cannot be changed since the mean timestamp change at each LP is a model parameter that is fixed for the simulation. This is both an advantage and a disadvantage. A disadvantage because this parameter can no longer be changed to improve the efficiency of the simulation. An advantage as this fixed value can lead to a better starting assignment. On the other hand, the arrival rate can be changed by a suitable allocation of LPs to processors.

An important use of predicting the effect of precedence relationships is in adaptive schemes for load reallocation. To provide a simple measure for use in such schemes, we define the *advance indicator ratio* AIR as

$$\mathrm{AIR} = \frac{\max AI_i}{\min AI_j} \ \forall i, j \in \{1, \cdots, n\} \qquad (3)$$

Description	Levels
Input streams ($= n$)	2, 4, 5
Arrival rate to LP_i	$1/n$
AIR	1, 2, 4
Service time at LP_i $1 \leq i \leq n$	$0.8n$
Service time at LP_{n+1}	0.3, 0.5, 0.8
Communication time	$0.1n$
Dist. of arrivals and service	Exp.
Dist. of logical clock times	Det., Exp.

Table I: Factor levels for experiment 1

$$\text{Where } AI_j = \text{Expected logical time increment in channel clock } j \text{ per unit physical time}$$

AI_j standardizes the logical clock increments with respect to the arrival rate of messages. In general, larger values of AI_j give better run-time performance. On the other hand, AIR shows the inequality (if > 1) among timestamp increments in the input channels, and large values of AIR are likely to give poor performance of the simulation. AIR provides a simple measure of the effect of precedence relationship on the run time, and can be easily predicted from observed values during the simulation. The next section describes a series of experiments conducted to study the effect of synchronization requirements on run time, and also to validate AIR as a metric useful in identifying inefficient assignments.

4 Experimental Design

Two different experiments were conducted to study the effect of precedence relationships on run time of a DS.

In the first experiment, the logical system representation of Figure 1 was considered for simulation. The objective was to isolate and study the effect of precedence relationships on the run time of a DS of feed-forward networks (FFN). While in most simulations the representation of Figure 1 would be but a small part of the network, for the purposes of the study we limit ourselves to determining the effect of precedence relationship at a merge LP (LP_{n+1}). An implicit assumption is that poor performance at that LP would reflect accordingly in the run time of the simulation. Also, lookahead capabilities were introduced at each LP by taking advantage of the queue discipline and of the non-preemptive service. The factor levels for this experiment are shown in Table I.

Based on the above experiments, another set of simulations were conducted to see if run time could be improved through a suitable reallocation. Here, the arrival rate of messages to LP_i, $i \in \{1, \cdots, n\}$ was changed (a result of a hypothetical reallocation) to minimize the difference in AI values while keeping all other factor levels the same as in the previous experiment. To form a basis for comparison, the net arrival rate was still set to 1.

In the second set of experiments, two closed queueing networks (CQN), the hypercube and ring topologies, were considered for simulation. In each case a network of 16 LPs was simulated on an Intel i860 Hypercube with 4 processors. We assume that each LP had the same service time distribution, and the same homogeneous branching probabilities. As the simulation involved closed queueing networks, the simulation load was varied by adjusting N the number of jobs placed in queue at each LP at the start of the simulation. N varied within the set $\{1, 5\}$. Logical clock times for each LP were chosen to provide different values of AIR. AIR varied within the set $\{1, 5, 10\}$. Again, lookahead capabilities were introduced at each LP, *but* only for experiments for which AIR> 1. This was to determine if good lookahead properties were enough to overcome synchronization penalties. If so, there should be no significant difference in run-time performance for different values of AIR. For all experiments, the simulations were replicated until statistically significant results were obtained at the 90% level. The next section describes the results.

5 Results

The results of the experiments are in Figures 2 through 5. Here, only a representative set of results is presented.

5.1 FFN Experiments

Figure 2 shows the effect of precedence relationship on the arrival rate of messages to the ready queue at LP_{n+1}. Even when good lookahead is exploited, the arrival rate to LP_{n+1} decreases significantly as AIR increases. As the arrival rate to each LP_i, $i \leq n$, is the same, the performance of the simulation for varying values of AIR is a result of the difference in timestamp increments in the n input channels. The predicted arrival rate (based on Equation (2)) is also shown (with legend n=?(p)). Clearly, large values of AIR lead to a deterioration in performance, and that Equation (2) serves as a good approximation. Note that as the arrival rate of messages to the ready queue is less than 1 (the maximum expected rate) for AIR > 1, the observed utilization at LP_{n+1} will be less than the maximum expected utilization.

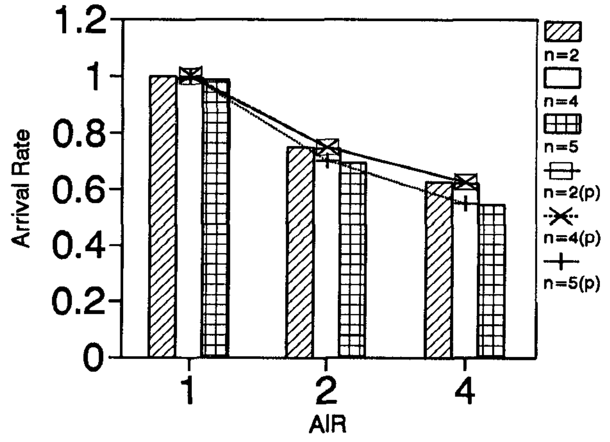

Figure 2: FFN — Arrival rate of messages to the ready queue

Clearly, disparity in AI values lead to poor performance of the simulation. As mentioned earlier, one approach in such situations is to change the arrival rate of messages by reallocating LPs. Now, the arrival rate to LP$_i$, $i \in \{1, \cdots, n\}$, was changed so as to minimize the difference in AI values. The net arrival rate of messages to the system was still set to 1, with the minimum expected time between arrival to LP$_i$ being $0.8n$ (to prevent an increasing queue there). The results are shown in Figure 3. Clearly, there is a significant reduction in run time.

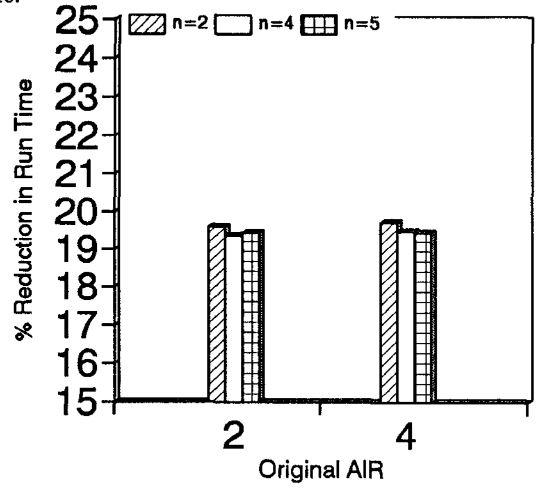

Figure 3: FFN — % reduction in run time

The above experiments support our approximation, and show that run time performance can be improved by considering a suitable allocation of LPs to processors.

A similar result is seen for CQNs. Figures 4 and 5 show the results. In both cases, values of AIR> 1 (even with good lookahead properties) lead to a deterioration in performance compared to when AIR= 1. The percent difference between actual and predicted values (using Equation (2)) are also shown (with legend N=?(%diff)). Clearly, Equation (2) is a good approximation for throughput.

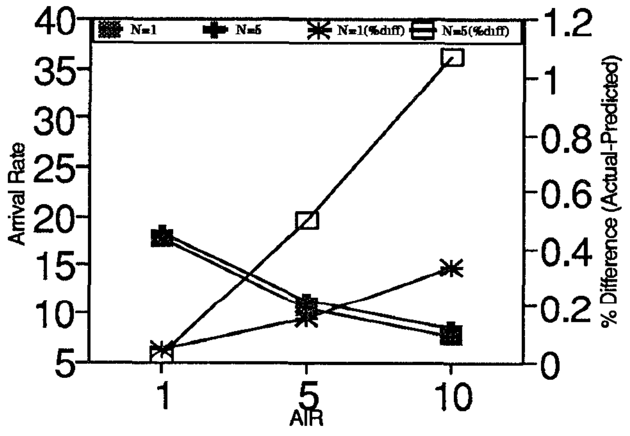

Figure 4: Hypercube Topology — Arrival rate of messages to ready queue

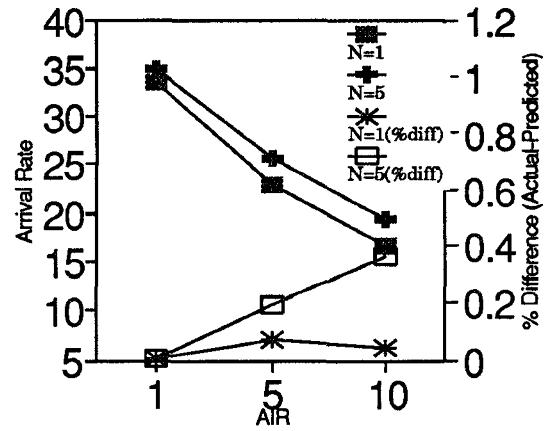

Figure 5: Ring Topology — Arrival rate of messages to ready queue

6 Conclusion

The above experiments show that precedence relationships can have a significant effect on the performance of a distributed simulation, even when good lookahead is exploited in the system. The experiments also suggest that it is necessary to consider the physical characteristics of the computer system as well as the characteristics of the simulation model while making an assignment of LPs to processors if good run-time performance of the simulation is desired. To aid in identifying inefficient mappings of LPs to processors, an arrival approximation and a measure to model the effect of precedence relationships are presented. Allocation strategies can then use the above measures to find efficient assignments of LPs to processors.

References

[Bry77] R. E. Bryant. Simulation of packet communication architecture computer systems. Technical Report MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[CM79] K. M. Chandy and J. Misra. Distributed simulation: a case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452, 1979.

[CM81] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24:198–206, 1981.

[Jef85] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[Mis86] J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1):39–65, 1986.