

The Role of Knowledge in Distributed Simulation

Wayne M. Loucks

Bruno R. Preiss

Department of Electrical and Computer Engineering
University of Waterloo,
Waterloo, Ontario, Canada, N2L 3G1

ABSTRACT

This paper examines the suite of closed, stochastic queueing networks proposed by Nicol in order to determine how much impact additional knowledge has on the computation load and the communication load for the multiprocessor running the distributed simulation. Four different networks have been studied at three different levels of traffic intensity. For each network-load pair, four simulation models were constructed, each based on increasing use of user knowledge at a queueing node: no knowledge; service discipline knowledge; queueing discipline knowledge; and routing knowledge. In each case the number of logical process activations and the number of messages used by the distributed simulation was determined and compared to the number of activations needed in a traditional, event-list-driven simulation.

In the area of computation needed, distributed simulation using a conservative synchronization algorithm permits between one half and two thirds of the computation to be eliminated when user knowledge is added to the simulation. The results presented also indicate that the differences caused by the additional knowledge are about twice as dramatic in their impact on communication load as they are on the computation load. The results from using an optimistic synchronization algorithm indicate savings of a similar magnitude, except there is a better reduction in computation than in communication. In all but one case the reduction caused by the additional knowledge is less than an order of magnitude.

1. INTRODUCTION

The work described in this paper is part of an ongoing effort to develop a message-passing multiprocessor capable of efficient support of distributed simulation. In fact, the project goal is slightly more focused in that it concentrates on the simulation of logic circuits. As part of this project it has been necessary to establish some measures of the load which may be placed on such a multiprocessor.

In order to evaluate the needs of a distributed simulation engine, we have chosen to characterize a distributed simulation load by the number of messages passed and by the number of times that a model is activated (or called). The number of messages passed represents a clear indication of the complexity and speed of the simulation engine's communication structure while the number of model calls very roughly estimates the computation load of the simulation.

One of the classic problems in the design of a new system is the characterization of the anticipated workload. To some extent much of the work in distributed simulation suffers from this problem, whether it be pucks on a frictionless surface (Hontalas *et. al.* 1989), ants in search of food (Ebling *et. al.* 1989), armies interacting (Wieland *et. al.* 1989), or graphs representing health care systems (Baezner *et. al.* 1989). In the work

described below we have analyzed the suite of closed queueing network models proposed by Nicol at the 1988 Winter Simulation Conference (Nicol 1988).

Using the results from simulations of these networks (described in Section 2) we examine how the number of messages and model calls vary with respect to: synchronization method (Chandy-Misra vs. virtual time — see Section 2.3); network topology (see Section 2.4); network load (see Section 2.5); and user knowledge (see Sections 1.1 and 2.6).

It is in the area of knowledge that this paper has the greatest concentration of effort. The question addressed is: can the user, by incorporating additional knowledge significantly alter the computation and communication load for a given distributed simulation?

Previous work which has examined the impact of various conservative and/or optimistic synchronization techniques (Presley *et. al.* 1989, Ebling *et. al.* 1989, Fujimoto 1988, Baezner *et. al.* 1989, Baezner *et. al.* 1989, Nicol 1988), compared the impact of various changes on the final completion time on a given (multiple) computer. These results obviously provide a very clear indication of the speed-up possible, however it does introduce another variable — the effectiveness of a given computer structure. In this analysis we calculate the number of model calls and the number of messages based on a "simulated" multiprocessor environment (see Section 2.3). While this prevents us from obtaining meaningful measures of speed-up, it permits us to establish a load measure which is more independent of the multiprocessor's architecture.

1.1. User Knowledge

A person, the user in this case, wishing to prepare a problem for simulation, can impose various constraints on the system being simulated. For example, in the case of logic simulation the user "knows" that when a gate's inputs are specified at a given time the output of the gate (and hence the Logical Process (LP) representing the gate), *must* have a given output some fixed delay, say Δ , later. This user "knowledge" may be incorporated into the system in hopes of improving the performance of the simulation. In this paper we examine the impact and the potential for adding user knowledge to a class of closed queueing network models.

2. EXPERIMENTAL METHOD

2.1. Overview

In this study, we examine certain characteristics of the performance of distributed discrete event simulations of closed, stochastic queueing networks. We have studied the performance of simulations of the same suite of four stochastic queueing networks as in (Nicol 1988). In each case, the system simulated was a static network of nodes. Each node consisted of an

n-input, n-output queue and a single server. Various network sizes and topologies were considered.

The simulation specifications were coded in the Yaddes language (Preiss 1989). Because these networks contain many instances of the same type of node, only one model specification is required. We investigated four different implementations of the node model specification — each with a varying degree of user knowledge (See Section 2.6).

Our implementation of Yaddes supports four different time synchronization methods (described below). Although we have performed simulations using all four synchronization methods this paper examines only the conservative and optimistic schemes.

Finally, three different network loads were simulated. The network load was adjusted by varying the number of customers in the closed queueing system.

In all cases, the following data on the queueing systems being simulated were collected: (1) mean and standard deviation of service times, (2) mean and standard deviation of the time a customer spends in the queue, (3) mean and standard deviation of queue length, (4) mean and standard deviation of the number of the output arc along which a customer leaves a node. These data were used to verify that all of the simulations of a given network topology and load are equivalent. In all cases, *without exception*, the simulation statistics were identical.

2.2. Simulation Performance Metrics

We are principally concerned with the computation and communication requirements of distributed discrete event simulation. Although we have an implementation of Yaddes that runs on a network of Apollo DN3000 workstations connected by a 12 Mbit token ring (Preiss 1989) as well as a version which runs on a network of Transputers, all the simulations presented in this paper were run on a single processor. The uniprocessor version of Yaddes supports the collection of certain simulation performance data and exhibits repeatable behaviour.

In this paper, we focus on two performance metrics — the number of model calls and the number of messages. The number of model calls is a measure of the computation load required to perform the simulation. A model call occurs for each event combination. An event combination is a set of events occurring at the same simulation time in one LP. Note that a model call occurs for each occurrence of simultaneous (in simulation time) events, not for each individual event.

The number of messages is a measure of the communications load required to perform the simulation. In this paper, when we refer to a message, we mean a message from one LP to a different LP. (In the results reported below any self-loop messages have been excluded from the total number of messages.)

2.3. Synchronization Methods

Our implementation of Yaddes supports four different synchronization methods. Yaddes is discussed in detail in (Preiss *et al.* 1988, Preiss 1989) and (Preiss and Loucks 1989).

The intent of this work is to examine to what extent optimistic and conservative distributed simulation can take advantage of additional user knowledge. The experiments discussed below were based on simulations using the Chandy-Misra-Bryant (CM) and the virtual time (VT) synchronization methods. All simulations were also performed using the traditional event list technique (EL) for comparison purposes.

The CM method is an implementation of the Chandy-Misra-Bryant (Misra 1986), conservative synchronization method. In our implementation of Yaddes, no deadlock detection or recovery is provided. It is up to the simulation programmer to send the necessary null messages. N.B., in Yaddes, a null message is thus equivalent to all other messages. Null messages simply carry no information other than the implied non-arrival of a non-null message.

The VT method is an implementation of a virtual-time-based, optimistic synchronization method. In the current version of Yaddes, we use aggressive cancellation (Jefferson 1985), we do not support preemption, and we do not provide any flow control.

These results were obtained using a “simulated” multiprocessor environment as discussed in the references. As a result absolute execution times cannot be discussed, instead changes in the computation and communication loads for the various simulations are discussed (see Section 3).

In the simulated multiprocessing environment there is the implicit assumption that all model calls, that is all activations of an LP require the same computation time. In this particular problem this is a reasonable assumption as all models (for a given level of knowledge) are the same and each model has only one event type which can be acted upon. In addition to the equal time for all model calls there is also no allowance made for the time to transfer the message. Thus messages appear at the destination input queue as soon as they are sent. This particular assumption would clearly lead to inaccurate results if the “execution” times were used in the analysis. Since in this work we only consider the number of model calls and the number of messages transferred, this assumption should not significantly impact the conclusions drawn in Section 5.

One final operational characteristic of Yaddes which impacts these results is the technique used in VT to increment the global virtual time (GVT). Although in previous work a “fairer” scheme was used, the models described here had such a large state and there were so many instantiations of these models that a more efficient (but less true to the distributed model) scheme had to be used. In this project, the Yaddes kernel for VT has been modified to re-evaluate GVT following one activation of each model. Although this would clearly be impossible in a truly distributed system, in this case we are not concerned with the efficiency of the state saving mechanism but rather only the number of model calls which must be executed and re-executed.

2.4. Network Topologies

A number of recent studies on the performance of distributed discrete event simulation have used simulations of closed, stochastic queueing networks as their benchmarks (Nicol 1988, Lubachevsky 1989a, Fujimoto 1988, Fujimoto 1987, Fujimoto 1989). In this study, we have chosen to use the same suite of network topologies as used in (Nicol 1988). The network topologies simulated are listed below.

- A multi-stage network consisting of six stages each with 64 nodes for a total of 384 nodes (two input and two output links per node).
- A 256-node bi-directional ring (two input and two output links per node).
- A 256-node two-dimensional nearest-neighbour mesh in which each node is connected to four adjacent nodes (four input and four output links per node).
- An eight-dimensional Boolean cube containing 256 nodes (eight input and eight output links per node).

Each node of the network consists of an n-input queue and a single server with n-outputs. Each node is simulated by a distinct LP.

Customers arriving at a node are placed into the queue. In the event of simultaneous arrivals of customers, the customers are placed into the queue in an arbitrary order. (This arbitrariness is of no consequence as customers are indistinguishable.) Customers are serviced by the server strictly on a first-come, first-served (FCFS) basis.

Service times are sampled from a biased exponential service time distribution. I.e., $T_{service} = T_{minimum} - (T_{mean} - T_{minimum}) \ln(rand())$. In our simulations, we used a mean service time of 10 and a minimum service time of 2.

Upon completion of service, the customer departs from the node on a randomly selected output arc. The routing of a message is random and not determined by the arriving message. In these networks routing is determined

by the node which receives a message.

2.5. Network Load and Initialization

Three different values of network load were simulated. We define the load as the average number of customers in a node (either in its queue or being serviced). Loads of 1, 4, and 8 were simulated. The networks were initialized by inserting into every queue at simulation time zero a number of customers equal to the desired load. In most cases, the simulations were performed for 10,000 time units (i.e., 1000 times the average service time).

2.6. Levels of User Knowledge in the Simulation

In this study, three types of user knowledge which may be used to (hopefully) reduce the cost of the simulation, are considered. These techniques are summarized in Table I. These techniques are adapted from those outlined in (Nicol 1988). The knowledge levels result from considering the nature of the problem and the types of messages needed in the problem. For convenience we have defined "customer bearing messages" as any message which transfers a customer or token to another process. This excludes all null messages and any messages sent to the local process (i.e. self messages).

Table I
Knowledge Level Usage

name	time of transmission of output event message	NPS	FCFS	RIFM
Departure	At the instant that the customer departs from the node.	no	no	no
Service	At the instant service begins (and thus knowledge about the service time and service discipline is used in the LP).	yes	no	no
Queue	At the instant that the customer enters the queue. (and thus knowledge of the queueing discipline is used)	yes	yes	no
Arrival	At the instant that the customer enters the queue. (Null message times are obtained by presampling service times.) In this scheme all of the attributes of a given customer token, except its arrival time, are precalculated prior to arrival.	yes	yes	yes

The three levels of user knowledge may be incorporated into the LP which defines a node in this problem. These three levels of user knowledge lead to four possible implementations of the LP. The node implementations are named according to the time of the transmission of the output message (as shown in Table I). The levels of knowledge are listed below.

1. *Non-Preemptive Service (NPS)* — In this problem once service has started, the customer in service cannot be prevented from completing service at a precalculated (random) time.
2. *First Come First Serve (FCFS)* — In this problem once a customer token has entered a queue to receive service, the only customers which can impact its service completion time are those in front of it in the queue.

3. *Routing Information Free Messages (RIFM)* — Since the messages do not contain any information as to the route to be followed by the customer, it is possible to predict in advance all future service times and all future routes to be taken out of a given node. The only reason to wait for the incoming customer message is to determine when service starts.

In the simplest case (*Departure*), a message containing a customer is not transmitted until the customer has completed service at an LP. Thus there is no user added knowledge in the LP. Specifically, if a customer is to complete service in some LP at time T , a message is sent from that LP to its successor at time $T-\epsilon$ where ϵ is the smallest resolvable non-zero time unit. This message indicates to the successor that a customer will arrive at time T . This use of ϵ is required by Yaddes in order to avoid deadlock when conservative synchronization methods are used. Null messages (i.e., non-customer-bearing messages) with the same time stamp, are sent to all other successor LPs that did not receive a customer.

In the *Service* scenario, the LP uses the knowledge that there is a non-preemptive service discipline. Thus, a customer message with time stamp $T+T_{service}$, where $T_{service}$ is the (random sample) service time for the customer, is sent from an LP at time T when the customer departs from the queue (i.e., begins to be serviced). Null messages (i.e., non-customer-bearing messages) with the same time stamp, are sent to all other successor LPs that did not receive a customer.

The *Queue* scenario uses not only the knowledge of non-preemptive service but also the knowledge that the queueing discipline is FCFS. Thus completion times can be sent to the appropriate destination as soon as the customer bearing message is inserted into the queue. Thus a customer message with time stamp $T+T_{service}+T_{queueing\ delay}$, where $T_{service}$ is as above and $T_{queueing\ delay}$ is the queueing delay experienced by the customer, is sent from an LP at time T when the customer arrives at the LP. Null messages (i.e., non-customer-bearing messages) with the same time stamp, are sent to all other successor LPs that did not receive a customer.

The fourth scenario, called *Arrival*, uses all of the knowledge levels discussed above. In effect the LP predicts the earliest possible time for next message on each outgoing channel and forwards this time to the next LPs. This scheme behaves similarly to Queue with respect to customers. However, the null messages carry predictive time stamps that are based on presampling of the random service time and routing distributions. (This is implemented using a *future list* as described in (Nicol 1988)). This presampling is possible because of the FCFS queueing discipline and the fact that the customers do not carry their own routing information. Null messages (i.e., non-customer-bearing messages) are transmitted bearing time stamps of the form $T'-\epsilon$, where T' is the earliest possible time that a customer can arrive at the successor (based on current knowledge).

When implementing the Arrival technique, there are two cases to consider: First, a message (customer or null) received when the server is idle requires the updating of the predictions on all outgoing arcs. Second, a customer arriving when the server is busy only requires the updating of the prediction on the outgoing arc on which the customer departs. (A null message received when the server is busy has no effect.)

Since the Arrival technique only affects the manner by which null messages are handled it can only impact the performance of the distributed simulation when CM synchronization is used. Unlike the *Queue* technique which sends null messages out all links except the one which just received a customer bearing message, the *arrival* technique sends null messages out *all* links. This aggressive null message sending will only lead to an overall reduction in messages in those cases in which the extra null messages (one in n where n is the number of outgoing links from the node) are compensated by more parallelism.

Although it is demonstrated in Section 3 that increasing knowledge can lead to improved simulation performance, it should be remembered that each of these levels of knowledge represent a constraint of the system that can be modelled.

2.7. Typical Experiment

For a given network topology and load (e.g., Hypercube with load 4) and a given knowledge level (e.g., Departure) and a given synchronization method (e.g., CM), the simulation was run for a duration of 10,000 time units with an average service time of 10 time units. Table II shows the number of messages and the number of model calls obtained for the Hypercube with load 4 using the various knowledge levels for both synchronization methods.

Table II
Typical Simulation Results
(Hypercube with a load of 4)

measure	model calls (in millions)		messages (in millions)		
	CM	VT ³	total		customer ²
synchronization	CM	VT ³	CM	VT ³	
Knowledge Level					
Departure ¹	2.56	2.85	20.9	3.10	0.213
Service ¹	2.56	1.27	18.8	1.41	0.213
Queue	2.19	0.509	4.447	0.832	0.214
Arrival	1.89	0.509	3.27	0.832	0.214

1. Although, for detailed Yaddes implementation specific considerations, it was necessary to send some "self-loop" messages in these cases, the self-loop messages have been omitted from these figures.

2. This value is derived from the number of customer bearing events processed during the EL execution for the given knowledge level. The slight variation is as a result of various termination conditions which result from the slightly different code used at each knowledge level.

3. The VT data were obtained by running the simulation for 1000 time units (instead of 10,000) and then the number of model calls and the number of messages were multiplied by 10. This was done as the computer which we were using to run the Yaddes system did not have enough main memory to contain all of checkpointed state information for these large networks. The resultant disk thrashing seriously degraded the throughput of the simulations. Although this "amplification" of time leads to statistically poorer simulation results, our results indicate that the simulation performance data were not very sensitive to this effect.

3. OBSERVATIONS

The raw results obtained are summarized in Figures 1 and 2. In these figures, the number of messages sent is plotted against the number of model calls. These figures present the same set of data points but use different legends. Although these figures are crowded they do serve to show: the size of the problems being examined (millions of messages and millions of model calls); the wide variation in the communication and computation loads; and the impact of knowledge in a gross sense (i.e. service and departure techniques are the only ones which produce a very high load).

Although the data points are crowded in the region below 1,500,000 model calls, it is readily apparent that there is a correlation between the number of model calls and the number of messages. This is expected since for each model call there is a fixed, maximum number of messages that can be sent.

A special comment is needed regarding the points which seem to form a vertical line at slightly above 2,500,000 model calls. (The numeric value is 2,560,512 model calls). At this point (for all network topologies except Multistage) the simulation (using CM synchronization) has degenerated into a time-driven mode of operation. I.e., each LP is processing at least one message (or null message) at each and every point in simulation time. Since there are 256 models activated in each of 10,000 time steps, we expect to see 2,560,000 model calls. Of these only from 1.4 % to 18 % of the model calls are "useful" in that they process customer bearing messages and

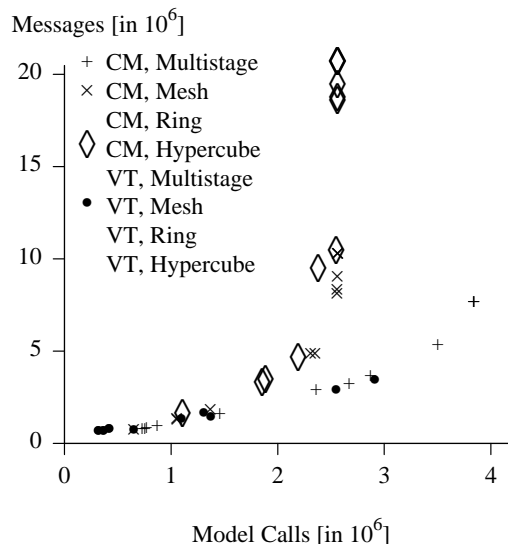


Figure 1. Number of Messages vs. Number of Model Calls.
(Network Topology and Synchronization Identified.)

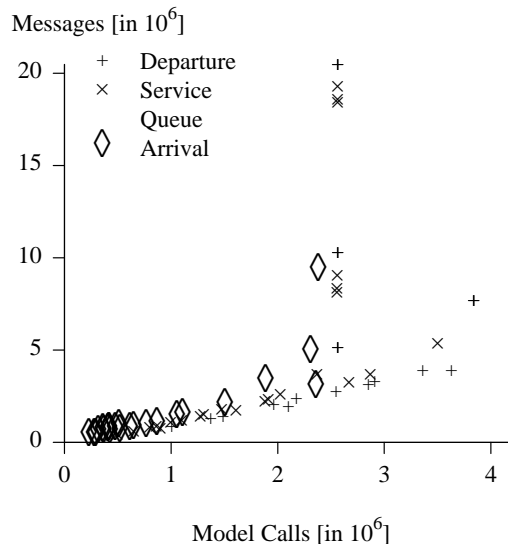


Figure 2. Number of Messages vs. Number of Model Calls.
(Knowledge Level Identified.)

are needed in the sequential, event-list execution of the same problem. The additional 512 model calls trigger the initial and final actions performed by the nodes before the simulation begins and after the simulation ends (respectively).

The beginnings of a second vertical line can be seen at above 3,840,000 model calls. This line corresponds to the degeneration of the simulation of the Multistage network into the time-driven mode of execution.

At the point where the simulation has degenerated into time-driven execution, the number of messages indicates the effect of the incorporation of knowledge. Figure 2 shows that the larger entries on the vertical line are a result of either the Departure or Service scenario (recall that Departure corresponds to no knowledge while Service uses knowledge about service

time only — non-preemptive service). Queue and Arrival knowledge only appear with smaller numbers of messages. This data support the intuition that the more knowledge that is incorporated into the model, the fewer the number of messages that are needed.

Figure 3 provides an expanded view of the lower left portion of Figure 2. It shows the preferred modes of operation in that the simulation has not degenerated to a time-driven situation. It is readily apparent from Figure 3 that, as would be expected, the more knowledgeable implementations (Queue and Arrival) exert the smallest computation and communication loads (the results for these levels are clustered at the lower left).

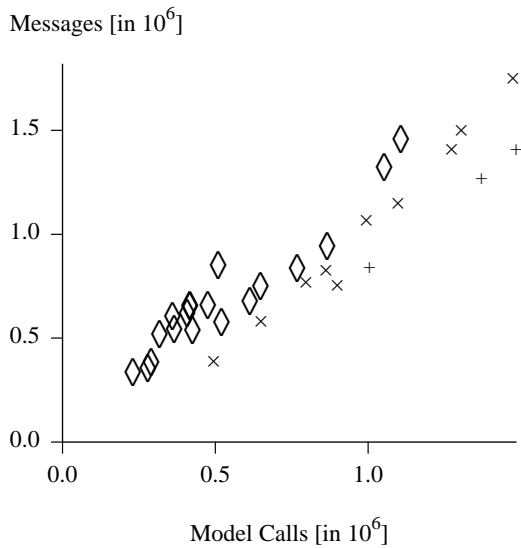


Figure 3. Number of Messages vs. Number of Model Calls.
(Knowledge Level Identified.)
(Bottom left corner of Figure 2 and the same Legend.)

4. IMPACT OF KNOWLEDGE ON COMMUNICATION AND COMPUTATION

In the previous section the raw data indicate that there is a wide variation in the computation and communication load in the systems examined. In this section we examine the direct impact of knowledge on computation and communication load in the distributed simulation. Figures 4 and 5 show the fractional improvement as a result of increasing the knowledge.