Proceedings of the 1991 Winter Simulation Conference
Barry L. Nelson, W. David Kelton, Gordon M. Clark (eds.)

DEADLOCK DETECTION AND RECOVERY BASED DISTRIBUTED SIMULATION: A PERFORMANCE STUDY

Devendra Kumar
Saad Harous

Department of Computer Engineering and Science
Case Western Reserve University
Cleveland, Ohio 44106

ABSTRACT

In this paper we present the results of an experimental performance study of distributed simulation of closed queuing networks using a minor variation of the deadlock detection and recovery based algorithm of Chandy and Misra. One major part of this study is to determine the effect of overhead on distributed simulation. Moreover, in these experiments we measure the values of certain refined notions of "ideal speedup ratio" which were defined in our earlier works. These ratios are more refined than just the number of processors in the sense that they capture the potentially achievable speedups of distributed simulation more closely.

1 INTRODUCTION

Several schemes for distributed simulation have been proposed in the literature, e.g., Chandy and Misra (1979, 1981), Jefferson and Sowizral (1982), Peacock, Wong, and Manning (1979). The schemes utilize overhead messages to handle the potential deadlock situations that may arise during the simulation.

Unfortunately, only a few performance studies of these schemes are available, e.g., Fujimoto (1988a, 1988b, 1989), Reed (1983), Reed, Malony, and McCredie (1988), Seethalakshmi (1979). These studies have provided useful data, and have shown some positive and some negative performance results for a few combinations of the distributed simulation scheme, the system to be simulated, and the available distributed system on which the simulation is to be carried out. Unfortunately, several obvious questions have remained unanswered. For example, what is the relationship between the amount of overhead and the performance of a distributed simulation scheme. Understanding this relationship is important since this can provide useful information as to whether there is much hope in trying to improve the performance by trying to find variations of a scheme that aim at reducing the amount of overhead. For example, suppose the distributed simulation of a particular system using the scheme Chandy and Misra (1979) has low performance. Would it be useful to define and study variations of the scheme that would tend to reduce the number of NULL messages? A related question is how much speedup one can expect in an ideal variation of a distributed simulation scheme that would somehow totally eliminate the negative effect of overhead messages. Can we expect a reasonable performance in such a simulation? Also, what kind of ideal speedup can be expected if one is considering possibilities across all the well known schemes Chandy and Misra (1979, 1981), Jefferson and Sowizral (1982), Peacock, Wong, and Manning (1979) and their simple variations.

The known performance studies have not addressed these issues adequately. The usual studies have provided data on the performance of distributed simulation for individual cases, but the above issues have been largely ignored. In some cases where performance of the distributed simulation turns out to be poor, some researchers have suggested that the poor performance is *due to* large overhead Chandy and Misra (1981), Reed, Malony, and McCredie (1988), Seethalakshmi (1979). Such a suggestion would normally mean that (i) indeed there is a large amount of overhead, and (ii) if somehow the overhead could be reduced significantly then the performance would be improved significantly. Claiming part (ii) of this statement for a specific case requires further study and such a study is reported here.

Note that it is not quite obvious as to how to study the relationship between performance and overhead. One simple-minded approach would be to define and study different variations of a given scheme which lead to different number of overhead messages, thereby providing some data regarding relationship between overhead and performance. Unfortunately such an approach would have several problems:

1. In coming up with the different variations, one has to ensure that deadlocks are handled ade-

quately, otherwise the scheme would simply be incorrect. This restricts the number of different variations one can come up with.

2. It is usually not feasible to vary the number of overhead messages in a reasonable range so as to enable one to determine the relationship between this number and the performance. In particular, the number of overhead messages may remain large, in which case it is difficult to judge what the performance would be like if this number were somehow reduced to a small value. Note that varying the number of overhead messages in a controlled way, i.e., as an independent variable is nearly impossible.

3. As the different variations are being considered it is quite possible that other activities in the simulator that are not related to overhead are also changing, e.g., the algorithmic rules that determine when an event message should be computed. This makes it harder to judge whether a change in performance is simply due to a change in the number of overhead messages or whether the other factors are also influencing the result.

We discuss here a simple and new approach towards determining the relationship between amount of overhead and performance. In this approach: (i) we *simulate* the distributed simulator instead of directly implementing it on a distributed system and directly measuring its performance, and (ii) in this approach we vary the amount of communication delay and the computation time for each overhead message including the case when these values are zero. Note that this approach is quite different from trying to vary the *number* of overhead messages.

Regarding the issue of "ideal speedup" in distributed simulation, the usual measure for ideal speedup used in the literature is N, which is the number of processors in the simulator. This measure is too conservative in many cases, and as such does not provide much information as to how much speedup one can hope to achieve by variations in the simulation scheme or faster communication of overhead messages, etc. (for any given processing speeds for computing the event messages). In the experiments reported here we determine the values of two new measures of ideal speedup that would provide more realistic bounds on what can be expected under an ideal distributed simulation. These measures were defined earlier in Kumar and Harous (1990, 1991), and the definitions are repeated here to keep this paper self-contained.

The overall ideas of this paper have been pointed out earlier in Kumar and Harous (1990, 1991) where we studied distributed simulation of open cyclic queuing networks using the NULL messages based scheme Chandy and Misra (1979). In this paper we extend that work by further studies of closed queuing networks using a variation of the distributed simulation scheme given in Chandy and Misra (1981).

In section 2, we define the systems under study. In section 3, we give a brief description of the distributed simulation scheme under study. Definition of the different speedup ratios and description of our implementation are given in sections 4 and 5. The experimental results are discussed in section 6. Finally, concluding remarks are given in section 7.

2 SYSTEMS UNDER STUDY

Borrowing terminology from Chandy and Misra (1979), in the following a system to be simulated is called the *physical system*. The physical system consists of a network of *physical processes* (or *pps* for short). Each physical system is simulated by a distributed simulator called the *logical system*. The logical system is a collection of *logical processes* (or *lps* for short), each one simulating a corresponding *pp*.

In this paper, we use the term *queuing networks* to refer to networks of *pps* from the five classes - *delay*, *fork*, *merge*, *sink*, and *source*. Examples of the first three classes of *pps* are *pp*1 (delay), *pp*2 (fork), and *pp*4 (merge) in Figure 2. The other two classes of *pps* are not used in the queuing networks discussed in this paper. A detailed definition of these five classes of *pps* is given in Kumar and Harous (1991). Since these *pps* are fairly common in the literature, we skip their definitions here. Queuing networks are useful in performance modeling of various kinds of service systems such as computer systems, computer-communication networks, telecommunication systems, and manufacturing systems. Here we study the queuing networks shown in Figures 1, 2 and 3.

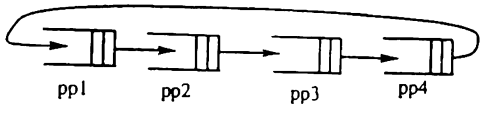

Figure 1: Queuing Network 1.

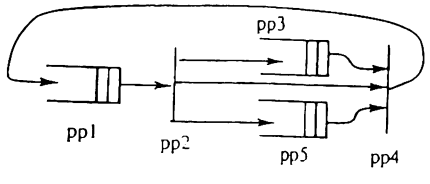

Figure 2: Queuing Network 2.

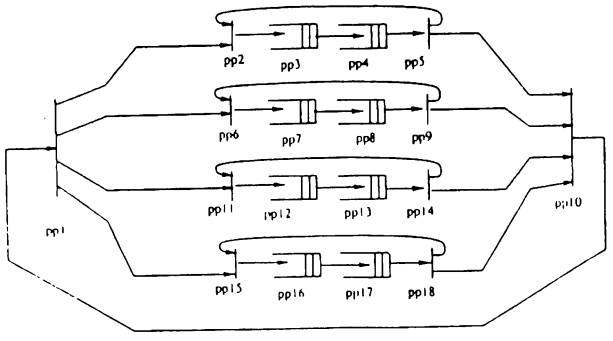

Figure 3: Queuing Network 3.

3 THE DISTRIBUTED SIMULATION SCHEME UNDER STUDY

In this paper we study the scheme based on deadlock detection and recovery Chandy and Misra (1981). We made some minor modifications to this scheme in order to improve the degree of concurrency.

3.1 The Original Scheme Of Chandy and Misra (1981)

The distributed simulation algorithm described in Chandy and Misra (1981) mainly repeats the following two phases: (i) simulate the physical system until a deadlock occurs, (ii) detect the deadlock and recover from it.

The scheme Chandy and Misra (1981) is based on the synchronous communication of Hoare's CSP. This can cause a minor performance degradation when implemented on an asynchronous system. It detects deadlocks by using a variation of the termination detection algorithm of Dijkstra and Scholten (1980).

3.2 Modifications

In order to achieve higher level of concurrency we made the following modifications: (i) we use a simple termination detection algorithm using a central process (CP), (ii) message communication in the logical system is assumed to be asynchronous, (iii) we assume infinite input buffers at the input port of any *lp*, and (iv) an *lp* can receive inputs any time, except when it is able to send out more outputs.

3.3 Our Termination Detection And Recovery Algorithm

We give here an overview of our termination detection and recovery algorithm. In order to keep the discussion simple, we assume that the physical system to be simulated is a queuing network consisting of the five classes of *pps* mentioned in section 2.

Let Z be the time up to (including) which the physical system is to be simulated. Unlike Chandy and Misra (1979), we allow for the possibility that several event messages may be sent at the same time on a given line in the physical system. When an *lp* has determined that it has sent out all the event messages up to time Z on an output line, it sends out exactly one special message (Z+1,NULL) on the line in order to provide this information to the receiver *lp*. No other NULL messages are sent here such as the ones in Chandy and Misra (1979). The *lps* receive inputs whenever they have none of the above event or NULL messages to send.

The set of *lps* in the logical system is statically divided into two groups: system X and system Y. System Y is any arbitrary subset of *lps* that are guaranteed not to be on a deadlock cycle. For example, source and sink *lps*, or other *lps* that are simply not on any cycle in the given network topology, may be placed in system Y. The *lps* in system Y only send event messages and the above NULL message (Z+1,NULL) on any given output line. These *lps* do not take part in termination detection. From the point of view of performance, the set Y should be chosen to be as large as possible; though from correctness point of view this set may be chosen to be any smaller subset, even the empty set. An *lp* in system Y terminates itself once it has sent or received a NULL message on each adjacent line.

The *lps* in system X do take part in the termination detection, in addition to the above (Z+1,NULL) messages on their output lines.

We say that a line (i,j) is in system X if both *lps* i and j are in system X. Otherwise it is said to be in system Y.

How is termination detected for *lps* in system X? When an *lp* i in system X has no further output tuples to send (event or NULL messages), and has no input message available for reception, it computes its *conditional-next-time*, defined to be the t-value of the next output message with minimum t-value from this *lp* (event or NULL message), assuming that no further inputs are received by the corresponding *pp*. If already the NULL messages have been sent on each output line, then *conditional-next-time* is (Z+2) indicating that no messages would be sent.

Then the *lp* compares its *conditional-next-time* with the *lineclocks* of all its input lines which are in system Y. (The *lineclock* of a line is the largest known lower bound on the time stamp of the next tuple to arrive along the line. Normally it is the time stamp of the last tuple received on the line, but sometimes the termination detection and recovery algorithm may advance it further, as dis-

cussed later.) If there is such a line satisfying *lineclock* < *conditional-next-time* ≤Z+1 then the *lp* considers itself to be *active* and it is assumed that the deadlock has not occurred yet. Note that if this boolean condition is true then a future message may arrive along this particular input line that may cancel the output message corresponding to the *conditional-next-time* computed above. On the other hand if there is no line satisfying the above condition, then the *lp* considers itself to be *idle*. In other words, a deadlock may have occurred and the *lp* needs to send relevant information to the process CP (central process) to determine if indeed this is so. To this end, it sends a message *idle*(t,A) to the process CP (central process) where t is the *conditional-next-time*, and A is an array that tells the number of event plus NULL messages received or sent on its each adjacent line in system X. These message counts only include the messages sent or received *after* sending the previous *idle* message (or after system initialization). As a simple performance improvement, we require that this message is sent out only if the parameter t has changed from the previous such message sent by this *lp*, or the array A contains at least one non-zero element (or this message is being sent the first time by the *lp*).

The CP, on receiving an *idle*(...) message, determines whether all the computation in system X has come to a point of termination, i.e., none of these *lps* can send out any (event or NULL) messages and there are no such messages in transit on the lines in system X. To detect this, the CP keeps information as to which are the active *lps* in system X. Initially all *lps* in system X are marked *active*. Also, the CP maintains an array B that contains a message count for each line in system X — the message count is the number of messages sent minus the number of messages received on the line, as known to the CP.

When an idle(...) message is received, the CP marks the sender *lp* as *idle* and updates the array B. At this point if all *lps* are marked idle and all entries of B are zero, then the CP concludes that termination has occurred. This idea of termination detection is based on the termination detection algorithm given in Kumar (1987).

Once the CP has determined the termination of computation in system X, it computes the minimum of the last *conditional-next-time* values received from each *lp* in system X (say T) and sends a *fire*(T) message to all those *lps* in system X whose *conditional-next-time* value equals T. These *lps* which are sent *fire*(T) messages are now marked *active*. If T=Z+2 then CP terminates.

On receiving a *fire*(T) message, if T≤Z+1 then *lp* *i* updates its input *lineclocks* of lines in system X to the maximum of T and the current *lineclock* value of the line. This would start phase 2 of the algorithm, i.e., the *lp* will again try to send output messages and receive input messages. If the value of T received is Z+2 then line clocks are not updated.

After receiving a *fire*(Z+2) message, an *lp* will continue to wait and receive messages on its input lines in system Y; and it will terminate itself once it has received a NULL message on each of these lines. (It is guaranteed that under these conditions, indeed it has sent or received a NULL message on every adjacent line).

4 OVERALL DESIGN OF OUR EXPERIMENTS

In our experiments we have a two level simulation. A sequential simulator simulates the behavior of the distributed simulator defined above while the distributed simulator is simulating the physical system (i.e., the queuing network being simulated).

4.1 Simulation Parameters

We first list below the various simulation parameters that were used in our experiments. Any items not listed here should be assumed to have their obvious default values (e.g., the time that an *lp* takes to receive an input message should be assumed to be zero). [1] Time up to which the physical system is to be simulated (Z), [2] mean and minimum service times at a delay *pp*, [3] branching probabilities at a fork *pp*, [4] number of initial jobs in the physical system (#IJ), [5] communication delay for event messages, also called NONNULL messages (NNCOMDEL), [6] communication delay for NULL messages (NCOMDEL), [7] communication delays for IDLE and FIRE messages (OVCOMDEL), [8] time to compute a NONNULL message by an *lp*, [9] time to compute a NULL message by an *lp* (NULLTM), [10] time to compute an IDLE message by an *lp* (IDLETM), and [11] time taken by the central processor to check if this is a point of termination (and break it) when it receives an IDLE message (CPTM).

4.2 Measures of Importance

The most important measure of performance is the actual speedup obtained by distributed simulation over sequential simulation, i.e., the ratio ASR = SST/DST of total elapsed times in the two methods of simulation:

SST = Sequential simulation time, i.e., the time taken by a sequential simulator to simulate some physical systems up to time Z.

DST = Distributed simulation time, i.e., the time

taken by a logical system to simulate the same physical system up to time Z.

Next we define some terms to capture various notions of "ideal speedup ratio". The first ideal speedup ratio, called ISR1, is defined to be simply N where N is the number of processors in the distributed simulator. This is the usual notion of ideal speedup ratio commonly used in the literature. Next we define more refined notions of ideal speedup ratio.

IDST2 (ideal distributed simulation time) = the time taken by a logical system to simulate the same physical system up to time Z, assuming that: (1) any *lp* has all input NONNULL messages available whenever it needs them in its computations; equivalently, any *lp* has all its input NONNULL messages available to it when simulation starts, (2) Communication delay is zero for both overhead and NONNULL messages (NCOMDEL=OVCOMDEL=NNCOMDEL=0), and time spent in any activity other than computing NONNULL message is zero. Then we define the ideal speedup ratio, ISR2, to be SST/IDST2.

Define a distributed simulation scheme to be a *BASIC-related* scheme if it involves simulating each *pp* by an *lp*. The well-known schemes of Chandy and Misra (1979, 1981), Jefferson and Sowizral (1982), Peacock, Wong, and Manning (1979) are *BASIC-related* schemes. Note that the ideal speedup ratio ISR2 indicates how much speedup can be achieved in simulating a given physical system by *any Basic-related* scheme. Also note that in the ideal concurrent simulation corresponding to the ISR2, if each *lp* spends the same amount of total time in processing NONNULL messages, then the ISR2 becomes the same as ISR1. However usually ISR2 is less than ISR1, making it a more refined "ideal speedup ratio".

Next we define ISR3 to capture an even more refined notion of "ideal speedup ratio":

IDST3 (ideal distributed simulation time) = the time taken by a logical system to simulate the physical system up to time Z with the assumption that NCOMDEL=OVCOMDEL=NULLTM=IDLETM =CPTM=0. In other words, the only time consuming activities in the distributed simulator are computation and communication of NONNULL messages. Then we define the ideal speedup ration ISR3 by ISR3 = SST/IDST3. Thus, ISR3 is the same as ASR when communication delay for all overhead messages is set to zero, and all overhead computation times are zero.

Note that ISR3 is an upper bound on the speedup ratio only for the given distributed simulation scheme under consideration. It may be possible to achieve higher speedup ratio than ISR3 via some other scheme. The reason is that a different scheme might produce input NONNULL messages earlier than the given scheme under consideration.

Obviously, in different simulation runs to determine ASR, ISR3, and ISR2, if we have the same events in the physical system (e.g., random numbers generated do not change the events of the physical system), then we would have ASR $\leq$ ISR3 $\leq$ ISR2 $\leq$ ISR1.

Our simulation program also records total number of overhead and NONNULL messages. This would be useful in considering whether it is reasonable in a particular case to say that "the poor performance is *due to* large overhead".

4.3 How To Compute The Performance Measures

In any experiment, with appropriate values for simulation parameters, the value of DST is measured directly by our sequential simulator that simulates the distributed simulator. In any such experiment we measure the total number of NONNULL messages sent on each line in the logical system during the distributed simulation. Using these values, we directly compute the sequential simulation time SST. In this computation we ignore the processing time for event list manipulations, and simply add the times taken by *lps* to compute each of their output NONNULL messages. In this way we can compute ASR.

The values of IDST2 are computed by simply measuring, in each of the ASR experiments, the number of NONNULL messages sent out by each *lp*. From these measurements, IDST2 can be directly computed. ISR2 can be computed from IDST2 and SST. Note that as the value of NNCOMDEL is varied, the values of IDST2 and ISR2 remain the same. Similarly, note that the values of NCOMDEL, OVCOMDEL, NULLTM, IDLETM and CPTM will not affect IDST2 or ISR2 since there are no overhead messages sent out in this case (except for possibly one NULL message on any line at the end of simulation, which we ignore in computing IDST2).

The value of IDST3 and ISR3 can be measured in the same way as DST and ASR by choosing appropriate simulation parameters as input.

4.4 Ordering Of The Experiments

An "experiment" here refers to a single execution of our simulation program. We ordered our experiments in the following manner. For a given value of the physical and logical system parameters (except for NNCOMDEL), we varied NNCOMDEL from 0 to a reasonably high value. For each such value of NNCOMDEL we conducted two experiments: (i) one to determine ASR and ISR2, and (ii) one to determine ISR3 and ISR2 (i.e., NCOMDEL, OVCOMDEL, NULLTM, IDLETM and CPTM are set to zero). Note that the values of ISR2 in these two experiments would be the same.

Then we repeated the above experiments for different values of physical system parameters.

5 FURTHER DETAILS ON OUR EXPERIMENTS

Our sequential simulator simulating the distributed simulator is implemented on SUN workstations. The sequential simulator mentioned above is written in the language MAY Bagrodia (1983). MAY is a simulation language for simulating distributed systems. It provides facilities for defining parameterized process types, instantiating processes at run time, message communications among processes, a global clock keeping the simulation time, conditional waiting for events, and process termination.

We simulated the three closed queuing networks shown in Figures 1, 2 and 3. These Networks are borrowed from Reed, Malony, and McCredie (1988).

5.1 Physical System Parameters

The service times at every delay *pp* in these experiments were chosen to be exponentially distributed, with the mean value equal to 3000.0 (this value is assumed in all our experiments) and minimum service time equal to 1.

The branching probabilities for the output branches of fork *pps* are specified in the later discussions for the individual networks.

Each physical system was simulated for the physical system time interval [0, Z] where Z is chosen such that a sufficiently large number of messages is sent through each line in the physical system.

5.2 Logical System Parameters

In all experiments, the time to compute one output NONNULL message for any given class of *lps* is: (i) delay : 300, (ii) fork : 90, (iii) merge : 120 and (iv) source : 60.

Experiments to Determine ASR and ISR2: The value of NNCOMDEL is varied from 0 to 6400. The values of NCOMDEL and OVCOMDEL are same as NNCOMDEL. The values of NULLTM are: (i) delay : 150, (ii) fork : 45, (iii) merge : 60 and (iv) source : 60. The values of IDLETM are: (i) delay: 15, (ii) fork : 15, and (iii) merge : 60. The value of CPTM is 150.

Experiments to Determine ISR3 and ISR2: NNCOMDEL here is varied as in case of determining ASR. By definition, NCOMDEL, OVCOMDEL, NULLTM, IDLETM and CPTM are zero.

For simplicity no times are associated with other activities in the logical system. In particular, no time is associated with receiving a NONNULL message.

6 SIMULATION RESULTS

Queuing Network 1:

Network 1 is a simple cyclic queuing network (see Figure 1) which is composed only of delays processes. Table 1 shows the main experimental results.

Table 1: Simulation Results Of Queuing Network 1 (C=NNCOMDEL)

#IJ	C→	0	200	400	800	1600	3200	6400	ISR2
2	ASR	1.9956	1.1978	0.8556	0.5444	0.3152	0.1711	0.0894	3.9948
	ISR3	1.9991	1.1995	0.8568	0.5452	0.3157	0.1714	0.0895	
4	ASR	3.9768	2.3873	1.7052	1.0850	0.6280	0.3409	0.1781	3.9963
	ISR3	3.9866	2.3920	1.7085	1.0873	0.6295	0.3417	0.1785	
8	ASR	3.9800	3.9664	3.3973	2.1671	1.2557	0.6817	0.3561	3.9877
	ISR3	3.9877	3.9852	3.4149	2.1761	1.2601	0.6841	0.3574	
16	ASR	3.9693	3.9611	3.9543	3.9401	2.4917	1.3549	0.7085	3.9761
	ISR3	3.9761	3.9739	3.9706	3.9619	2.5069	1.3635	0.7131	
24	ASR	3.9686	3.9619	3.9552	3.9373	3.7052	2.0134	1.0524	3.9750
	ISR3	3.9750	3.9708	3.9667	3.9584	3.7371	2.0315	1.0620	
32	ASR	3.9666	3.9586	3.9497	3.9334	3.9018	2.6610	1.3908	3.9728
	ISR3	3.9728	3.9708	3.9678	3.9598	3.9370	2.6915	1.4070	
40	ASR	3.9628	3.9574	3.9496	3.9341	3.9015	3.3061	1.7363	3.9673
	ISR3	3.9673	3.9654	3.9634	3.9585	3.9352	3.3526	1.7614	

The following points may be noted regarding this data.

1. At NNCOMDEL= 0 the value of ASR is close to the minimum of #IJ and the number of *lps* in the system - which is obviously fairly high usually.

As NNCOMDEL is increased, the value of ASR goes down - and becomes less than or roughly equal to 1 for sufficiently large value of NNCOMDEL.

Thus, for any given value of #IJ, there is a range of NNCOMDEL values between 0 and a small value (which depends on #IJ), such that in this range the value of ASR is high.

2. As #IJ is increased, the range of values of NNCOMDEL which gives high ASR values is also increased.

3. The value of NNCOMDEL has a significant impact on ASR. Therefore a good way to improve performance would be to cut down on NNCOMDEL.

4. Interestingly, for each entry of the Table the difference between ASR and ISR3 values is negligible. [This is true irrespective of whether ASR values are low or high]. This shows that:

4.1. Reducing OVCOMDEL, IDLETM or CPTM would not have any significant effect on ASR.

4.2. Consider variations of this distributed simulation scheme where one tries to reduce the number of overhead messages. Such variations are unlikely to significantly improve the performance.

5. Consider the cases where performance is poor, i.e., ASR values are less than or close to 1. (As indicated above, this happens when NNCOMDEL value is sufficiently high for any given #IJ value). It would be wrong in these cases to claim that the poor performance is "due to a large number of overhead messages":

5.1. since in these cases ISR3 is less than or close to 1, this indicates that even if, somehow, one could re-

duce these overhead messages to zero, the ASR value is unlikely to be significantly higher than 1.

5.2. Moreover the number of overhead messages was not large in these cases (as well as other cases in this table). For experiments reported in this Table, the number of overhead messages is much less than the number of NONNULL messages.

6. In many cases ISR3 is significantly less than ISR2 or ISR1. This shows that in studying variations of this scheme which try to reduce overhead, ISR3 captures the ideal speedup ratio in a more realistic manner than ISR1 or ISR2.

7. For a large number of cases (with low values of NNCOMDEL and high values of #IJ), the values of ASR are close to ISR2. Thus, for these cases, the distributed simulation scheme considered is nearly optimal with respect to all *BASIC-related* schemes.

8. The values of ASR shown here in Table 1 are slightly smaller than the corresponding values of ASR we obtained in an earlier study Harous (1991) of the same physical system using the distributed simulation scheme based on NULL messages. However the difference is negligible. The values of ISR3 are the same in both studies because in the simulation of this particular system the overhead messages have no effect in either scheme — in this scheme deadlock does not occur and in the scheme of Chandy and Misra (1979) the NULL messages do not help produce any event messages.

9. In the two studies of this network with the different simulation schemes, the values of ISR2 are the same, which immediately follows from the definition of ISR2.

Queuing Network 2:

This network is shown in Figure 2. As pointed out in Reed, Malony, and McCredie (1988) this queuing network is a good test for distributed simulation because of its topology (presence of a fork and a merge processes). In all experiments, the initial jobs are assumed to be at *pp 1*. Table 2 shows the main experimental results.

Table 2: Simulation Results Of Queuing Network 2 (With branching probabilities = 0.33, 0.34, 0.33 respectively for *pps* 3, 4 and 5) (C=NNCOMDEL)

#IJ	C→	0	200	400	800	1600	3200	6400	ISR2
2	ASR	0.6764	0.3448	0.2222	0.1298	0.0709	0.0372	0.0190	2.3553
	ISR3	0.9975	0.4911	0.3225	0.1946	0.1078	0.0570	0.0293	
4	ASR	0.8231	0.4276	0.2789	0.1643	0.0902	0.0474	0.0244	2.3557
	ISR3	1.1028	0.5545	0.3700	0.2221	0.1234	0.0653	0.0337	
8	ASR	1.5268	1.0190	0.7215	0.4433	0.2495	0.1330	0.0687	2.3586
	ISR3	1.8171	1.1937	0.8451	0.5275	0.3003	0.1611	0.0836	
16	ASR	2.2521	2.1017	2.0050	1.6007	0.9740	0.5370	0.2819	2.3586
	ISR3	2.3215	2.2423	2.1258	1.6529	1.0091	0.5577	0.2930	
24	ASR	2.3198	2.2377	2.2333	2.2073	1.7041	0.9606	0.5084	2.3586
	ISR3	2.3558	2.3520	2.3417	2.2832	1.7118	0.9652	0.5110	
32	ASR	2.3198	2.2394	2.2390	2.2354	2.1348	1.3270	0.7089	2.3586
	ISR3	2.3558	2.3525	2.3492	2.3422	2.2121	1.3362	0.7140	
40	ASR	2.3198	2.2394	2.2390	2.2364	2.1983	1.6949	0.9163	2.3586
	ISR3	2.3558	2.3525	2.3492	2.3427	2.3223	1.7099	0.9247	

The following points may be noted regarding this data.

1. For any given value of #IJ>8, at NNCOMDEL= 0 the value of ASR is high.

As NNCOMDEL is increased, the value of ASR goes down - and becomes <1 for sufficiently large value of NNCOMDEL.

Thus, for any given value of #IJ> 8, there is a range of NNCOMDEL values between 0 and a small value (which depends on #IJ), such that in this range the value of ASR is high.

2. As #IJ is increased, the value of ASR increases or remains the same. Also, the range of values of NNCOMDEL which gives high ASR values is increased.

3. The value of NNCOMDEL has a significant impact on ASR (the impact is higher when NNCOMDEL is large or when #IJ is small). Therefore a good way to improve performance would be to cut down on NNCOMDEL.

4. Notice that in a large number of cases the difference between ASR and ISR3 values is negligible. [This happens in the cases where we have large values of #IJ (say #IJ $\geq$ 16). This shows that in such cases points 4.1 and 4.2 in the previous discussion of network 1 remain true.

5. Consider the cases where performance is poor, i.e., ASR values are < 1. (As indicated above, this happens when NNCOMDEL value is sufficiently high or when #IJ value is small). It would be wrong in these cases to claim that the poor performance is "due to a large number of overhead messages":

Point 5.1 in case of network 1 remains true here.

[It is true that the number of overhead messages is somewhat large in some of these cases. For experiments reported in this Table, the ratio of number of overhead messages to NONNULL messages was about 2.2 for #IJ=2, and reduced to about 0.66 as #IJ was increased to 40.]

6,7. these points are the same as points 6 and 7 in the case of network 1.

8. The values of ISR2 are significantly less than ISR1 in these experiments. This clearly shows that in studying variations of *BASIC-related* schemes, ISR2 captures the ideal speedup ratio in a more realistic manner than ISR1.

9. The values of ASR and ISR3 in these experiments are slightly smaller than the corresponding values of ASR and ISR3 reported in an earlier simulation of this system using the distributed simulation scheme based on NULL messages Harous (1991). This can be understood in the following way. Suppose some NONNULL messages are available at the input of a merge *lp* but the *lp* is waiting for further information on its other input lines. In such cases it is possible that NULL messages may arrive on these other input lines even though there is no deadlock. In such

a case, the merge *lp* may be able to send out some of its input NONNULL messages. In contrast, in the deadlock detection based schemes, the merge *lp* may have to wait until a deadlock occurs, which may happen much later.

10. This point is the same as point 9 in the case of network 1.

In addition to the experiments shown in Table 2, we also did experimentations with varying branching probabilities at *pp 2*. This factor does affect ASR, ISR3 and ISR2 values, but mildly only. Also, the observations made above regarding data in Table 2 remain true. Tables corresponding to different branching probabilities are given in Harous (1991).

Queuing Network 3:

This network is shown in Figure 3, and was studied in Reed, Malony, and McCredie (1988). In all experiments with this network, the initial jobs are assumed to be at *pp 1*. We considered a few variations of the probability assignments at the fork pps in the system. Below we consider three such probability assignments in turn.

Probability Assignment 1: Table 3 shows the main experimental results for the indicated values of the branching probabilities. The results here are quite different from what we have seen previously in the cases of networks 1 and 2. Specifically, the following points may be noted regarding this data.

Table 3: Simulation Results Of Queuing Network 3
(With branching probabilities: 0.25 for each output line at *pp* 1, 0.05 for each input line of *pp* 10)
(C=NNCOMDEL)

#IJ	C→	0	200	400	800	1600	3200	6400	ISR2
2	ASR	0.6746	0.3608	0.2342	0.1376	0.0753	0.0396	0.0203	6.2335
	ISR3	1.0014	0.5007	0.3338	0.2003	0.1113	0.0589	0.0304	
4	ASR	0.6777	0.3620	0.2349	0.1380	0.0756	0.0397	0.0204	6.6379
	ISR3	1.0028	0.5015	0.3343	0.2006	0.1114	0.0590	0.0304	
8	ASR	0.6813	0.3633	0.2357	0.1384	0.0758	0.0398	0.0204	7.0777
	ISR3	1.0051	0.5022	0.3347	0.2008	0.1115	0.0590	0.0304	
16	ASR	0.6848	0.3650	0.2369	0.1392	0.0761	0.0400	0.0205	7.5345
	ISR3	1.0090	0.5041	0.3360	0.2016	0.1120	0.0593	0.0305	
24	ASR	0.6870	0.3661	0.2376	0.1396	0.0765	0.0402	0.0206	7.7963
	ISR3	1.0109	0.5051	0.3366	0.2019	0.1122	0.0594	0.0306	
32	ASR	0.6869	0.3661	0.2376	0.1396	0.0765	0.0402	0.0206	7.8384
	ISR3	1.0130	0.5061	0.3373	0.2023	0.1124	0.0595	0.0307	
40	ASR	0.6887	0.3773	0.2383	0.1400	0.0767	0.0403	0.0207	7.9385
	ISR3	1.0172	0.5081	0.3386	0.2031	0.1128	0.0597	0.0308	
64	ASR	0.6947	0.3713	0.2411	0.1417	0.0777	0.0408	0.0209	8.0378
	ISR3	1.0301	0.5148	0.3431	0.2059	0.1144	0.0605	0.0312	

1. The value of ASR is very low (< 0.7) in all cases. As NNCOMDEL is increased, the value of ASR goes down very quickly.

2. As #IJ is increased, the value of ASR remains almost the same.

3. Even though NNCOMDEL has a significant impact on ASR, simply trying to reduce NNCOMDEL is not going to help much, since ASR is always very small.

4. Notice that in a large number of cases there is a significant difference between ASR and ISR3 values. (This is quite different from what we have generally seen in networks 1 and 2). This shows that in such cases:

4.1. Reducing OVCOMDEL, IDLETM and CPTM would improve ASR somewhat but this improvement is insignificant as discussed in point 5 below.

4.2. There is a very small potential for improving the speedup by devising variations of this distributed simulation scheme where one tries to reduce the number of overhead messages since ISR3 is less than or close to one.

5. Note that in all cases the performance is poor (i.e., ASR values are less than or close to 1). It would be wrong in these cases to claim that the poor performance is "due to a large number of overhead messages":

Point 5.1 in case of network 1 remains true here.

[Incidentally, it is true that the number of overhead messages is a bit large. For experiments reported in this Table, the ratio of number of overhead messages to NONNULL messages was about 3.5 for #IJ=2, and decreased to about 3.3 as #IJ was varied to 64.]

6. In all cases shown in the Table, ISR3 is very small compare to ISR2 or ISR1. Therefore point 6 regarding Network 1 still holds here.

7. In all the cases shown in the Table, the values of ASR are significantly less than ISR2. This shows the possibility that ASR values may be increased significantly by considering other *BASIC-related* schemes.

8. This point is the same as point 8 in the case of network 2.

9. The values of ASR and ISR3 shown here are much smaller than the corresponding values of ASR and ISR3 reported in the earlier study of this system using the distributed simulation scheme based on NULL messages Harous (1991). This difference can be explained in the same way as point 9 in the case of network 2.

10. This point is the same as point 9 in the case of network 1.

Probability Assignment 2: This is the case when the branching probabilities were 0.25 for each output line at pp 1, and 0.5 for each input line of pp 10.

In this case the values of ASR and ISR3 are slightly better, and a few cases where the ASR values are slightly higher than 1. But in general the results in this case are also poor. The nature of our conclusions in this case is similar to our previous conclusions regarding Table 3, with the exception that #IJ has some affect on both ASR and ISR3. A detailed discussion is skipped here. The results are given in Harous (1991).

Probability Assignment 3: In this case we chose the probabilities to be 0.45, 0.45, 0.05, and 0.05 for the

output lines of pp 1 going to pps 2, 6, 11, and 15 respectively. Also, the probabilities are 0.5 for each input line of pp 10.

The results in this case are similar in nature to those in Table 3. Therefore we skip further details here. These results are given in Harous (1991).

7 DISCUSSION

In this paper, we have presented a framework to address three important issues (discussed below) in studying performance of distributed simulation schemes. (Here we are considering only *BASIC-related* schemes, i.e., there is a one to one correspondence between the *pps* and the *lps* simulating them.)

1. How to study the relationship between overhead and performance in a controlled manner. This is a difficult problem if one tries to see the relationship between the *number* of overhead messages and the performance. In our approach, one would vary the *time parameters* related to computation and communication of overhead messages (e.g., IDLETM, CPTM and OVCOMDEL). This approach leads to some interesting results:

(a) Typically, researchers have either implemented a logical system on an actual hardware or have simulated the logical system with a fixed set of values for computation and communication times of overhead messages. If performance is poor (along with an observation that the number of overhead messages is large) then some researchers have suggested that (A) *performance is poor due to large number of overhead messages.* Such a claim (i.e., in cases where performance of distributed simulation is poor, it is *due to* a large amount of overhead) is also generally believed by other researchers. Our approach, and the experimental results reported here, shed some new light on these claims.

The claim A would normally be taken to mean that (i) the number of overhead messages is large, (ii) if somehow one could reduce the amount of overhead (without getting into deadlocks in the simulator), then the performance would be good (i.e., speedups fairly above 1), and (iii) perhaps, if overhead is high then speedup would always be bad. Note that there is no clear way to reduce the number of overhead messages significantly in a controlled manner and study the effect on speedup, and thereby validate the above statement (A). But our results show that indeed the statement (A) is not true in many cases. We have pointed this out in our remarks on Tables 1, 2, and 3 (observations 5 in section 6).

(b) Also, by varying computation and communication times of overhead messages, our approach indicates how effective it would be to try to reduce overhead messages. In our experiments we did not specifically vary these parameters much; by considering the cases where OVCOMDEL=IDLETM=CPTM=0 we were able to arrive at some useful conclusions, as discussed in more detail in 2(a) below.

2. What kind of ideal speedup one could expect via algorithmic variations for the simulation scheme. Here we are given a physical system and characteristics of the hardware on which simulations are executed in terms of processing times of the event messages and their communication delays. More specifically:

(a) if one varies details of a given distributed simulation scheme so as to try to reduce the number of overhead messages, how much speedup can be expected from most ideal such variation? We defined ISR3 to capture this. Our experimental results indicate that ISR3 is a useful measure of ideal speedup.

(i) In several cases (e.g., points 4 for networks 1 and 2 in section 6) we noticed that the difference between ASR and ISR3 is small — and hence this tells us that trying to reduce the amount of overhead itself is not going to substantially increase the speedup. In some cases the value of ISR3 is less than or close to 1 which shows that overhead related variations of the particular scheme studied here is unlikely to produce significant speedup.

(ii) In the case of network 3 we observed that there is no hope for the scheme studied here since in all cases both ASR and ISR3 are less than or close to 1.

(b) If one considers other *BASIC-related* schemes, how much speedup can be expected from the best such simulation? We defined ISR2 to capture this.

Typically, researchers have used N, the number of *lps* in the system, as the ideal speedup. The values of ISR2 and ISR3 are usually much less than N (as shown by our experiments); which shows that our definition of ideal speedups provides a much tighter bound on what kind of ideal speedup can be expected.

Note that in order to evaluate ISR3, one needs to *simulate* the logical system, as opposed to *directly implementing* it on an actual hardware. Using the approach of *simulating* the distributed simulator first and measuring the ASR, ISR3, and ISR2 values one could make some determination of whether it is worthwhile to directly implement it on a parallel or distributed architecture.

3. How does performance of distributed simulation depend on hardware characteristics of the implementation. Again, our approach of *simulating* the logical system is very helpful in this regard. In particular, we illustrated how to see the effect of varying NNCOMDEL. Obviously using our approach of *simulating* the logical system, one can vary other parameters of logical system as well in a straightforward manner.

ACKNOWLEDGMENTS

The first author is thankful to Professors K. M. Chandy and J. Misra for several fruitful discussions on distributed simulation, including their idea of simulating the distributed simulator and Professor Chandy's thought of comparing the *conditional-next-time* with the *lineclocks* of lines in system Y. The first author is also thankful to them for their financial support under the Air Force Grant AFOSR 81-0205 during which he developed part of the simulation software. We are especially thankful to them and to Professor R. Bagrodia for the use of their May system.

This material is based upon work supported by the National Science Foundation under Award No. CCR-9110347.

REFERENCES

Bagrodia, R. 1983. May: A Process Based Simulation Language. *Master's Report*, Dept. of Computer Sciences, University of Texas at Austin, Austin, Texas.

Chandy, K.M., and J. Misra. 1979. Distributed Simulation: A Case Study In Design And Verification of Distributed Programs. *IEEE Transactions on Software Engg.*, SE-5(5): 440-452.

Chandy, K.M., and J. Misra. 1981. Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. *Communications of the ACM*, Vol. 24, No. 4: 198-205.

Dijkstra, E.W., and C.S. Scholten. 1980. Termination Detection for Diffusing Computation. *Information Processing Letters*, Vol. 11, No. 1.

Fujimoto, R.M. 1988a. Performance Measurements of Distributed Simulation Strategies. *Proceedings of the SCS Multiconference on Distributed Simulation*, 14-20.

Fujimoto, R.M. 1988b. Lookahead in Parallel Discrete Event Simulation. *Proceedings of the 1988 International Conference on Parallel Processing*, 34-41.

Fujimoto, R.M. 1989. Time Warp on a Shared Memory Multiprocessor, *Proceeding of the 1989 International Conference on Parallel Processing*, 242-249.

Harous, S. 1991. Studies in Distributed Simulation. *Ph.D. Dissertation*, Department of Computer Engineering and Science, Case Western Reserve University, Cleveland, Ohio.

Jefferson, D.R., and H.A. Sowizral. 1982. Fast Concurrent Simulation Using The Time Warp Mechanism, Part I: Local Control. *Technical Report*, The Rand Corporation, Santa Monica, California.

Kumar, D. 1987. Efficient Algorithms For Distributed Simulation And Related Problems. *Ph.D. Dissertation*, Department of Computer Sciences, University of Texas, Austin, Texas.

Kumar, D., and S. Harous. 1990. An Approach to Study Performance Properties of Distributed Simulation. *The Second IEEE Symposium on Parallel and Distributed Processing*, 866-869, Dallas, Texas.

Kumar, D., and S. Harous. 1991. A Study of Achievable Speedup in Distributed Simulation via NULL Messages. 10^{th} *Annual IEEE International Phoenix Conference on Computers and Communications*, 113-119, Scottsdale, Arizona.

Peacock, J.K., J.W. Wong, and E.G. Manning. 1979. Distributed Simulation Using A Network of Processors. *Computer Networks* 3(1): 44-56.

Reed, D.A. 1983. A Simulation Study of Multimicrocomputer Networks. *Proceedings of International Conf. on Parallel Processing*, 161-163.

Reed, D.A., A.D. Malony, and B.D. McCredie. 1988.Parallel Discrete Event Simulation Using Shared Memory. *IEEE Transactions on Software Engineering*, 14, 4: 541-553.

Seethalakshmi, M. 1979. A Study And Analysis of Performance of Distributed Simulation. *Master's Report*, Dept. of Computer Sciences, University of Texas, Austin, Texas.

AUTHOR BIOGRAPHIES

DEVENDRA KUMAR is an assistant professor of Computer Engineering and Science at Case Western Reserve University. He received a Ph.D in Computer Sciences from The University of Texas at Austin in 1987. His research interests are parallel and distributed processing, program verification and simulation. He is a member of ACM and IEEE.

SAAD HAROUS is a Ph.D candidate in the Department of Computer Engineering and Science at Case Western Reserve University. He graduated as an "Ingenieur d'informatique" from the University of Constantine, Algeria in 1984. He received an M.S in Computer Sciences from Case Western Reserve University in 1987. His research interests are in distributed processing and simulation. Saad is a member of ACM and IEEE.