DISTRIBUTED COMPUTER SIMULATION OF DATA COMMUNICATION NETWORKS: PROJECT REPORT I

Shun Cheung
Jack W.Carlyle
Walter J. Karplus

August 1984
Report No. CSD-840037

Acknowledgment of Support

Work leading to the preparation of this report was supported jointly by the State of California and by Doelz Networks, Inc. under the UC-MICRO grant "Distributed Computer Simulation of Data Communication Networks."

Abstract

This is the initial report on a continuing study of the development of methodologies for implementation of distributed simulation, with application to the problem of estimating the performance of data communication networks, and with particular reference to an extended slotted ring architecture for communication. Items discussed include simulation modeling considerations, timing/synchronization among the processors, deadlocks, and preliminary experiments which have been carried out.

Table of Contents

page

1. Introduction 1
   1.1 Long-Range Project Goals 1
   1.2 Available Resources 2
   1.3 Summary of Initial Studies and Results 2
   1.4 Organization of this Report 3
2. Overview of Technical Issues 3
   2.1 The Selection of a Suitable Simulation Language 3
   2.2 Simplification of the Simulation Model 4
      2.2.1 Flow equivalent service centers 4
      2.2.2 Primary/background service centers 6
   2.3 Partition and Allocation 6
   2.4 The Implementation of Distributed Simulation 7
3. Synchronization Methods 7
4. Problems of Asynchronous Distributed Simulation 9
   4.1 Deadlock 9
      4.1.1 The Null Message Method 11
      4.1.2 The Virtual Ring Method 12
   4.2 Collection of Statistics 13
5. The Processing Environment at UCLA 15
6. Experimental Work 15
7. Conclusions and Future Directions 18
8. References 19

1. Introduction

This is the first in a projected series of reports on work-in-progress, supported by the research grant "Distributed Computer Simulation of Data Communication Networks" (DCSDCN), sponsored jointly by the State of California and by Doelz Networks, Inc., under the University of California/Microelectronics Innovation and Computer Research Opportunities (UC-MICRO) program.

Research being initiated in the DCSDCN project is broadly directed toward understanding and implementing discrete-event systems simulation via distributed processing in networks of computers, with application to the simulation of data communication networks, using a particular DCN as a source of examples and a test case in our development of simulation tools. This model DCN is based on an extended slotted ring architecture (ESRA) for communication [DOEL 84] and will be referred to here as the Doelz network.

1.1 Long-Range Project Goals

A principal long-range objective of our project is to develop a methodology for application of distributed simulation using networks of microcomputers or workstations as the distributed processing resource with which to carry out the computations in the simulation runs. With reference to data communication applications in particular, a long-range goal is to develop simulation tools, running in this distributed processing environment, which may be of use in field-engineering situations where performance of proposed data network installations must be estimated.

We use the term "simulation" here in its broad sense to mean computer-assisted modeling of physical systems, incorporating appropriate analytical techniques and results as well as actual simulation computations using random-number generation and simulation software systems. Simulation is not regarded as a goal in itself, but rather as one tool useful in, e.g., estimating performance of a complex system such as a data communication network.

Our decision to focus upon a distributed processing environment in which to carry out simulations is motivated by pragmatic considerations, rather than by any desire to establish that simulations can be done "more efficiently" in some absolute sense when distributed instead of being done on a single large processor. The latter is a current research issue over which there has been some controversy. Our point of view is that networks of microcomputers or workstations are rapidly emerging as a medium-scale computing resource which will be very widely available in the near future (accessible to field engineers, for instance), that complex simulation applications may be beyond the capacities of individual stations in such networks but within the capabilities of the network as a whole, and therefore that distributed simulation in these computing environments can and should be contemplated as a practical matter, with attention to efficiency of execution within the stated environment.

1.2 Available Resources

At UCLA, within the Computer Science Department, computing resources available for research use are centered upon a network of minicomputers (Digital Equipment Corporation VAX machines) with a UCLA-developed network-transparent operating system, LOCUS, whose facilities for distributed processing are of particular interest to our project. This environment will be enlarged in the near future in several directions, with microcomputer workstations a projected feature, again with LOCUS available as the distributed operating system. This provides a very appropriate computing environment in which to develop and test ideas and techniques applicable to our project.

1.3 Summary of Initial Studies and Results

In this first phase of the project, we have kept the LOCUS operating system features in mind and our initial experimentation in simulation software systems development is being carried out on the minicomputer network, since the results will be transportable to workstations in the near future.

Specifically, our initial studies have been directed toward understanding and controlling the consequences of time-asynchronous operation in carrying out distributed simulations, and some very preliminary systems experiments have been carried out as an aid in identifying the issues.

We have concentrated on timing and synchronization problems at the outset because they arise immediately when experimentation is attempted, even on idealized models. Other issues, including details more specific to our data communication network application, will be treated in future reports.

In summary we have:

- examined some timing problems of distributed simulation
- proposed some methods for addressing preemption and deadlock problems

- carried out preliminary experiments in distributed simulation to verify feasibility and identify problems.

1.4 Organization of this Report

An overview of several technical issues concerning distributed simulation is given in Section 2. Synchronization methods, deadlock problems, and possible solutions are presented in Section 3 and 4 respectively. Section 5 contains further details on the computing environment and the LOCUS operating system currently available at UCLA for our use in research. Section 6 describes the preliminary experiments we have carried out recently and some of the problems identified in the course of the experimentation. Conclusions and possible directions to be taken in continuing this work are enumerated briefly in Section 7.

2. Overview of Technical Issues

Before distributed simulation can be implemented, a number of problems introduced by the new processing environment must be resolved. As in other multiprocessor applications, in distributed simulation it is important in general to be able to divide the load on the processors in such a way as to optimize the computation time, or at least to improve the time in comparison to single-processor implementations, even if complete optimization is not attempted. Moreover, there are specific problems to be considered, such as process synchronization, deadlock and collection of statistics for the results of distributed simulation being conducted.

2.1 The Selection of a Suitable Simulation Language

Before any simulation can begin, a model for the application to be simulated, such as a communication network, must be specified in a computer-readable form. Specification through the use of general-purpose discrete-event simulation languages is a widely used approach. Many such languages are available; GPSS and SIMSCRIPT are among the most popular ones. There are also several newly developed languages specially designed for the simulation of communication networks [BALA 84]. Although these languages are very useful tools, they are intended for use on target computer systems with one CPU. For example, SIMSCRIPT is compiled into Fortran, which is not in itself suitable for distributed processing.

In order to take the full advantage of the processing power of a distributed computer system, significant portions of the inherent parallelism in a model must be preserved by the simulation language and its implementation. One alternative is to select a suitable existing simulation language and develop a new compiler for distributed simulation; significant modification of the syntax of the language may also be necessary. Another alternative is to design a special language for distributed simulation of data communication networks. Much additional investigation is needed in this area, which is part of current research in distributed processing in general. A third alternative is to develop software modules in an existing general-purpose programming language, such as C, and to implement the distributed features of the overall software simulation package by exploiting distributed processing features of a network operating system. This is the approach we have chosen for the initial stage of our work; the operating system is LOCUS, a network-transparent extension of UNIX, and C is the language we have used in implementing preliminary experiments (see Sections 5 and 6 further details).

2.2 Simplification of the Simulation Model

A fundamental problem in any complex application of discrete event simulation is the substantial amount of computation time required by simulation runs. For a large model, the time needed to reach steady state may be prohibitively long. It is then necessary to reduce the execution time by simplifying and approximating the behaviors of portions of the model, as a substitute for the computation-intensive task of simulating every node, queue and buffer in the physical system being modeled. From a basic modeling point of view, introduction of such simplifications is part of the effort to "reduce dimensionality," and has sometimes been referred to as "aggregation" in the literature of queueing networks.

In recent years, two approximation methods for the simulation of communication systems have been studied: flow equivalent service centers (FESC) [SAUE 81] and primary/background service centers [OREI 84].

2.2.1 Flow equivalent service centers

The simulation model of a communication network may be simplified by replacing one or more subnetworks by FESCs [SAUE 81]. A typical FESC has one server and a single queue. As the name implies, a FESC behaves exactly like, or at least very similarly to, the (sub)network it represents. Usually, this type of service center has a variable service rate which is a function of the number of customers in the corresponding subnetwork. For example, the two-queue system in Figure 1 may be replaced by a FESC.

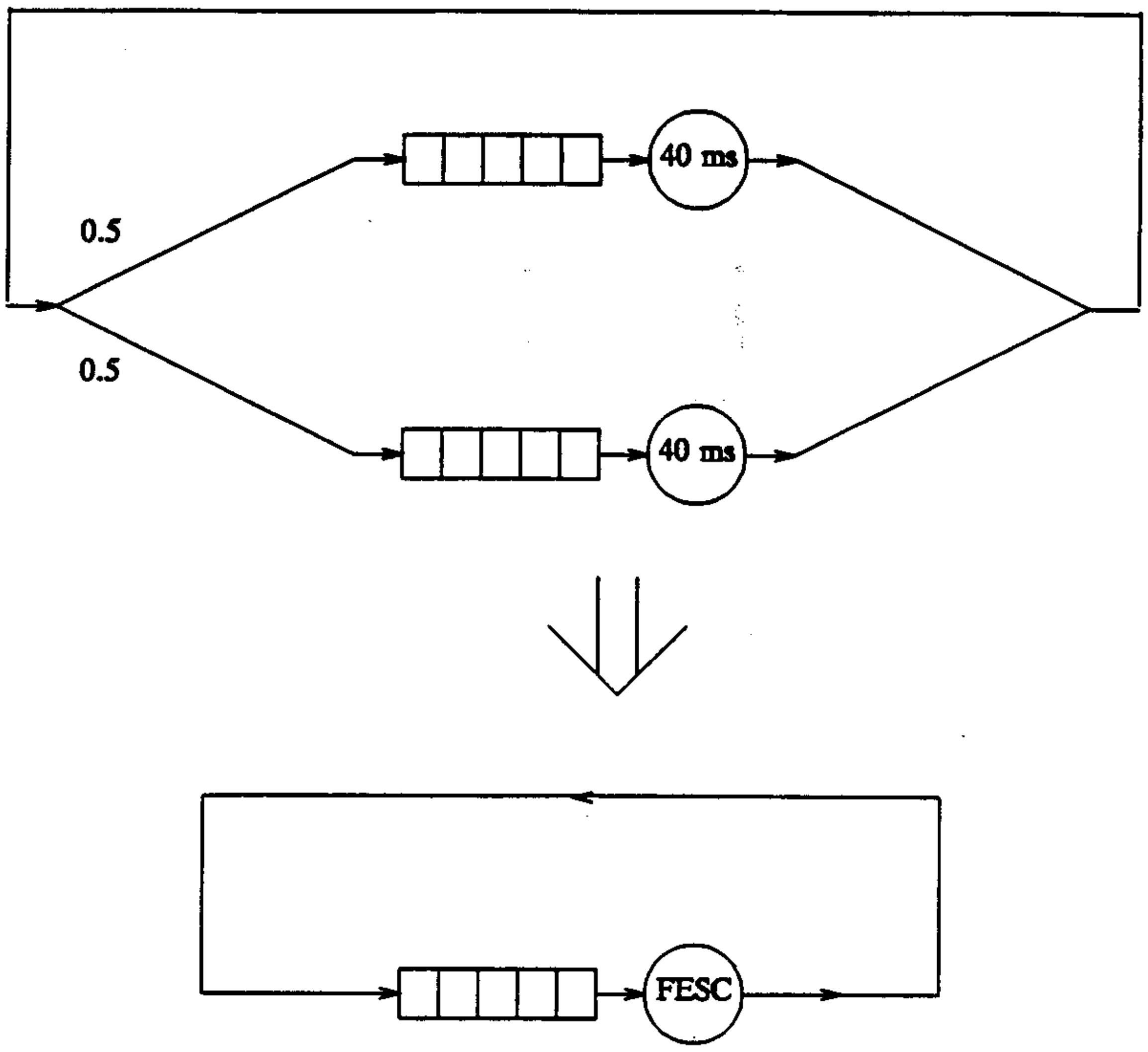

Figure 1: A Closed Two-queue System and its corresponding FESC

Referring to Figure 1, if there is only one job in the system, clearly, only one of the servers will be busy and the mean interdeparture time is 40 msec. If there are two jobs, when they are in the same queue, the mean interdeparture time is still 40 msec. Otherwise, it becomes 20 msec. Since the probabilities of the two cases are the same, the overall interdeparture time is 30 msec and the throughput is 33.3 jobs/sec. When there are many jobs in the system, the probability that both service centers are busy is very high and the mean interdeparture time will approach 20 msec. (throughput = 50 jobs/sec.) from above.

In this simple example, the load-dependent service rate is straightforward to calculate. However, when the subnetwork to be replaced is complicated, the service rate function may become very difficult to evaluate and is not always analytically solvable. A possible alternative is to simulate the behavior of the subnetwork separately with varying numbers of customers. The service rate function can then be derived or tabulated from the statistics obtained.

2.2.2 Primary/background service centers

A primary/background approximation technique has recently been developed [OREI 84] for the simulation of local area networks (LAN) with CSMA/CD access protocol. This method classifies the network stations into primary stations and background stations. Primary stations are simulated precisely while the collective effects of the background stations are approximated using a separate background algorithm. The main function of the background stations is to provide an environment to approximate the interactions among the primary stations and the other stations. Experimental results indicate that this method provides a good approximation of the behaviors of the primary stations. However, since the events in the background stations are not actually simulated, no results can be obtained for these stations. Currently, this method is developed only for the simulation of LAN with CSMA/CD but may be generalizable for a larger class of networks.

2.3 Partition and Allocation

Ideally, a multiprocessor system for distributed simulation might be set up in the same manner as the graph of the application model. In this case, each processor (workstation) represents a node (service center) in the model, and the virtual connections among the processors are identical to the arcs in the graph. In other words, there is a one to one correspondence between the model graph and the multiprocessor network. The advantage of this arrangement is that the relation between the model and the real system is clear so that information such as queue length and deadlock can be obtained relatively easily. However, the number of processors available is in general not equal to, and may be far less than, the number of nodes in the model graph, even when simplifications and approximations have already been introduced to reduce the model's dimensionality. If there are more nodes than processors, it will be necessary to combine some of the nodes together and assign more than one node to certain processors. Clearly, there are many different ways to combine nodes, and some of them will have better utilization of the computation power. It is then necessary to develop a method for assigning the work load to the processors which is advantageous in terms of processor cycle utilization.

2.4 The Implementation of Distributed Simulation

Once the partition and allocation of the model is completed, actual simulation may be carried out. In a distributed environment, there are several approaches which deal with the problems of synchronization of the processors. A taxonomy tree which classifies the synchronization methods may be found in [PEAC 79]. Moreover, there are problems with deadlocks and collection of statistics which are very different from their counterparts in conventional simulation. These issues will be discussed in Sections 3 and 4 respectively.

3. Synchronization Methods

Discrete event simulation is usually classified into two types: time-driven and event-driven. Time-related problems arise in both, as follows.

In time-driven discrete event simulation, the simulated clock is advanced by a constant amount every time; that is, the interval is of fixed length. After each advance, the simulator will check every part of the model. If there are events which should take place in that interval, they will be simulated. After all event simulations have been carried out, the clock will be advanced again. A dilemma for time-driven simulation is that if the time increment is large, many events will occur in each interval and, depending on the order in which the nodes are inspected, some of the events may be simulated out of order and hence the correctness of the simulation is violated.* If the time increment is very small, only a few events will occur in each interval and the portion of computation time wasted on overheads such as checking each node in the system will increase.

*Here we refer to the correctness of the simulation in the sense of exact reproduction of event sequences as they would occur in the system being modeled; it may be, in a particular application, that some out-of-sequence behavior in the simulation is not harmful because the performance of the physical system is relatively insensitive to sequencing. For instance, in our data communication network application, the actual system may implement virtual circuits and incorporate sequence numbers or time stamps as tags. However, the distinction between actual system behavior and artifacts of the simulation must be kept in mind, with the objective of eliminating artifacts which would actually be misleading in terms of performance prediction.

In event-driven simulation, the events are sorted into an event list according to their time of occurrence. When an event-driven simulation is carried out, the event which has the earliest occurrence time (top of the list) will be simulated next. When a new event is generated, it will be inserted into the right place on the list. The concept of event list is very suitable for realization on conventional computer systems.

In distributed simulation, it is still possible to be in a time-driven mode, but it is then necessary to have centralized or distributed control which detects the termination of a simulation time interval (increment) for every processor and then enables each processor to start a new time interval. If a centralized controlling processor is used, this processor has to communicate back and forth with each processor at every time step, so it may easily become a bottleneck when the number of processors is large. Moreover, the problems with large and small increments remain in a distributed environment and affect the efficiency of the simulation.

In a distributed environment, the method of event lists is less suitable for two reasons. First, since each processor is contributing new events to the list, it becomes difficult to sort the event list in a distributed manner. More importantly, after an event has been initiated, it is not always appropriate to initiate the next event on the list even though the second event may be executed on a different processor. The reason is that the first event may generate a third event which should be simulated before the second. This situation is called *preemption*. As a result, if pure event-driven simulation is implemented, even in a distributed environment, it is possible to simulate at most one event at a time and hence only one processor may be busy. Clearly, under these circumstances, a processor network would provide no improvement over a single-processor system as far as execution time is concerned. An alternative is to save the current state of the system and then simulate the second event. If a preemption occurs, the simulation of the second event is aborted and the previous state is restored. This method has the drawback that a significant amount of memory and computation overhead is required to save the states of the system.

For both the time driven and event driven methods described above, the processors are synchronized because each one of them has approximately the same simulation time. On the other hand, the main advantage of asynchronous simulation is that it allows a higher degree of concurrency. Since the events in a simulation run are in general only partially interdependent, that is, there is a large amount of inherent concurrency, some events may be simulated in a different order from that in which they actually occur in the actual system without violating the correctness of the simulation. As a result, unnecessary processor idleness may be reduced. The correctness of the simulation is achieved by preventing preemptions completely. Preemptions may be caused by messages coming from the same arc or a different arc from which the message being served arrived. The requirement that messages passing through each arc must have an monotonically increasing simulation time avoids preemptions by messages from the same arc. Strategies to avoid preemptions by messages arriving from a different arc are discussed in Section 4.1.

The trade off for this high concurrency is that the correspondence between the actual system and the simulation becomes less clear. Since different parts of the model have different simulation times, it is no longer possible to take a "snap shot" of the model and relate it to the corresponding snap shot of the actual system. (In a sense, a model being asynchronously and distributively simulated is in a world with four dimensions. Various values of the fourth dimension, namely time, may exist "simultaneously" in different parts of the model.) As a result, the collection and merging of statistics becomes very different from conventional practice.

4. Problems of Asynchronous Distributed Simulation

In asynchronous distributed simulation, the strategies which prevent preemptions introduce an unfavorable side effect, namely deadlocks as artifacts of the simulation which are not necessarily present in the physical system being modeled. Deadlocks in simulation can also be caused by deadlocks appearing in the actual system being simulated, as well as by deadlocks arising from the operating system being used. It is important to distinguish these different types of deadlocks. (Ideally, a good simulator should also be able to detect deadlocks in the actual system, rather than to "hang up" without controllable recovery in these situations.) In this section, our discussion is centered on deadlocks as artifacts of simulation, and on problems of collection of statistics from the simulation runs. Some possible operating-system problems are mentioned in Section 6.

4.1 Deadlock

One of the major problems in asynchronous distributed simulation may arise when a node has more than one incoming arc as shown in Figure 2.

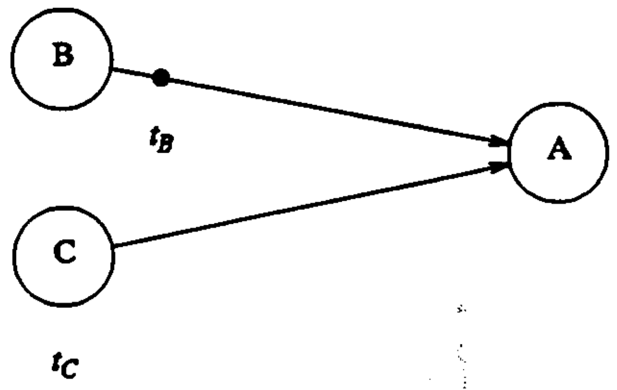

Figure 2: A Node with Multiple Inputs

In simulation, the simulation time of a message should be approximately equal to the time when this message is generated in the actual system. The simulation time of a node is defined to be equal to the simulation time of the last (most recent) message generated by this node. In time-driven distributed simulation and event-driven synchronous distributed simulation, a message which arrives at node A in Figure 2 (in the simulation) from either one of the arcs is guaranteed to correspond to the next arriving message in the actual system. This is not true, however, in asynchronous distributed simulation. For example, assume a message is sent from node B to A in Figure 2. Let the simulation time of this message be t_B, which is also the simulation time of node B when this message is generated. If t_B is ahead of t_C, the simulation time of node C, it is possible that node C will send a message with a smaller simulation time to A later on in the simulation. Therefore, in order to maintain the correctness of the simulation, node A should not consume the message from B until C also has a message ready to be sent to A. This requirement further guarantees the absence of preemption by messages from different arcs. When node A has one or more empty incoming arcs, it is said to be *blocked*. When there is at least one message at each input arc, a comparison can be made on the simulation times of the messages and the one with the smallest time should be consumed first. Unfortunately, the requirement that there is at least one message at each arc is likely to cause inefficiency and even deadlock during a simulation run. For example, if there is a large number of messages sent from B to A but only very few from C to A, messages sent along the BA arc will be unnecessarily delayed due to the lack of messages at the other arc for simulation time comparisons. A deadlock occurs if there are no messages being sent from C to A in the actual system so that messages at the BA arc will be waiting forever.

Two approaches to avoid such deadlocks are the Null Message Method and the Virtual Ring Method, both of which will be discussed here.

4.1.1 The Null Message Method

A simple example of deadlock is shown in Figure 3.

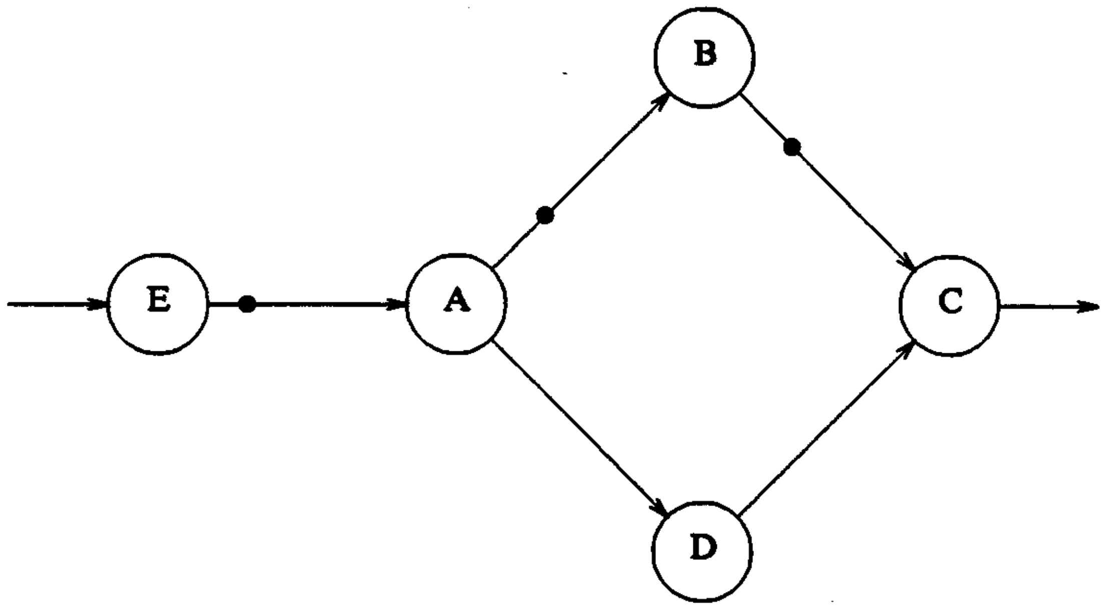

Figure 3: A Deadlock

Again, at node C, there is the trouble-causing situation where a node has multiple input arcs. If we make the simplification that there may not be any queues at the arcs, and each node has storage for at most one message, three consecutive messages from A through B to C will create the situation shown in Figure 3. (Assume that no message is stored in D initially.) C cannot accept the message from B because there is no incoming message from D for simulation time comparison. Meanwhile, B cannot accept the message from A because it is already occupied by a message. For the same reason, A cannot accept the message from E. Hence no message can be sent from A to C through D either. As a result, the message B intends to send to C can never get a simulation time comparison with a message from D. The system is deadlocked! (If there are queues along the arcs, deadlock is still possible when both queues in the ABC path are full.)

Chandy and Misra [CHAN 78] have suggested the use of *null messages* to solve this type of deadlock problem. At node A where there are two outgoing paths, whenever a message is sent from one outgoing path, a null message should be sent to the other path. The null message has the same simulation time as the "real message" but contains no other information. In the example in Figure 3, when two consecutive messages are sent form A to C through B, two null messages should be sent from A to B through D. With null messages, comparison of simulation times of messages going into C becomes possible and deadlock is prevented. A formal proof of the absence of deadlock can be found in [CHAN 79c]. Although the null message method guarantees that no deadlock will be caused by the asynchronous simulation technique, deadlocks in the actual system will still be reflected in the simulation.

There are several drawbacks of the null message method as described above. One or more null messages are always generated whenever a real or a null message leaves a node with more than one outgoing arc. However, these null messages are never consumed or destroyed except at sinks in the model. Worst of all, if there is a feedback path from node C to node E in Figure 3, new null messages will be generated in an unstable manner. As a result, the simulation is dominated by processing the null message overhead. Moreover, if there are multiple classes of jobs which require different service times, the null message method becomes even more complicated.

4.1.2 The Virtual Ring Method

In conventional event-driven simulation, the event which has the smallest simulation time can never be preempted. The same concept may be applied to asynchronous distributed simulation to resolve deadlocks. Let M be the message which has the smallest simulation time among all available incoming messages to a blocked node N. If the simulation time of M is smaller than that of any other node in the network, clearly, no node will be able to send a message to preempt the message M. Therefore, even though N is blocked, this message may be consumed by N without the risk of violating the correctness of the simulation.

The requirement that M has a smaller simulation time than any other node in the network is a sufficient but not necessary condition for the absence of preemption. In other words, this method is expected to be not the most efficient strategy. However, it has the significant advantage of being simple to determine. Tighter bounds may be obtainable but probably not without a large amount of overhead. If there are only very few messages in the network, many nodes will become blocked, and the simulation will be slowed down because it is time consuming to "unblock" these nodes.

The remaining problem is to find an efficient method to determine the node (other than the destination of the message) which has the smallest (earliest) simulation time in the entire system. In other words, it is desired to find the node which has the second smallest simulation time because N, the destination node of message M, always has a smaller simulation time than that of M.

In the null message method, additional null messages traveling along the arcs in the original network are used to resolve the deadlock problem. To determine the smallest simulation time among the nodes, it is easier to use additional data paths. A simple approach could employ a centralized controller which collects the current simulation time from all nodes and determines the smallest two. Again, a centralized controller may become a bottleneck when the number of processors is large.

A second method of determining the smallest simulation time among nodes is to map the processors on a virtual ring in addition to the arcs in the network. A single special message which contains the simulation times of each node and the identifications of the two nodes which have the smallest simulation time is sent around the ring. The length of this message is proportional to the number of processors in the network. When a node receives this message, it updates its simulation time in the message as well as the identifications of the two nodes which have the smallest simulation time. If the node is blocked, it also checks the simulation time of its message M to determine whether it may be consumed or not.

4.2 Collection of Statistics

Since a "snap shot" in an asynchronous distributed simulation is meaningless, the collection of statistics becomes difficult; for example, it is inappropriate to "measure" the actual length of a queue simply by counting the number of customers in it, as is shown by the following illustration.

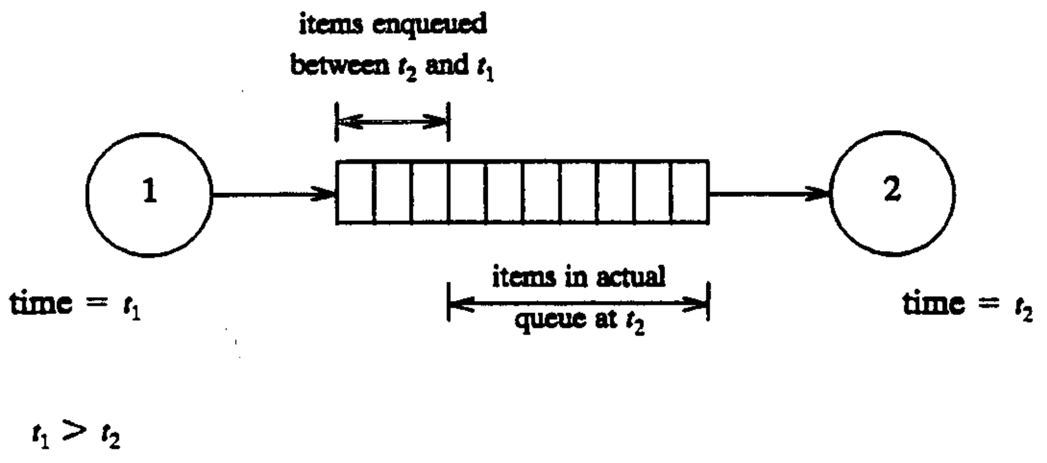

Figure 4: Queue Length in Asynchronous distributed simulation, $t_1 > t_2$

In Figure 4, there is a queue which connects service centers 1 and 2. Customers joint the queue at center 1 and leave at center 2. If the simulation time at service center 1 (t_1) is greater than that of center 2 (t_2), the simulated queue tends to contain more customers than the corresponding queue in the actual system.

The second case (Figure 5) is $t_2 > t_1$. The effect of this situation is just the opposite; i.e., the queue being simulated tends to be shorter than the actual queue.

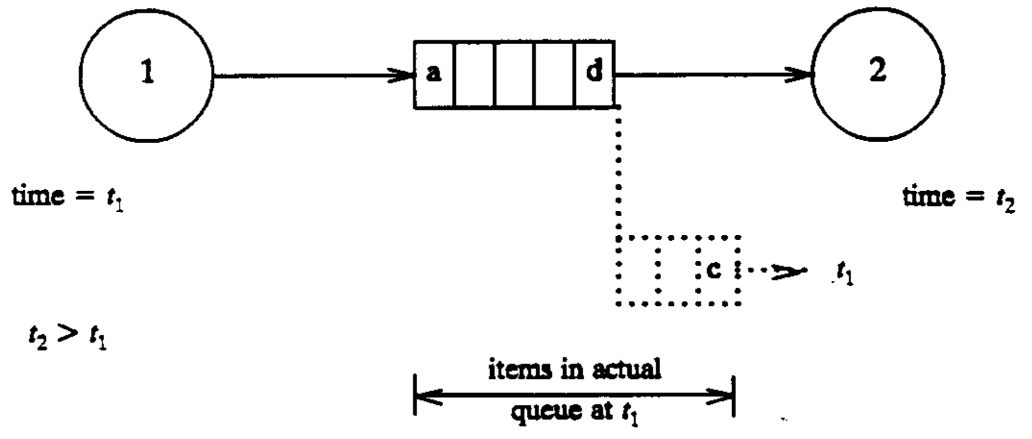

Figure 5: Queue length in Asynchronous Distributed Simulation, $t_2 > t_1$

Since it is impossible to predict the arrivals between t_1 and t_2 before service center 1 has been advanced to t_2, a simple way to keep track of the queue length is to save the customers which have departed at center 2. That is, it is necessary to extend the tail end of the queue. When a customer departs at a time $< t_1$, the customer must be "duplicated" in the simulation and a copy remains in the extended part of the queue. It remains there until t_1 has been advanced to a time greater than the departure time of that message. The length of the actual queue at t_1 is simply the sum of the lengths of the simulated queue and the extended part.

Since the queue in the simulation tends to be longer than or at least equal to the corresponding queue in the actual system, it is necessary to reserve more memory space in the simulator than the actual queue length or the queue in the simulation will become full prematurely. The farther apart t_1 and t_2 may become, the more extra room must be reserved.

5. The Processing Environment at UCLA

The Center for Experimental Computer Science, a support unit attached to the UCLA Computer Science Department, maintains a system of 20 VAX computers interconnected by Ethernet. The operating system on these machines is LOCUS, a network-transparent extension of UNIX.

In LOCUS, as in UNIX, a process can create *pipes,* which are unidirectional communication links between two processes. After pipes have been set up, a new process may be *forked,* and the parent and son processes can communicate with each other. This procedure may be repeated so that a collection of processes with arbitrary pipe connections may be created. Finally, under LOCUS, these processes may be migrated to different sites (different machines in the network) so that a real distributed processing environment is created. In our simulation application, such processes and pipes may be considered as the analogs of nodes and arcs in a graph model.

6. Experimental Work

We have carried out some preliminary experiments on the computing facilities described above. The model used is a simple slotted ring architecture for communication, incorporating some of the basic features of the Doelz network, and extensible to incorporate further features in future experiments.

Keeping the dimensionality (and hence the complexity) low initially, we chose to implement for demonstration purposes a ring of four nodes and siding queues as shown in Figure 6. Only the siding queue of node B is shown in the figure; the others are identical.

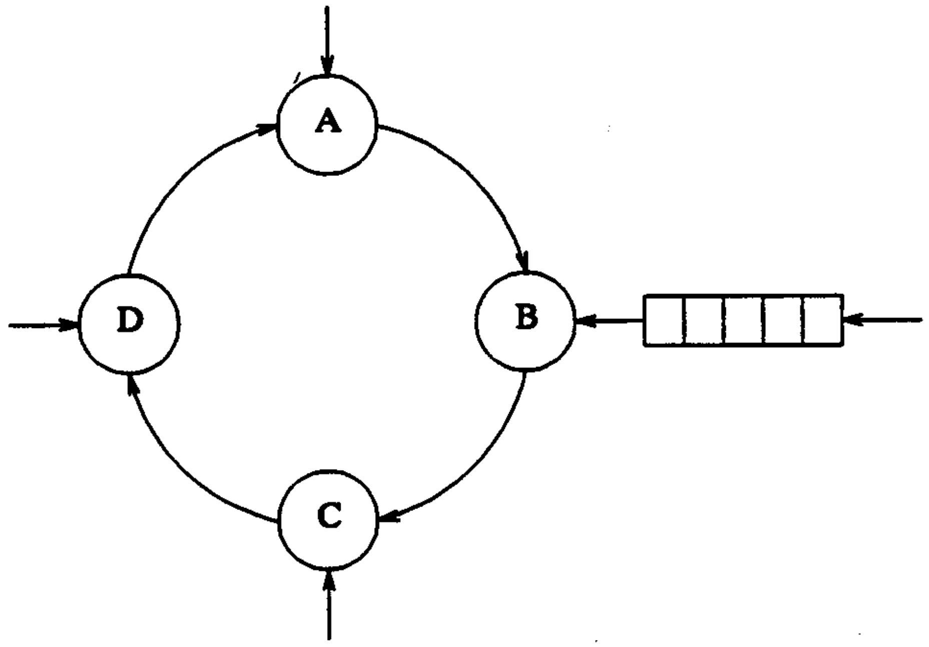

Figure 6: A Four-node Ring Network

As in the Doelz network, the nodes are connected conceptually by a conveyor belt representing the slotted ring (time slots in the actual system are defined through specification of the fixed length of a basic message packet and by the controlling clock rate). The distance between two adjacent nodes is measured by the number of slots between them. At each node, new messages are generated for the simulation with exponential inter-arrival times. The simulated destination of a new message is selected randomly among the other three nodes. The simulated network provides priority to messages already on the ring. When a message arrives from the neighboring node and needs to be forwarded to the next node, it is always transmitted to the next node even if a new message may be waiting at the siding queue. A new message is accepted only if an empty slot is received or if the incoming message leaves the network at the present node. Processes representing nodes may, under LOCUS, be migrated to different processors to investigate the effects of distributed processing.*

In these simulation experiments, the basic feasibility of distributed simulation using LOCUS (or a similar distributed operating system) was confirmed qualitatively, and a number of issues to be addressed were uncovered as anticipated. These issues are largely connected with problems of time-asynchronous distributed simulation; hence we have concentrated upon this area in the discussion of the preceding sections.

There are related issues arising more specifically from characteristics of the operating system being used. In particular, it should be pointed out that in UNIX and in LOCUS, pipes are designed for one-way data transfer from one process to another, and are not intended for bi-directional communication between two processes, although it is possible to create two pipes, one for each direction, between two processes. At the input end of a pipe, the source process can use *write* system calls to send data to the pipe. At the other end, the destination process can use *read* system calls to remove data from the pipe. The current implementation of LOCUS allows a maximum of 4096 bytes of data to be stored in a pipe. If a process writes to a full pipe, it will wait until a *read* appears at the other end so that there is room in the pipe to complete the *write*. Similarly, when a *read* is attempted to an empty pipe, it will also wait until something has been written at the other end. These properties open the possibility of another type of deadlock when bi-directional communication is implemented.

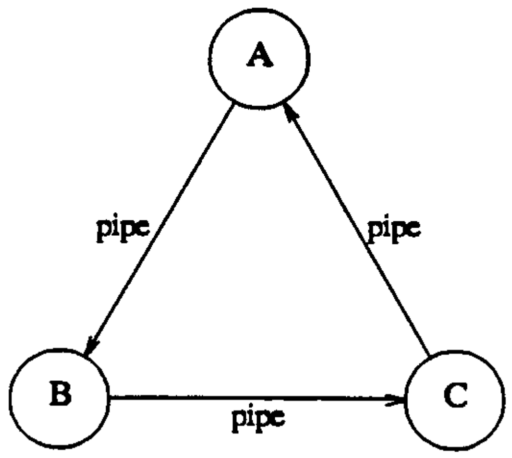

Figure 7: A Three-Node Network Connected by Pipes

*Software modules needed in implementing this simulation were written in C, the primary high-level language support under UNIX and LOCUS, from which calls to the operating system are most easily made.

To illustrate this possibility, consider three processes A, B and C connected together by three pipes as in Figure 7. If all three processes happen to be attempting reads and all three pipes are empty, the processes will be waiting forever because nothing can be written to any one of the pipes to break the deadlock. A similar situation occurs if all three processes are trying to write when the pipes are full.

In continuing our experimental work, we must take such phenomena into account in the system design and in the modeling techniques to be developed. Of course, such a software-system-related deadlock problem is not unique to LOCUS, but is encountered frequently in distributed systems implementations generally.

7. Conclusions and Future Directions

Some of the major problems of distributed simulation, such as preemptions, deadlocks, collection of statistics, and the need for model simplification, have been identified and addressed within the framework of the intended application to data communication networks with slotted-ring architecture, and experimental implementation has been initiated.

In the next phase of the project, we plan to pursue further details concerning simplification of the simulation models and to expand our experimental work to test some synchronization methods; it may be necessary to combine several methods to resolve the synchronization problem before proceeding to implement application models of greater complexity.

8. References

[BALA 84]	Balaban, P., K.S. Shanmugan and B.W. Stuck, "Computer-Aided Modeling, Analysis, and Design of Communication Systems: Introduction and Issue Overview," *IEEE Journal on Selected Areas in Communications*, Vol. SAC-2, No. 1, pp. 1-7, January, 1984.
[BRYA 77]	Bryant, R.E., "Simulation of Packet Communication Architecture Computer Systems," MIT/LCS/TR-188, MIT, November, 1977.
[CHAN 78]	Chandy, K.M. and J. Misra, "A Nontrivial Example of Concurrent processing: Distributed Simulation," TR-82, Department of Computer Science, University of Taxes in Austin, September 1978.
[CHAN 79a]	Chandy, K.M., V. Holmes and J. Misra, "Distribution Simulation of Networks," *Computer Networks*, pp. 105-113, March, 1979.
[CHAN 79b]	Chandy, K.M. and J. Misra, "Deadlock Absence Proofs for Networks of Communication Process," TR-105, University of Taxes at Austin, June 1979.
[CHAN 79c]	Chandy, K.M. and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transections on Software Engineering*, Vol. SE-5, No. 5, pp. 440-452, September, 1979.
[CHAN 81]	Chandy, K.M. and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Comm. ACM*, Vol. 24, No. 4, pp. 198-206, April, 1981.
[CHAN 80]	Chandy, K.M. and J. Misra, "Termination Detection of Diffusing Computations in Communication Sequential Process," TR-144, Department of Computer Science, University of Taxes at Austin, April, 1980.

[CONC 82] Conception, A.I. and B.P. Zeigler, "Distributed Simulation of Distributed System Models," SC-82-016, Department of Computer Science, Wayne State University, Detroit, MI, 1982.

[DOEL 84] Doelz, M.L. and R.L. Sharma, "Extended Slotted Ring Architecture for a Fully Shared and Integrated Communication Network," To be presented at the MIDCON 1984 Conference to be held in Dallas on September 12, 1984.

[LAVE 83] Lavenberg, S.S., ed., *Computer Performance Modeling Handbook,* Academic Press, 1983.

[LOCU 84] Locus Computing Corporation, "Locus Distributed Unix: A Comparison with Unix," May 1984.

[MARI 79] Marie, R.A., "An Approximate Analytical Method for General Queueing Networks," *IEEE Transection on Software Engineering,* Vol. SE-5, No. 5, September, 1979.

[OREI 84] O'Reilly, P.J.P. and J.H. Hammond, Jr., "An Efficient Simulation Technique for Performance Studies of CSMA/CD Local Networks," *IEEE Journal on Selected Areas in Communications,* Vol. SAC-2, No. 1, pp. 238-249, January, 1984.

[PEAC 79] Peacock, J.K., J.W. Wong and E.G. Manning, "Distributed Simulation Using a Network of Processors," *Computer Networks,* Vol. 3, No. 1, pp. 44-56, February, 1979.

[PEAC 80] Peacock, J.K., J.W. Wong and E.G. Manning, "Synchronization of Distributed Simulation Using Broadcast Algorithms," *Computer Networks,* Vol. 4, pp. 3-10, 1980.

[SAUE 81] Sauer, C.H. and K.M. Chandy, *Computer Systems Performance Modeling,* Prentice Hall, 1981.

[SAUE 83] Sauer, C.H. and E.A. MacNair, *Simulation of Computer Communication Systems,* Prentice Hall, 1983.