DISTRIBUTED COMPUTER SIMULATION OF DATA COMMUNICATION NETWORKS: PROJECT REPORT 2

Shun Cheung
Jack W. Carlyler
Walter J. Karplus

March 1985
CSD-850013

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 second in a series of reports on continuing studies of discrete-event simulation using distributed processing (e.g., on networks of microcomputer workstations), with application to performance prediction for data communication networks. Issues examined include time asynchronism, task allocation, and specification of a benchmark application model based on an extended slotted ring architecture for data communication.

Table of Contents

1. Introduction

This is the second report 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.

The DCSDCN project is broadly directed toward understanding and implementing discrete-event systems simulation via distributed processing in networks of computers, with specific application to the simulation of data communication networks, using as a test case the Doelz network architecture [Doel84] (an extended slotted ring for communication). A long-range objective of the DCSDCN project is to develop simulation tools, executable via distributed processing on microcomputer workstations, usable in field-engineering situations where performance of proposed data communication network installations must be estimated.

It is well known that discrete event simulation is in general very computer-time consuming. For example, using a currently available queueing-system simulation software package, a detailed simulation of a simple model of the Doelz network can run for hours or even days on a substantial minicomputer (without time-sharing) before the steady state is reached [Belg84]. In many applications, due to time, computation power, or equipment constraints, conventional discrete event simulation using a single machine is not feasible (as this example suggests). One solution is to simplify the model description to reduce the amount of computation in the simulation while compromising the accuracy of the results. Networks of microcomputer workstations are rapidly emerging as a medium-scale computing resource, so it is reasonable to consider more detailed simulations in such a distributed environment. We do not seek "optimality" when solving the application problem in this environment (since use of a single processor with unlimited resource may be more efficient, at least in execution time), but rather practicality in medium-scale simulations and extensibility to larger problems when more processors are available.

There are currently two different approaches to distributed simulation: partitioning the model [Brya77, Peac79, Chan79b, Chan81] and partitioning the functions; i.e., assigning functions such as random number generation, statistic collection, event processing, etc. to different processors [Shep85]. The latter approach by itself has the advantage of not involving the synchronization problem, but it can only exploit a limited amount of concurrency because the functions in simulation usually can be classified into just a few types in this

manner, and is therefore not considered in our work; we have concentrated upon applying the model-partitioning concept. However, it should be kept in mind that a combination of these two approaches can provide significant speed up if a sufficient number of processors is available. Furthermore, if the parameters in a portion of the model are rarely modified from one simulation to another, it is a waste of computation power to simulate it over and over in different runs. An approach called hierarchical modeling/simulation, which first studies the behaviors of different portions of the model separately and then uses their aggregate characters to approximate the collective behaviors of the entire model, is more appropriate in this case.

Before distributed simulation can be implemented, a number of issues introduced by the processing environment must be resolved, such as:

a) Selection or design of languages or packages for simulation software development; should have facilities for exploiting concurrency, interprocessor communication, and interaction with the (distributed) operating system.

b) Model partitioning and task allocation (among the processors), to exploit concurrency and improve execution time.

c) Avoidance of inconsistencies and deadlock caused by time-asynchronous processing of the tasks.

d) Design of methods for the collection of statistics (results of the simulation) generated in the distributed environment.

In addition, the application area (simulation of a data communication network) should be scrutinized with a view toward

e) Model simplification, e.g., reduction of the complexity of a queueing network through replacement by simple "summary" models or approximations, hence reducing the computation time in a simulation of the network, possibly at the expense of accuracy, while maintaining essential features.

When these issues are resolved, at least tentatively, it is possible to begin

f) Experimentation, developing software to implement simplified modules, to test feasibility.

The results of these studies and experiments should lead to

g) Selection of a benchmark problem, i.e., a model of intermediate complexity and realistic connections to the full-scale application -- a data-communication ring network in our case -- which can then be used as a vehicle for substantial software systems development and verification.

In our first report [Cheu84], issues (a) through (e) were introduced and discussed briefly, and the results of preliminary experimentation (f) were mentioned. In the present report, we discuss issues (b), (c), (d), (e), and (g), with emphasis upon dealing with the fundamental problems of synchronization (c) and specifying a benchmark model (g).

At UCLA, computing resources available for research use are centered upon a local-area network of 20 minicomputers with a UCLA-developed distributed operating system, LOCUS [Locu84], whose facilities for distributed processing are of particular interest to our project. This environment is being enlarged in several directions, with microcomputer workstations a planned feature. Our initial experiments in simulation software systems development have therefore been carried out on the minicomputer network, since the results will be transportable to workstations in the next phase of the DCSDCN project. Our initial approach has been to develop software modules in C, an existing general-purpose programming language, and to implement the distributed features of the overall simulation software package by exploiting the facilities of LOCUS.

2. Model Simplification

Complex computer systems and data communication networks are in general difficult to study through mathematical analysis or simulation. Their models, often based on network of queues, may be mathematically intractable so that only approximate solutions are possible. Computer simulation may be too time consuming to be practical. Model simplification becomes an important issue under these conditions. Three approaches to simplify a model can be identified, as follows.

An easily comprehended direct method to simplify a queueing network model is to replace a subnetwork by a flow equivalent service center (FESC) [Chan75]. The subnetwork to be replaced should have one output arc and one input arc connecting it to the main network. Under these conditions, the subnetwork can be analyzed separately by imagining the output to be connected

to the input. The throughput at this feedback path can be determined for various job populations in the subnetwork. A FESC is then defined to have a mean service time which is a function of the number of customers in the subnetwork and is equal to the reciprocal of the throughput. If the subnetwork itself has a product form solution, the FESC will have the same behavior as that of the subnetwork. Otherwise, the parameters of the flow equivalent service center may only be an approximation. The amount of error involved will depend on "how much" the subnetwork deviates from having a product form solution.

While the FESC concept is useful, it should be kept in mind that if the subnetwork cannot be isolated in a way such that it has only one input and one output, this approach is not applicable. Moreover, if the subnetwork model is far from having a product form solution, the FESC parameters obtained by the above method may be completely unrealistic. When no exact solution is available, there are methods to estimate the parameters of the flow equivalent service center; this involves the service time distributions, usually the mean and second moment, and the queueing discipline [Chan78].

A more general method (especially when solutions dependent on analytical modeling are not possible) is to simulate the subnetwork repeatedly for varying numbers of jobs. Once the parameters are obtained from this preliminary simulation, the entire subnetwork can be approximated by one service center in future simulations. A problem with this approach is that if there are many classes of jobs or many different priorities, there may be many states for the subnetwork. The parameters for each one must be determined, unless it is known in advance precisely which states will appear in the final simulation (which will rarely be the case). As a result, generally speaking, characteristics of the subnetwork must be studied (in advance) much more thoroughly than might actually be needed when overall system simulations are carried out.

A third approach separates the main model into two portions: a small part whose behavior needs to be investigated thoroughly so that its details must be included in the model, and the remainder of the system which can be represented by a very simplified model just sufficient to maintain interactions with the detailed part.

Our benchmark communication network model (see Section 11) is sufficiently complex so that simplification techniques, while not mandatory, become desirable and are feasible to carry our as tests of the method.

3. Asynchronous Distributed Simulation

Time-asynchronous distributed simulation has the potential to exploit the inherent concurrency in discrete event models. A general discussion on this point appeared in our first report [Cheu84]; here we summarize and extend the discussion to encompass additional approaches. In asynchronous distributed simulation, a convenient way to specify a system is to represent the model as a collection of objects. (Objects in distributed simulation are similar to service centers in a queueing network.) The interactions among these objects are maintained by inter-object messages. Most of these messages are event messages which signal the completion of events in the simulation. Others are control messages needed to maintain the correctness of the simulation. The number of objects in a realistic model is in general (much) greater than the number of processors. It is therefore necessary to map several objects to each processor in a way to minimize the time needed to perform the simulation.

Every object and every message must have a time stamp which is necessary to synchronize the events. The time stamp of an object is the simulation time when the most recent event in that object terminated. Usually, a message is sent at that point and its time stamp will have the same value as that of the object. The time stamp of a message should be approximately equal to the time when the corresponding message is generated in the actual system. However, since the objects are not synchronized, some of them will be ahead in simulation time. Therefore the order of arrival among messages from different inputs to an object is not always the same as that in the actual system. This creates the message synchronization problem in asynchronous distributed simulation.

A simple example will illustrate how the problem can arise. In Figure 1, assume message X arrives object C before message Y in the simulation; i.e., in real time, but Y might have a smaller time stamp than X because D is trailing object B. This situation is called a preemption. If C simply executes in a data-driven fashion, it faces the risk of processing the events in the wrong order and eventually produces incorrect simulation results.

Normally, the time stamp of an object as well as those of its output messages will increase monotonically. Therefore, in asynchronous distributed simulation, direct preemptions are caused only by messages arriving from different sources to a node. In the special case where no nodes have more than one input arc, the synchronization problem disappears. Unfortunately, this special case only includes nodes connected as a tree or as a loop with arcs pointing in one direction only. (In addition to that, except for the one at the

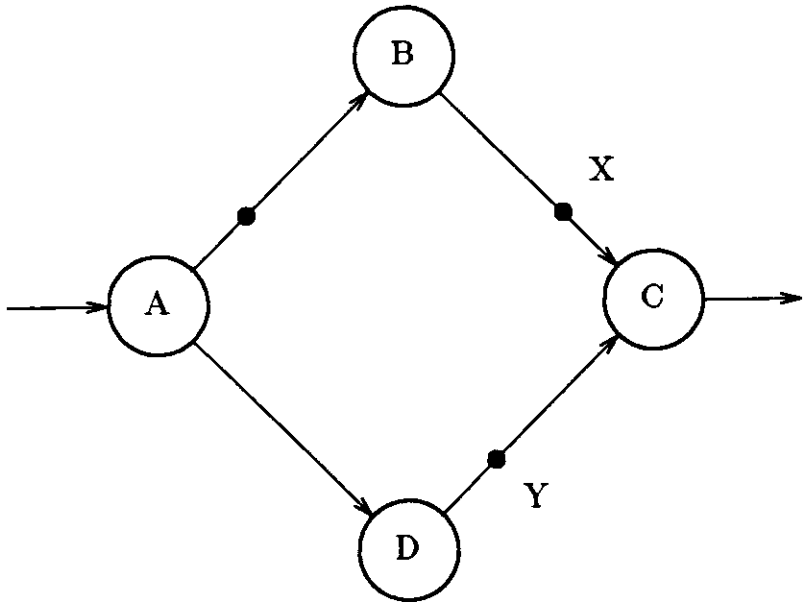

Figure 1: A Simple Graph with Alternative Paths

root of the tree, the objects should not accept any external inputs.) Once an object is vulnerable to preemption, it may send messages with out-of-order time stamps, and all objects which will directly or indirectly receive messages from it may subsequently be preempted. Clearly, these requirements are very restrictive. For example, two-way interactions among a pair of objects prohibit the tree connection. Therefore, the synchronization problem cannot be avoided in our work except in some special cases.

4. Conservative Synchronization Techniques

The following requirements guarantee that input messages will be processed in the right time stamp order and hence the correctness of the simulation:

1. The order of the messages going from one object to another through an arc must be the same as that of the corresponding messages in the actual system. That is, a message must have a larger time stamp value than its predecessor. (Messages coming from different arcs, however, may arrive in different order.)

2. There must be at least one message in each incoming arc to an object so that the most imminent message can be identified and then processed. Otherwise, the object will be considered blocked and none of the messages may be consumed.

Although these two requirements are sufficient to guarantee the absence of preemption, they can cause deadlocks in the simulation. For example, in Figure 1, if there were no messages sent to object D from C, message X would be waiting forever, and a deadlock occurs. One way to avoid deadlocks is by sending "null messages." Whenever object A sends a message to B in the example, it should also send object C a null message which carries no information but has the same time stamp as the real message sent to B. This is necessary even though the message sent to B is null itself. It has been proved that deadlocks can never occur under this condition because a null message will always be passed to object D from C for each real message from B to C [Chan79a]. Synchronization techniques based on these observations may be called "conservative," in the sense that they wait until it is "safe" to proceed.

5. Problems with the Conservative Synchronization Methods

The major problem with the null message method is that one or more null messages will be generated whenever a message (real or null) leaves a node with two or more output arcs. If there are feedback loops in the model, the number of null messages can grow very quickly, leading to a significant amount of processing overhead. Since the main function of null messages is to provide a lower bound for the next real message arriving from an arc, all null messages which are waiting in a queue but not the last element of that queue may be removed. The correctness of the simulation is not affected because the last message, which has the largest (latest) time stamp, always defines the highest lower bound. The greater the number of null messages in a simulation run, the more likely that there are several of them waiting in the queues. Therefore the number of null messages in the system may be expected to be self-regulating. Although they will not grow in an unstable manner, null messages certainly introduce a significant amount of overhead. Empirical results [Seet79] have indicated that this approach to deadlock prevention is costly.

To avoid this, Chandy and Misra [Chan81] have suggested letting the system run into deadlock and then recovering from it via a special deadlock detection algorithm using the Dijkstra-Scholtem scheme [Chan81], executed in parallel with the simulation. When a deadlock appears, the system should find the event which has the earliest simulation time. Again, since this event can

never be preempted, it may be executed without hesitation to break the deadlock. The problem with this approach is that the system can, and probably will, get into deadlock frequently. Moreover, once a deadlock is detected and resolved, the system may remain in a "near-deadlock" state. As a result, the amount of overhead involved to detect and recover from deadlocks may be large. It is not clear that the concurrency gained by parallel simulation is justified in the presence of such potential overhead.

The Virtual Ring method described in our ealier report [Cheu84] uses a special message which goes around a ring and visits each object in turn to determine the smallest (earliest) simulation time (SST) in the entire system. This method has the advantage of requiring very little overhead. However, if there is a large number of objects in the model, it will take a relatively long amount of time for this message to go around the ring once. As a result, the value of SST will not be updated very quickly. When a significant portion of the nodes are blocked, the updating of SST will become a limitation which prevents the system from exploiting its inherent concurrency. Moreover, this method is also subject to problems similar to those in the Chandy and Misra method outlined above.

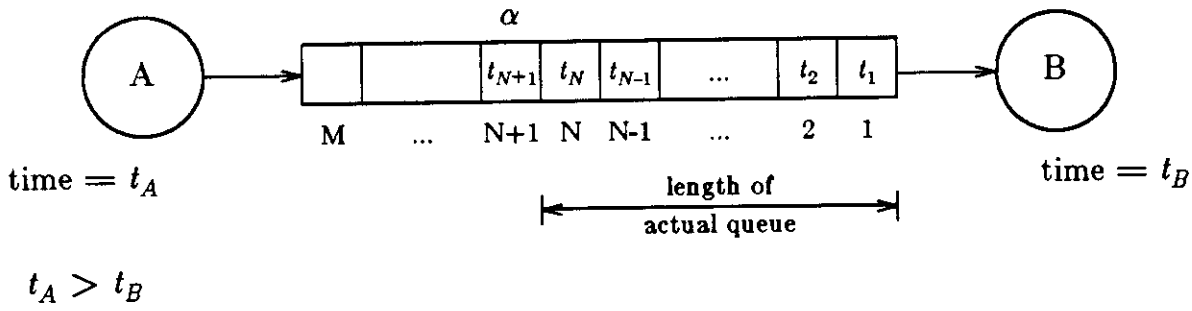

Figure 2: Queue Length in Asynchronous Distributed Simulation, $t_A > t_B$

In asynchronous distributed simulation, the length of a queue may be very different from that in the actual system since the simulation time at some of the objects may be far ahead than those of the others [Cheu84]. For example, in Figure 2, assume the length of the actual queue between objects A and B is N. If the time stamp of object A is ahead of that of B; i.e., $t_A > t_B$, there may be more than N items in the queue but the correctness of the simulation is not necessarily violated. Assume the size of the queue in the simulator is M, M > N, and the time stamp of the item at the i^{th} position is t_i; i.e., it joints the queue at simulation time t_i. Therefore, we need to make sure that $t_{N+1} > t_B$. If $t_{N+1} < t_B$ at any point in the simulation, it indicates that there are more than N items in the queue of the real system (at the corresponding time) and an overflow has occurred. Unfortunately, if the queue length in simulation is greater than N+1 when object A sends a message α to B, A cannot predict whether t_B will be greater than t_α when α becomes the $N+1^{st}$ item in the queue. In order to guarantee the correctness of the simulation, the length of the queue in the simulation must be limited to N although some potential concurrency is sacrificed.

6. The Time Warp Method

The Time Warp method introduced by Jefferson and Sowizral [Jeff83a, Jeff83b, Jeff85] is based on a completely different philosophy from the methods discussed above. It is appropriate to call those methods "conservative" because they guarantee the correctness of the simulation at the cost of possibly leaving the processors unnecessarily idle and entering deadlock states. In contrast, the Time Warp philosophy is "liberal" in the sense that simulation is carried forward, even in the presence of possible inconsistencies, then retracted when inconsistencies are actually detected.

In the Time Warp model, every object has just one input queue which receives messages from several other objects. There is no requirement that the messages from one object to another be in any particular order. Moreover, no objects are ever considered blocked. When a processor becomes free, it always accepts the message which has the smallest time stamp among the input queues of its objects at that point and does not consider the possibility that a message with a smaller time stamp may arrive in the future. It can be seen that Time Warp does not attempt to prevent preemptions. Nevertheless, as mentioned before, messages must be consumed in the correct order in discrete event simulations. To resolve this problem, Time Warp saves the old states of an object so that when a preemption occurs, the object will be able to *roll back* to a

previous state which has a time stamp smaller than or equal to that of the preempting message. Moreover, anti-messages are sent to cancel the messages which were sent during the rolled back period. If the previous message has already been consumed, an anti-message will cause the destination object to roll back as well. Otherwise, the messages will nullify each other. In this way, errors and side effects caused by the out-of-order message should be canceled.

7. Global Virtual Time (GVT)

Global Virtual Time in the Time Warp model is similar to the concept of simulation time in conventional discrete event simulation on a single processor, in which the event having the smallest time stamp cannot be affected by any event not yet simulated at that time. However, in a distributed processing environment, there are in-transit messages which may preempt their destination objects, so their time stamps need to be taken into account in the calculation of GVT. Therefore, Global Virtual Time is defined to be the minimum value among all time stamps of the objects and the in-transit messages; i.e., the time stamp of the most imminent event in the entire simulation. GVT is a non-decreasing function of real time. Assume that at real time t_{REAL}, the time stamp in object A has the same value as GVT (t_{REAL}). Object A cannot be preempted at this point because no messages, either in-transit or to be sent by any object in the future, can possibly have a time stamp less than GVT (t_{REAL}). Therefore, A may erase all of its saved (old) states. (However, A should either save its current state or keep the most recent saved state if not every state is preserved in order to reduce the memory requirement). For the same reason, the other objects may erase all but the last saved state which has a time stamp less than or equal to GVT (t_{REAL}).* Moreover, the calculated simulation result for simulation time up to the current GVT can be considered final. Thus, all outputs from the simulation should not be sent to the outside world until GVT has passed the value of the time stamp of the output, and in this sense, GVT is also a (conservative) measure of how fast the distributed simulation progresses.

*Theoretically, the system cannot roll back to a state prior to that at GVT (t_{REAL}). However, since some of the objects might not have saved the state at exactly GVT (t_{REAL}), a preemption at time GVT (t_{REAL}) + ϵ, $\epsilon > 0$, will cause a roll back to the last saved state before GVT (t_{REAL}).

8. Problems with the Time Warp Method

One of the main problems with Time Warp is its potential for large memory usage. A significant amount of memory is needed to save the previous states, especially when the state description is complicated. In fact, as the simulation progresses, any amount of memory could be used up unless some type of garbage collection is implemented. The concept of GVT is very useful towards this end.

Fortunately, it is not necessary to save every (previous) state of an object. A state save is needed only "once in a while" at certain *check points*. The tradeoff is that when a preemption occurs, the old state with the greatest time stamp which is less than or equal to that of the preempting message might not have been saved, causing the object to roll back further than necessary to reach the closest saved state and resume simulation.

Since Time Warp permits an object to "regret" what it did and return to a previous state, new problems which do not have counterparts in the conservative methods arise. For example, it is possible for the objects to enter a certain mode such that they simulate forward for a certain period of time, send anti-messages to one another, roll back to the starting point and the cycle repeats. Although the processors are busy all the time, no real progress is achieved and the system is actually in a deadlock situation. It has been proved, however, that if it always takes a positive (non zero) amount of simulation time to serve an event, this type of deadlock can never occur [Sama85]. (The restriction of positive *step size* creates a problem because several common modeling distributions, such as the exponential, include the value zero.)

9. Modifications of Time Warp

In our project, several alternatives are being considered for the application of Time Warp ideas. At present, three issues have been identified and are being studied:

1. The measurements we wish to obtain from simulation, when applied to our data communication network (DCN) models, are primarily response times and queue length distributions. Response time includes two factors: the elapsed times between the issuance of a command (as a DCN input) and the receipt of (i) the beginning of the response and (ii) the end of the response. Recall that the basic Time Warp model supposes only one

input queue for each object although there may be several input arcs; this merging of the inputs hides the individual behaviors of the queues and is therefore inappropriate for our particular application. Thus it is necessary to consider either separation of queues or retention of a status record for each queue, as a modification of the basic method.

2. The current algorithms for GVT calculation require the broadcast of messages from the initiating processor [Sama85] at each update of GVT. Global broadcasting is undesirable, particularly when the number of processors is large. Alternative methods are needed to perform GVT calculations in a more distributed or hierarchical manner.

3. Combination of the basic Time Warp concept with other mechanisms may result in improvements in implementations. For example, it may be possible to introduce aspects of the conservative method so as to reduce memory requirements and processing overhead in pure Time Warp.

10. Dynamic (Re)allocation and Statistical Analysis of the Simulation Results

One of the main objectives of distributed processing in general is to reduce the time required by a simulation run. Since the amount of overhead is usually sensitive to the manner in which objects are assigned to processors, job allocation and load balancing has always been a major problem for multi-processing and distributed computation. A good allocation method for Time Warp should take into account the following major sources of overhead:

1. Roll backs

2. Inter-processor communications

3. The saving of object states

4. GVT calculation

Compared to the conservative methods, a roll back in the Time Warp model should not be considered as overhead, because it is the consequence of an object making a wrong guess and simulating forward without complete knowledge of incoming messages. The wasted simulation time would have been wasted in the conservative approach too, because the object would have been blocked. The only additional overhead in the Time Warp model is the

computation time needed to restore the appropriate saved state and to send anti-messages. However, in the absolute sense, a roll back clearly indicates processor time has been wasted. Therefore, it is important to reduce the number of preemptions in simulation. Since a preemption occurs when an object with small *local virtual time* (LVT)* sends a message to one with large LVT, it would be desirable to minimize the differences among the LVT values in the objects as well as the number of messages. Unfortunately, the number of messages among the objects is inherent in the model and cannot be controlled. Moreover, empirical results have suggested that the overhead needed to update GVT is insignificant when the number of processors is small [Sama85]. It can be concluded that an allocation algorithm should (i) allocate the objects which have much communication with one another to the same processor in order to reduce the communication overhead, (ii) balance the progress of the simulation times in the objects, and (iii) balance the loads among the processors.

Samadi [Sama85] has suggested several *static allocation* methods using the criteria outlined above. We are developing more complex heuristic methods for our specific application. However, the ultimate goal is to *(re)allocate dynamically*. The reason is that, in some models, not all of the parameters needed for static allocation are accurately known or even available beforehand. Moreover, the state of the simulation may vary due to change of input parameters and special events in the model. Clearly, it is desirable to consider an alternative allocation if the current one is introducing too much unnecessary overhead to the simulation. Some initial research in a similar area which has different requirements has been reported by Nicol and Reynolds [Nico85].

There are at least two major difficulties with dynamic allocation:

1. It is necessary to be able to predict, or at least estimate, the future behavior (remaining portion) of the simulation. Questions to be answered include: which processor will have a considerably heavier load than that of the others? Which one will have a LVT retarding the progress of the entire simulation? In addition, if the remaining part of the simulation is small, is it worthwhile to reallocate at all?

2. Once reallocation is assumed to be desirable, it is necessary to determine how to reallocate the objects to the processors so as to reduce the overall computation time.

There is potential for substantial overhead in monitoring the progress of the simulation, in order to determine the necessity of reallocation, as well as in

*When the application is asynchronous discrete event simulation, local virtual time in Time Warp is the same as the simulation time of an object.

actually reallocating the objects. Determining when a simulation run should terminate, based on accuracy assessments, is itself a difficult problem. This raises the basic question as to whether dynamic allocation is worthwhile at all. Experimentation as well as analysis is needed in this area; we are planning such experiments for the benchmark model described in the following section. The processing context should be taken into account in reaching conclusions; for example, in an extensible network of small processors, it may be cost-effective to accept a suboptimal implementation on a somewhat larger network.

11. The Benchmark Communication Network Model

Initially, a simple model of the Doelz network as shown in Figure 3 will be used as the benchmark model. The basic features of the slotted ring architecture of interest are present [Doel84]; we have briefly reviewed the conceptual framework in our previous report [Cheu84].

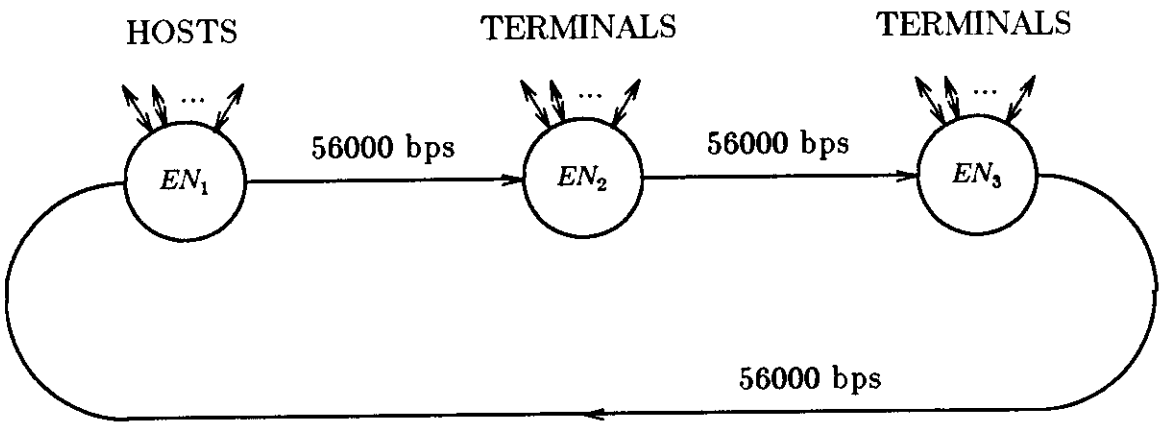

Figure 3: A Simple Benchmark Ring Network

Figure 3 shows an Extended Slotted Ring network containing three intelligent nodes, called Elite One nodes (EN) in the Doelz network architecture [Doel84]. EN_1 is connected to 16 hosts and EN_2 and EN_3 are each connected to eight terminals. Requests arrive from the terminals as Poisson processes. Initially, each request will be considered to have a common fixed length, e.g., four bytes;

the hosts are assumed to reply to each request by returning, say, a 300-byte response. Other initial parameters are as follows: The host "think time" is normally distributed with a mean equal to 1000 msec, the speeds of the terminals are fixed to 9600 bps, and the speed of the node-to-node communication links is 56000 bps. These parameters will later be varied to study network performance under different situations. This basic model can also be expanded, once the modeling and simulation techniques have been implemented, by adding more Elite One nodes and eventually additional rings hierarchically.

12. Conclusions and Upcoming Work

To summarize, processor synchronization is probably the most serious fundamental problem in asynchronous distributed discrete-event simulation. Conservative methods developed in the late '70s and early '80s have not been able to provide satisfactory solutions. The more recent Time Warp method seems to be much more promising towards this end, but it must be recognized that there are difficulties connected with memory requirements and the possibility of slow progress (due to frequent roll backs). Another fundamental point is that the performance of distributed simulation is very sensitive to the allocation of objects to the processors. The desire to (1) minimize inter-processor communication, (2) equalize the simulation time of the objects and (3) balance the load on the processors makes the allocation problem difficult (in fact, NP complete in the sense of complexity theory). Heuristic methods and combinations of approaches will be necessary in making progress to resolve this problem.

In this phase of the project, we have examined the Time Warp mechanism closely, with a view toward identification of features pertinent to out application, and we have specified a benchmark ring network model deemed suitable for experimental study, based upon our analysis of the issues and our feasibility experiments carried our on simpler models.

In the next phase of the project, we are planning to

1. Begin the design of a user-friendly input processor module to help the user create the model, within our context of data communication network simulation.

2. Develop heuristic methods which will statically assign objects to

processors once a correct model is available and the number of processors is known. Since the influences of the three factors mentioned above are not clear at this point, several heuristic methods which emphasize the solution with respect to different factors are being examined.

3. Simulate the benchmark ring network model, first on our distributed minicomputer network and then on a network of workstations, using some aspects of Time Warp for synchronization.

For the long range, we are also planning to investigate:

4. Dynamic allocation of the objects to the processors to improve the progress of the simulation.

5. Statistical analysis of the results of the distributed simulation.

6. Combinations of approximation and simulation to study the behaviors of large models.

References

[Belg84] Belghith, A., private communication, 1984.

[Brya77] Bryant, R.E., "Simulation of Packet Communication Architecture Computer Systems," MIT, Cambridge, Massachusetts, Tech. Rep. MIT/LCS/TR-188, November, 1977.

[Chan75] Chandy, K.M., U. Herzog, and L. Woo, "Parametric Analysis of Queueing Networks," *IBM Journal of Research and Development,* January, 1975.

[Chan78] Chandy, K.M. and C.H. Sauer, "Approximate Methods for Analyzing Queueing Network Models of Computer Systems," *Computing Surveys,* Vol. 10, No. 3, September, 1978, pp. 281-317.

[Chan79a] Chandy, K.M. and J. Misra, "Deadlock Absence Proofs for Networks of Communicating Processes," University of Texas at Austin, Austin, Texas, Tech. Rep. TR-105, June, 1979.

[Chan79b] Chandy, K.M., V. Holmes, and J. Misra, "Distributed Simulation of Networks," *Computer Networks,* March, 1979, pp. 105-113.

[Chan81] Chandy, K.M. and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Comm. ACM,* Vol. 24, No. 4, April, 1981, pp. 198-206.

[Cheu84] Cheung, S., J. W. Carlyle, and W. J. Karplus, "Distributed Computer Simulation of Data Communication Networks: Project Report 1," Department of Computer Science, University of California at Los Angeles, Los Angeles, California, Tech. Rep. CSD-840037, August, 1984.

[Doel84] Doelz, M.L. and R.L. Sharma, "Extended Slotted Ring Architecture for a Fully Shared and Integrated Communication Network," in *Proceedings of the MIDCON 1984 Conference,* Dallas, Texas: September 12, 1984.

[Jeff83a] Jefferson, D. and H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism Part I: Local Control," The Rand Corporation, Santa Monica, California, Tech. Rep. Rand Note N-1906AF, 1983.

[Jeff83b] Jefferson, D., "Virtual Time," in *Proceedings of the IEEE International Conference on Parallel Processing,* 1983.

[Jeff85] Jefferson, D. and H. Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism," in *Proceedings of the Conference on Distributed Simulation 1985,* San Diego, California: Society for Computer Simulation, 1985.

[Locu84] "Locus Distributed Unix: A Comparison with Unix," Locus Computing Corporation, May 1984.

[Nico85] Nicol, D.M. and P.F. Reynolds, "A Statistical Approach to Dynamic Partitioning," in *Proceedings of the Conference on Distributed Simulation 1985,* San Diego, California: Society for Computer Simulation, 1985.

[Peac79] Peacock, J.K., J.W. Wong, and E.G. Manning, "Distributed Simulation Using a Network of Processors," *Computer Networks,* Vol. 3, No. 1, February, 1979, pp. 44-56.

[Sama85] Samadi, B., *Distributed Simulation, Algorithms and Performance Analysis*: Ph.D. Dissertation, University of California, Los Angeles, February, 1985.

[Seet79] Seethalakshmi, M., *A Study and Analysis of Performance of Distributed Simulation*: Master thesis, University of Texas at Austin, May, 1979.

[Shep85] Sheppard, S., U. Chandrasekaran, and K. Murray, "Distributed Simulation Using Ada," in *Proceedings of the Conference on Distributed Simulation 1985,* San Diego, California: Society for Computer Simulation, 1985.