Distributed Simulation of Network Protocols*

F. Paterra, C. M. Overstreet, and K. Maly

April 16, 1990

Abstract

Simulations of high speed network protocols are very CPU intensive operations requiring very long runtimes. Very high speed network protocols (Gigabit/sec rates) require longer simulation runs in order to reach a steady state, while at the same time requiring additional CPU processing for each unit of time because of the data rates for the traffic being simulated. As protocol development proceeds and simulations provide insights into any problems associated with the protocol, the simulation model often must be changed to generate additional or finer statistical performance information. Iterating on this process is very time consuming due to the required runtimes for the simulation models. In this paper we present the results of our efforts to distribute a high speed ring network protocol, CSMA/RN[1].

1 Introduction

Computer simulations of real world entities can be computationally intense tasks taking many hours or even days to run. Simulation analysis of network topologies and protocols is of this type. Because communication media speeds are ever increasing, a need exists for protocols which can fully use the newly available bandwidth. Their development, however, relies on the use of computer-based simulations. One promising method for lowering the time

*This work was supported in part by CIT under grant INF-89-002-01, by NASA under grant NAG-1-908, and Sun Microsystems under RF596043.

One such protocol currently understudy at Old Dominion University is a gigabit ring network called CSMA/RN. This a fiberoptic, carrier sensed, multiple access protocol that operates nominally at 1 gigabit per second. Because of the amount of simulated traffic that must be processed before the network reaches a steady state is very high, long simulation times are required. In addition, because each second of simulation time accounts for a large amount of data, hence simulated events, each second of simulation time take longer to process.

2 Model Decomposition and Processor Synchronization Schemes

While we are interested in using distributed simulation as a tool for studying high performance networks, questions concerning the effective distribution of a simulation model must be addressed: decomposing the model into cooperating processes, and insuring that the results of the distributed simulation match those obtained in a single processor version.

Key to efficiently utilizing concurrent computing techniques is determining an effective decomposition of the model. Because the environment which is most commonly available is a number of losely coupled workstation computers, connected via a low speed network, the overhead of interprocessor communication can easily exceed any time gains resulting from the concurrent computation. This implies that a successful decomposition will attempt to minimize the amount of interprocessor communication.

Three basic methods for performing model decomposition have been identified as workable for certain problem domains: server decomposition, physical model decomposition, and arbitrary decomposition.

The server method requires that the developer identify support functions to the simulation, such as random number generation or events list maintenance, that can be isolated and placed on separate processors. In this scenario one processor is left to run the main simulation while other processors provide the support functions. [2] and [3] have used this method successfully to reduce simulation times by as much as 80%.

To allow further study of the server model, Old dominion University has developed a suite of tools, described in [4], that can be linked to simulations written in Simscript, C, or Pascal to provide the necessary communication and handshaking software for server model decompositions.

In contrast, physical model decomposition breaks the model into submodels corresponding to the components of the physical system being simulated. Different components share a single processor or may be execute on separate processors. This method results in a loosely coupled distribution which can, under some circumstance may reduce the communication requirements among processors. The drawback of only be able to utilizes many processors as there are concurrently operating components in the corresponding physical entity. This method seems easily applicable some communication network protocol simulations such as the CSMA/RN protocol described in [1].

The arbitrary method of model decomposition is to simply divide the model into as many parts as available processors, making the breaks where ever convenient from a programming point of view, such as programming modules. This results is a decomposition which may require more interprocessor communication, but could also allow greater concurrency during the run. There is little chance for reduction in simulation time with this method given the runtime environment. A better environment for simulations employing this decomposition would be a tightly coupled set of processors working with a pool of shared memory.

While each of these may provide some reduction in serial computer time, each only works well for some small problem domains and have not been shown to function well in the general case.

The second major problem faced by distributed simulation researchers is that of processor synchronization. Two methods are much discussed in the literature, the "conservative" and the "optimistic" methods. However, as with model decomposition, neither appears to work well for all problems.

Both the conservative and optimistic methods rely on developing a usable decomposition of the simulation model, and running each component on separate processors. The conservative method allows the parallel execution of events as long the events can be insured to be safe. A safe event is one who's inputs are fully defined and will not be directly or indirectly affected by the output of any other event. A more complete description of the conservative method synchronization can be found in [5].

The optimistic method, also called Time Warp, allows all processors to execute the events as soon as they are available, regardless of their relative logical times and the state of their inputs. As inappropriate timing sequences are detected, the simulation for the node with the inappropriate sequence is rolled back to a early time when the simulation was known to be correct and then restarted. In order to "roll back," the state of the computation on each processor must be saved at various points, so that that process state can be restored if that processor must be rolled back. Saving the states may incur more overhead than the benefits gained by the distributed computation. A good introduction to the optimistic or Time Warp method of synchronization can be found in [6].

3 Experiences

Four possible decompositions of the CSMA/RN model have been studied, all are described below.

3.1 Physical decomposition, one node per processor

The first attempt resulted in a decomposition of the events list based on node. Each node in the network was assigned to a physical computer and each computer maintained its own events list.

We found that this decomposition resulted in a I/O bound, very tightly coupled model that was executed in lock step form, with all processors waiting for a single resource, the ring media. Because CSMA/RN is a carrier sensed network, messages being transmitted may be interrupted if another messages passes the sending node. As the simulation runs, new messages enter the ring and can either 1) be successfully placed on the ring or 2) are interrupted by passing messages. This means that each node, before transmitting a message, must determine that the ring in front of the node is not occupied and must have knowledge of the next time a message will pass in front of the it in order to properly terminate the transmission (as complete or interrupted). In addition, because the network has a ring topology, any node can influence any other node, so global information about the state of the ring is required by all nodes before each transmission. Finally the amount of computation at each processor between interprocessor communication operations is very small therefore the processor spend most of their time waiting for I/O and comparatively little time performing the simulation. This decomposition resulted in longer runtimes than the single processor model.

3.2 Physical decomposition, one node per processor with replication

The second method studied also assigned a separate processor to each node in the network, however because global information is required by each node, all parts of the simulation that could effect the node was run on each local host.

This decomposition resulted in a very loosely coupled model that required virtually no interprocessor communication. Each processor was loaded with very computationally intensive code, but resulted in most of the simulation being replicated on each processor with corresponding replication of computations. In order to determine which operations, all operations of the simulation were independently studied for interoperation dependencies. The complete analysis can be found in the appendix, however, the collection of statistically data was identified as the only opertations who's execution did not require global information and therefore could be executed separately for each node in the network. If statistics collection consumes a large percentage of the runtime, then the amount of statistics time would be divided by the number of nodes in the network. Unfortunately, as shown in the analysis, statistics collection represents a very small percentage of the runtime so observed speedup would have been minimal.

3.3 Segmented ring

The third method studied was an attempt to increase the amount of processing on each physical processor and reduce the amount of interprocessor communication required. The network ring was to be segmented, as shown in the figure below, and placing a number of simulated nodes on each processor.

This method to suffers from the same some of the same problems as the first decomposition described. There is a high level of computation than in the first method, however, at the points in the simulated network where the ring must pass to a new physical processor, the execution be comes lock stepped again. The resulting simulation allowed one processor, representing a bank of connected nodes, to operate for the length of time equal to the simulated network propagation delay for the two nodes that reside on the beginning and end of the adjacent ring segments.

With this and the first decomposition studied, the problem of deadlock

Figure 1: Segmented Ring

had to be addressed. Each processor in cooperating in the simulation must communicated with the processor representing the nodes behind its nodes on the ring. Because this is a ring interprocessor breaks of the, there is no concept of beginning or ending to the network so the dependencies are circular. A scheme to insure that the simulation progress was developed, however because the the speedup would be limited, if observed at all, it was not fully detailed.

3.4 Server Decomposition

One observation made during our tests was that if data could be provided by one module to a second one without a time dependent cycle developing, the synchronization needs are much less. The major problem being addressed currently is the development of methods for model decomposition that reduce the amount of intermodule time based dependency.

Using static code analysis tools previously developed at Old Dominion University, we are performing data flow analysis of existing simulation models, written in the SIMSCRIPT, C, and Pascal languages, to determine the prevalence of code sections which either supply and/or consume time independent data objects during the simulation run. One major problem maybe the the relative costs of computing the data objects. Simple objects, such as random numbers, can be computed quickly, so the overhead of communication for can easily exceed ny benefits derived from computing the values on other machines. As discussed in [4], one solution to this problem may be to bunch the data objects and send them in quantity. This results in an inventory model with a reduction in communication overhead.

Additionally, we believe that code containing time dependent data cycles can be distributed if there is sufficient computation time between data requests to allow for the synchronization to occur or that the dependencies are not tight, one generation per synchronization, so that values can be precomputed and the simulation can be made to proceed.

4 Conclusion

Distributed simulation is a very hard problem [7]. Simulations of very tightly coupled systems such as network protocols that share a common resource have proven to be more difficult due to the amount of shared information that is required. Initial efforts in developing decompositions along physical lines has proven fruitless. In order for a distributed simulation to provide reductions in runtime, the modules must be designed so that the can perform compute intensive operations and require very little intermodule communication.

Currently we have simulation models for the FDDI[8], DQDB[9] (formally QPSX), and CSMA/RN[10] protocols available for analysis. To support the distribution of modules detected, we will use tools described in [4] to provide interprocessor communication and server model synchronization. These tools may need to be extended to allow for two way data flow and synchronization under a request and deliver scheme.

A Appendix – Analysis of CSMA/RN Replication Distribution

A.1 The data set

The data set was developed to show four scenarios when simulation the network; A message immediately sent, A message forced to wait, A message interruption, and a neighbor to neighbor transmission.

From	*To*	*Length*	*Time*
A	*B*	2	1
C	*B*	2	2
A	*C*	3	4
C	*B*	2	5
A	*B*	2	10
B	*C*	2	10

The network being simulated has three nodes, equally spread 1 time unit apart to form the ring

$$A \Longrightarrow B \Longrightarrow C \Longrightarrow A$$

A.2 Events and operations

Seven operations have been identified for the operations analysis of this algorithm.

1. A new message begins transmission
2. The start of a message passes through a node without causing an interruption
3. A message interrupts another
4. A message reaches its destination
5. A message completes transmission [1]
6. The end of a message is passed through a node
7. An interrupted message is restarted

Each of these operations can be further broken into suboperations to which execution times can be attached.

[1]That is the message has completely left the originator

1. A new message begins transmission
   (a) Copies of a **Start** event are placed on the *Actual* list for all nodes from the originator to the neighbor preceding the destination.
   (b) An **End** event is placed in the *Possible* list for the neighbor of the originator
   (c) The *Transmitting* array is updated to indicate that the originator is currently transmitting.
   (d) The delay for the **Start** is added to the packet delay
2. The start of a message passes through a node without causing an interruption
   (a) The **Start** event for the executing node is removed from the *Actual* list
   (b) The *Transmitting* array is updated to show that the node is no longer transmitting.
3. A message interrupts another
   (a) Fracture count is incremented
   (b) Copies of an **Interrupt** event are placed on the *Actual* list for all nodes from the originator to the preceding neighbor of the destination.
   (c) Remove the **Start** event, that caused the interruption at the current node, from the *Actual* list
   (d) Compute the number of bits transmitted
   (e) Compute the number of bits forced to wait
   (f) Add the number of bits transmitted to the count of bits delivered so far
   (g) Remove the interrupted message's **End** event from the *Possible* list
   (h) Place new **Start** and **End** events on the *Possible* list
4. A message reaches its destination
   (a) NO ACTION REQUIRED
5. A message completes transmission
   (a) The corresponding **End** event is moved from the *Possible* list to the *Actual* list and copies of it are posted for each of the nodes from the originator to the neighbor preceding the destination node

(b) The total delay for the message is added to the total message delay

(c) The number of bits in the last packet message is added to the total bits delivered

(d) The transmitting flag is reset

6. The end of a message is passes through a node

   (a) The End or Interrupt event is removed from the *Actual* list

   (b) The *Transmitting* array is updated

7. An interrupted message is restarted

   (a) Copies of a Start event are placed on the *Actual* list for all nodes from the originator to the neighbor preceding the destination.

   (b) An End event is placed in the *Possible* list for the neighbor of the originator

   (c) The *Transmitting* array is updated to indicate that the originator is currently transmitting.

   (d) The delay for the new Start is added to the packet delay

All of the subevents are weighted as taking 1 unit of execution time except subevents 1.a, 3.b,5.a, 7.a which take $(n/2)$ and 4.a taking 0 units[2].

A.3 The simulation

The following is a table of the operations executed and their ordering for the data set given in section A.1.

[2]n is the number of nodes in the network being simulated

This information can be gotten from section A.3.

Operation Class	Number Required	Units per Instance	Total for Class
1	6	5	30
2	3	2	6
3	1	9	9

Time	Operation/ Node	*Transmitting* Array A	B	C
1	1-A	1	0	0
2	1-C	1	0	1
2	4-B	1	0	1
3	5-A	0	0	1
3	2-A	1	0	1
4	4-B	1	0	1
4	5-C	1	0	0
4	6-A	0	0	0
4	1-A	1	0	0
5	1-C	1	0	1
5	2-B	1	1	1
6	4-C	1	1	1
6	3-A	1	1	1
7	6-B	1	0	1
7	4-B	1	0	1
7	5-C	1	0	0
8	6-A	0	0	0
8	7-A	1	0	0
9	5-A	0	0	0
9	2-B	0	1	0
10	4-C	0	1	0
10	6-B	0	0	0
10	1-A	1	0	0
10	1-B	1	1	0
11	4-B	1	1	0
11	4-C	1	1	0
12	5-A	0	1	0
12	5-B	0	0	0

[3] J. Comfort. The simulation of a microprocessor bassed event set processor. In *Proceedings of the Fourteenth Annual Simulation Symposium*, pages 17–33, 1981.

[4] F. Paterra, C.M. Overstreet, and K. Maly. Distributed simulation: no special tools required. 1990. Submitted to the 1990 Winter Simulation Conference.

[5] Jayadev Misra. Distributed discrete-event simulation. *ACM Computing Surveys*, 39–65, 1986.

[6] David Jefferson. Distributed simulation and the time warp operating system. *ACM SIGOPS*, 77–93, Nov. 1987.

[7] Peter A. Tinker and Jonathan R. Agre. *Object Creation, Messaging, and Stat Manipulation in an Object Oriented Time Warp System.* Society for Computer Simulation International, Mar. 1989.

[8] W. E. Burr. The fddi optical data link. *IEEE Communications*, 24(5):18–23, May 1986.

[9] R. M. Newman, Z. L. Budrikis, and J. L. Hullett. The qpsx man. *IEEE Communications*, 26(4):20–28, April1988.

[10] E. Foudriat, K. Maly, C. M. Overstreet, D. Game, S. Khanna, and F. Paterra. Csma/rn a protocol for high speed fiber optic networks. 1990. Submitted for publication WHERE? — ED IS THIS THE CORRECT TITLE??