PARALLEL DISCRETE EVENT SIMULATION: A SHARED MEMORY APPROACH

(Extended Abstract)

Daniel A. Reed
Department of Computer Science
University of Illinois
Urbana, Illinois 61801

Allen D. Malony
Center for Supercomputing Research and Development
University of Illinois
Urbana, Illinois 61801

Bradley D. McCredie
Department of Electrical and Computer Engineering
University of Illinois
Urbana, Illinois 61801

Introduction

The inherently sequential nature of event list manipulation limits the potential parallelism of standard simulation models. Although techniques for performing event list manipulation and event simulation in parallel have been suggested, large scale performance increases seem unlikely. Only by eliminating the event list, in its traditional form, can additional parallelism be obtained; this is the goal of distributed simulation.

Several distributed simulation techniques have been proposed. In the remainder of this abstract, we present the Chandy–Misra distributed simulation algorithm [ChMi81] and the results of an extensive study of its performance on a shared memory parallel processor when simulating queueing network models.

Distributed Simulation

Consider some *physical* system composed of independent, interacting entities. A natural, distributed simulation of the physical system creates a topologically equivalent system of *logical* nodes. Interactions between two physical nodes are modeled by exchange of timestamped messages. The timestamp is the simulated message arrival time at the receiving node.

Each logical node is subject to some constraints. First, node interaction is *only* via message exchange; there are no shared variables. Second, each node must maintain a clock, representing the local simulated time. Finally, the timestamps of the messages generated by each node must be non-decreasing.

This work was supported in part by NSF Grant Number DCR 84-17948 and NASA Contract Number NAG-1-613.

Intuitively, the distributed simulation has no single "correct" simulation time; each node operates independently subject only to those restrictions necessary to insure that events happen in the correct simulated order (i.e., *causality* is maintained). Independent events can be simulated in parallel even if they occur at different simulated times. For specificity's sake, we describe this simulation technique in the context of our RESQ implementation [SaMS80] for simulating queueing networks.

In the RESQ scheme, there are five node types: *service*, *fork*, *merge*, *source*, and *sink*. Service nodes correspond to the interacting entities of a physical system (e.g., servers in a queueing network). In contrast, fork and merge nodes exist only to provide routing. Finally, source and sink nodes respectively create and destroy network messages. Thus, the central server model [Buze73] of Figure 1a would be represented, using the RESQ scheme, as shown in Figure 1b.

Many performance studies of traditional simulation algorithms have been conducted, and, based on these studies, new event list algorithms have been proposed. Only limited *simulation* studies of distributed simulation have been reported [Seet78, JeSo85, Reed85]; little or no empirical data are available. In the remaining sections we discuss our experimental environment, implementation, and experimental results.

Experimental Environment

All simulation experiments were conducted on a Sequent Balance 21000 containing 20 processors and 16 Mbytes of memory. All processors are connected to a shared memory by a shared bus with a 80 Mbyte/s (maximum) transfer rate.

Parallel programs consist of a group of Unix processes that interact using a library of primitives for shared memory allocation and process synchronization. Shared memory is implemented by mapping a region of physical memory into the virtual address space of each process. Once mapped, shared memory can be allocated to specific variables as desired.

Distributed Simulation with Shared Memory

A shared memory multiprocessor, such as the Balance 21000, provides a flexible testbed for studying the performance

of distributed simulation. The problems associated with mapping a node network onto a network of processors are removed; the shared memory processors are, effectively, completely connected. By implementing message passing using shared memory, communications costs are the same for all processors. However, a shared memory implementation of distributed simulation requires special consideration for synchronization of shared message queues and processor allocation.

In a shared memory implementation of distributed simulation, all node state information, including input message queues, resides in shared memory. Message-based communication between nodes is implemented via shared access to the message queues of each node. Each message queue is protected by a synchronization lock to guarantee mutual exclusion.

There are two basic approaches to processor allocation in a shared memory implementation of distributed simulation. The first approach, *static node assignment*, fixes the assignment of nodes to processors for the duration of the simulation. The second approach, *dynamic node assignment*, assigns nodes to processors during the simulation. Idle processors obtain work from a shared queue of unassigned network nodes.

Simulation Experiments

Experimental evaluation of distributed simulation requires not only an implementation but also a set of test cases. This is particularly important in light of earlier simulation studies [Seet78, Reed85], which showed that the performance of distributed simulation is extremely sensitive to the topology of the simulated network. As tests, we selected several simple queueing networks and a few complex ones.

- tandem networks (1, 2, 4, 8, and 16 server nodes)
- general, feed-forward networks (6, 10, and 14 nodes),
- cyclic networks (2, 4, and 8 nodes)
- central server networks (5 nodes), and
- cluster networks (10 and 18 nodes).

Each of these networks was simulated for a variety of workloads, (e.g., routing probabilities, arrival rates, and service times) using six variations of a Chandy-Misra implementation: static node assignment with deadlock avoidance, static node assignment with deadlock recovery, dynamic node assignment with deadlock avoidance, dynamic node assignment with deadlock recovery, dynamic node assignment with waiting and deadlock avoidance, and dynamic node assignment with waiting and deadlock recovery. Together, these simulations represent approximately two weeks of computation time on the Sequent Balance 21000. As an example of the results obtained, we consider the central server network of Figure 1.

Central Server Networks

Central server networks have long been used as models of computer systems [Buze73], and consequently have pragmatic importance. Because they contain nested cycles, central server networks are susceptible to deadlock in a distributed simulation. Figure 2 shows the speedup obtained for the central server network in Figure 1. Even with five processors, the speedup barely exceeds unity. Moreover, this is using the single processor, static node assignment case as the basis for calculating speedup. As Table 1 shows, *the parallel implementation rarely completes more quickly than the sequential implementation.* Indeed, static node assignment with deadlock avoidance runs 16 times more slowly than the sequential implementation. Consequently, the speedups over an event-driven simulation are much lower than Figure 2 suggests.

These results are *significantly* more negative than earlier simulated results [Reed85]. A sequential simulation of a network, by its nature, imposes some sequential ordering on the evaluation of network nodes. When those nodes are not being evaluated, they do not generate null messages, nor can they deadlock. In contrast, all nodes are always active in a fully parallel implementation. They continue to receive and generate null messages while awaiting receipt of real messages. Thus, the overhead is *higher* than suggested by a sequential simulation of distributed simulation.

Summary

Distributed simulation has been the subject of several *simulated* performance studies; little or no experimental data have heretofore been available. Using queueing networks as the simulation application, we simulated a variety of such networks with varying workloads using several variations of the Chandy-Misra algorithm on a shared memory machine. These experiments show that, with rare exception, the Chandy-Misra approach to distributed simulation is not a viable approach to parallel simulation of queueing network models. There are two primary reasons for this. First, a single processor implementation of the Chandy-Misra algorithm is sometimes slower than the equivalent sequential, event-driven simulation. Second, networks with cycles require deadlock avoidance or recovery techniques. These techniques are extremely costly, and there is little prospect that they can be reduced to acceptable levels.

Acknowledgments

Jack Dongarra and the Advanced Computing Research Facility of Argonne National Laboratory graciously provided both advice and access to the Sequent Balance 21000.

References

[Buze73] J. P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," *Communications of the ACM,* Vol. 16, No. 9, September 1973, pp. 527-531.

[ChHM79] K. M. Chandy, V. Holmes, and J. Misra, "Distributed Simulation of Networks," *Computer Networks,* Vol. 3, No. 1, February 1979, pp. 105-113.

[ChMi79] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Transactions on Software Engineering,* Vol. SE-5, No. 5, September 1979, pp. 440-452.

[ChMi81] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM,* Vol. 24, No. 4, April 1981, pp. 198-206.

[Reed85] D. A. Reed, "Parallel Discrete Event Simulation: A Case Study," *Record of Proceedings: 18th Annual Simulation Symposium,* March 1985, pp. 95-107, *invited paper.*

[SaMS80] C. H. Sauer, E. A. MacNair, and S. Salsa, "A Language for Extended Queueing Networks," *IBM Journal of Research and Development,* Vol. 24, No. 6, November 1980, pp. 747-755.

[Seet78] M. Seethalakshmi, "Performance Analysis of Distributed Simulation," *M.S. Report,* Computer Science Department, University of Texas, Austin, Texas, 1978.

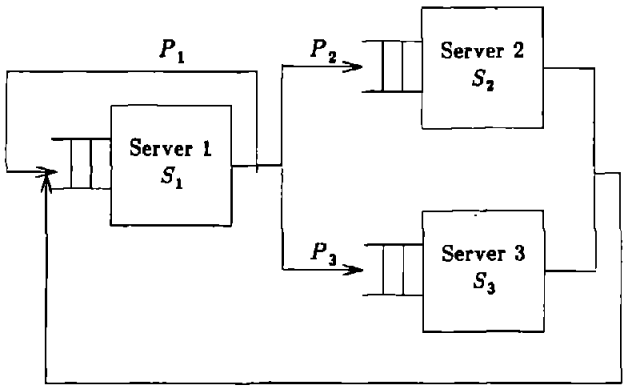

Figure 1a
Central Server Queueing Model

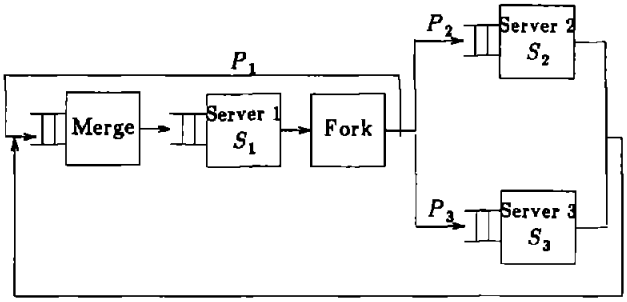

Figure 1b
RESQ Representation of Central Server Model

Figure 2
Speedup for five node central server
(static node assignment)

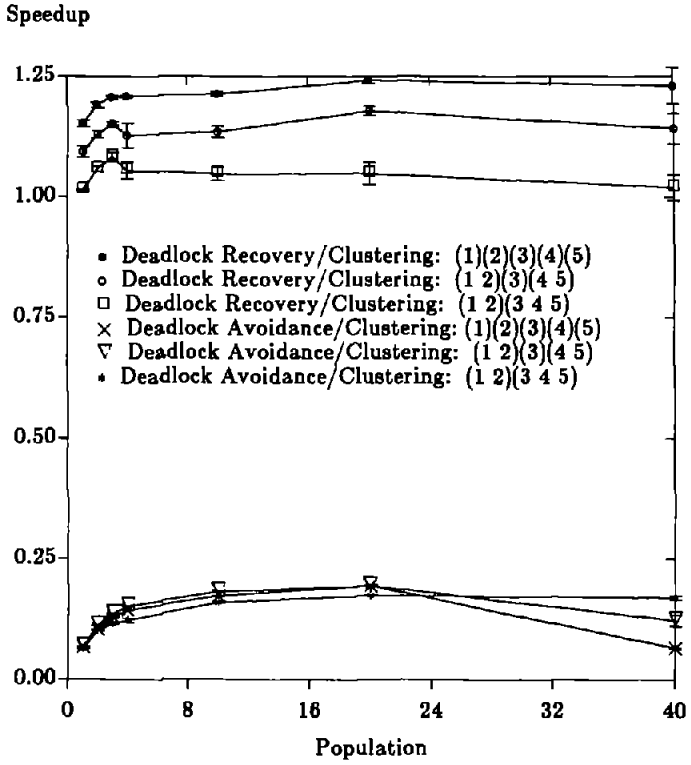

Table 1
Sequential and parallel mean execution time
for five node central server
(time given in seconds)

Population	SEQUENTIAL	STATIC PARALLEL		DYNAMIC PARALLEL			
		Recovery	*Avoidance*	*Recovery*	*Recovery w/ Waiting*	*Avoidance*	*Avoidance w/ Waiting*
1	26.32	28.97	491.85	33.62	35.47	569.00	662.30
2	42.80	44.67	510.71	50.19	56.70	619.56	655.15
3	51.44	52.73	490.48	59.89	61.95	599.29	632.47
4	56.96	56.92	477.92	63.64	67.02	580.51	623.12
10	67.22	67.20	471.20	76.66	82.69	601.95	602.09
20	74.42	70.58	450.91	83.47	88.70	628.46	620.91
40	87.78	74.74	1419.22	86.08	93.38	602.21	595.28

Parameter	Value
Routing Probability	(1) 0.10, (4) 0.45, (5) 0.45
Clustering case (5 PEs)	(1) (2) (3) (4) (5)*

*Node numbers refer to Figure 1. Parenthesized node groups execute on one processor.