Relaxing Causal Constraints in PDES*

Narayanan V. Thondugulam, Dhananjai Madhava Rao,
Radharamanan Radhakrishnan, and Philip A. Wilsey
Department of ECECS, University of Cincinnati,
P.O. Box 210030, Cincinnati, Ohio 45221, U.S.A.

Abstract

One of the major overheads that prohibits the wide spread deployment of parallel discrete event simulation (PDES) is the need to synchronize the distributed processes in the simulation. Considerable investigations have been conducted to analyze and optimize the two widely used synchronization strategies, namely the conservative *and the* optimistic *simulation paradigms. However, little attention has been focussed on the definition and strictness of causality. Does causality need to be preserved in all types of simulations? Previously, we had suggested an answer to this question. We had argued that significant performance gains can be achieved by reconsidering this definition to decide if the parallel simulation really needs to subscribe to the preservation of causality. In this paper, we investigate this issue even more closely. An in depth analysis using several example simulation models is presented in this paper. In addition, a comparative analysis between unsynchronized and Time Warp simulation is presented.*

1 Introduction

Despite its benefits, parallel simulation introduces the need to periodically synchronize the simulating entities for maintaining causality in the simulation. The synchronization needed for parallel simulation introduces overheads that often dominate simulator execution time. The notion of causality (or synchronization) is embedded in every aspect of traditional parallel discrete-event simulation (PDES). The distributed synchronization techniques chiefly fall under two categories, namely *optimistic* [5] and *conservative* [1]. While optimizations to these techniques have produced a remarkable improvement in performance, researchers have constantly been faced with the problem of reducing overheads in the simulation to improve performance [3]. Traditionally, conservation techniques have had to contend with the overheads of lookahead and deadlock avoidance & recovery [4] while optimistic techniques, such as Time Warp, have had to contend with rollback and state saving overheads [12]. Several of these overheads can be attributed to the synchronization (or maintaining causality) requirement of the simulation technique.

In our earlier work [13], we motivated the need for studying unsynchronized simulation. We investigated the notion of ignoring causality violations (and thereby avoid synchronization altogether) and studied the consequences of such a step. We had conducted a preliminary analysis and had presented our early results. In this paper, we presented an in depth study of the effect of unsynchronized simulation on queuing models and computer network models to analyze and characterize the simulation outputs. To verify the correctness of the simulation outputs, we compare the results with results obtained from WARPED [11], a Time Warp based parallel discrete event simulator. The remainder of this paper is organized as follows. Section 2 overviews parallel discrete event simulation. In Section 3, the motivation for studying unsynchronized parallel simulation in distributed environments is discussed. In Section 4, we present the models developed to evaluate the unsynchronized simulator. Finally, Section 5 presents some concluding remarks.

2 Background

In this section, a brief overview of PDES[3] is presented. In PDES, the model to be simulated is decomposed into *physical processes* that are modeled as *simulation objects*. Each simulation object is assigned to a *Logical Process* (LP); the simulator is composed of a set of LPs concurrently executing their simulation objects. Simulation objects communicate by exchang-

*Support for this work was provided in part by the Advanced Research Projects Agency under contracts J–FBI–93–116 and DABT63–96–C–0055.

ing time-stamped messages through the LPs. Thus, each LP (which can be associated with multiple simulation objects) receives messages from other LPs and forwards them to the destination objects. In order to maintain causality, LPs must process messages in strictly non-decreasing time-stamp order [5, 7]. There are two basic synchronization protocols used to ensure that this condition is not violated: (i) *conservative* and (ii) *optimistic*. Conservative protocols [1, 9] strictly avoid causality errors, while optimistic protocols, such as Time Warp [5] allow causality errors to occur, but implement some recovery mechanism.

3 Why Unsynchronized Simulation?

Is synchronization overrated? This is precisely the question we seek to investigate and study. Nicol [10] reminds us of the dangers of allowing "risk" when synchronizing a parallel discrete event simulation; a simulation code that runs correctly on a serial machine may, when run in parallel, fail catastrophically. This can happen when Time Warp presents an "inconsistent" message to an LP, a message that makes absolutely no sense given the LP's state. Failure may result if the simulation modeler did not anticipate the possibility of this inconsistency. While the problem is not new, there has been little discussion on how to deal with this problem; furthermore the problem may not be evident to new users of parallel simulation.

While Nicol [10] reports how this problem may occur and the damage it may cause, this paper presents a different approach to this problem. What if the modeler was aware of the occurrence of inconsistent messages in a simulation and chooses to ignore them? This is particularly true if the modeler was more interested in quick and less accurate results as opposed to highly accurate results. This is, of course, dependent on the application being simulated and the type of results that are monitored. Martini, Rümeskasten and Tölle [8] experiment with the notion of relaxing causality in their simulations of interconnected computer networks. In their experiments, optimistic execution is performed within intervals of a conservatively synchronized simulation and no recovery scheme is invoked when a causality violation is encountered. While this exposes the simulation to erroneous computations, Martini *et al* argue that the advantages of "tolerant or relaxed synchronization" heavily outweigh the disadvantages of such an approach. Specifically, they mention that the error in the simulation results is low and very often within the range of the confidence intervals. In addition, Martini *et al* report that the simulation execution times can be significantly reduced with the introduction of tolerant synchronization and this reduction in execution time is a very "practical" advantage of tolerant synchronization.

In this paper, we investigate this notion of relaxing causality in more detail and study the effects of ignoring causality in simulations. Only the systems that exhibit stochastic behavior were considered for unsynchronized simulation. In particular, we looked at discrete queuing network applications which form a vast majority of real life systems. For this purpose, an unsynchronized simulator (called NoTime) was developed along with a library of queuing models and computer network models. A detailed description of the architecture of the NoTime simulation kernel is available in the literature [13].

4 Analysis

Unsynchronized simulation can be applied to observe lumped properties of a number of discrete-time Markov chains which are easy to conceptualize and understand [6]. Queuing systems can be considered as a case of feedback networks, where the feedback path results from the server interacting with the queue to select the next customer. The feedback nature of a network makes it more difficult to perform distributed synchronization. In addition, unsynchronized simulation has a greater probability of yielding inaccurate results. However, the stochastic nature of the queuing model makes it an interesting case for simulation using unsynchronized simulation. Queuing systems are a subset of birth-death processes. The arrival of a customer to a queue can be represented as a birth in the system. When a customer leaves a queue (before or after being serviced) can be modeled as a "death". Queuing systems play a central role in a number of important physical systems. Simulation has often been employed when mathematical analysis of complex queuing systems becomes intractable. The queueing model we consider is a very general queueing $G/G/m$ system [6]. This is a system whose inter-arrival time distribution is completely arbitrary and whose service time distribution is also arbitrary (all inter-arrival times are assumed to be independent of each other). The system has m servers and order of service is also quite arbitrary (in particular, it need not be first-come-first-serve). The queuing library developed is based on these queuing systems. A number of parameters of the queuing system can be specified in the library. The parameters that can be evaluated using these models are average of wait times, average queue length, average of server idle times and average server utilization.

A computer network model is an example of a feed-

forward network where messages are exchanged between two arbitrary nodes through some intermediate nodes. The stochastic nature of a computer network model makes it suitable for unsynchronized simulation. The messages are generated randomly based on some random distributions *e.g.,* Poisson or Normal. The network model consists of sets of nodes connected to routers. The number of nodes and routers and their interconnections can be specified in the configuration file. The network traffic can be adjusted by varying the message generation rate λ at each of the nodes. The other parameters that can be fine-tuned are the service times and buffer sizes of routers and the network propagation delays. The routing algorithm is static which is evaluated once the network topology is known. Performance characteristic of the network can be monitored by observing the average message latencies and router utilization.

Several experiments were conducted using the developed models to evaluate the accuracy and performance of the NoTime kernel. The simulations were performed on dual Pentium-Pro(x686) workstations running Linux version 2.0.33. The experiments conducted can be divided into two categories (i) those performed to assess the accuracy of NoTime and (ii) those performed to evaluate the performance of NoTime.

4.1 Accuracy of NoTime

The first question that comes to mind while considering unsynchronized simulation is the accuracy of unsynchronized simulations. The limits of the kernel were explored towards answering this issue. The primary factors affecting the accuracy of simulation are processor speeds, communications latencies, granularity, partitioning and execution time of simulations (in terms of events processed). The first two factors (processor speeds and communication latency) are not within the control of the simulator. Hence, granularity, partitioning, and length of simulations were varied to study their effects on accuracy. Since the simulation environment consisted of only dual Pentium-Pro workstations running Linux, the processor speeds were uniform except when the work load assigned to each machine was different.

In the first set of experiments, different partitioning schemes were devised to assess their effects on accuracy. As expected, the simulation results were quite sensitive to partitioning. Table 1 shows a comparison of the results of different partitioning strategies and their deviations from desired result. A general example of a queuing system encountered in banks or airline reservations counters, with a source feeding a simple FIFO queue served by a number of servers was used.

Clients	Real Qlen	LPs	Avg Qlen/Partition		
			1	2	3
10000	45.0	2	829.2	34.1	27.4
		3	848.5	205.1	**46.1**
		4	1270.0	461.6	39.8
50000	129.3	2	8809.8	**120.7**	83.0
		3	6381.5	1457.8	70.0
		4	8650.5	2667.4	89.6
100000	204.5	2	2483.2	224.0	166.3
		3	12650.7	2361.8	145.2
		4	177218	5790.8	**193.4**
300000	392.4	2	16251.1	541.3	**445.4**
		3	31449.5	7528.6	509.1
		4	123265	14116.8	485.8
500000	663.3	2	34510.1	854.9	**750.4**
		3	56052.8	12529.6	799.6
		4	132783	28495	784.8

Table 1. Effect of 3 different partitions

In the first partitioning scheme, the simulation objects were evenly distributed across the processors. *e.g.,* if there are nine servers and three CPUs then each processor would be allocated 4 objects since the total number of simulation objects equals twelve. In the second partitioning strategy, source, queue and sink objects were assigned one processor and the servers were evenly distributed across the remaining processors. In the third strategy, the source and the queue object were assigned one processor while the server and sink objects were evenly distributed across remaining processors. It is clear from the table that the third partitioning strategy is the most effective and the results are fairly accurate for any number of processors. Several configurations of the queuing model were simulated with varying processor loads, to test the stability of the third partitioning scheme. In a majority of the test cases results were fairly accurate indicating the feasibility of unsynchronized simulation under suitable partitioning schemes. In the second set of experiments, the granularity of the events were varied. The granularity of the events(customers) was increased in order to make the time spent in communicating events a small fraction of the event processing time. As expected, the results were far more accurate and the dependency of simulation on communication latency and on partitioning could be eliminated. Beyond a certain granularity it was noticed that the simulation results tend to stabilize. In accordance with the *strong law of large numbers*, the various random parameters of the queuing system smoothen out and converge to the ex-

Messages	AML (actual)	LPs	AML (observed)	Error(%)
1500	2794.6	2	2853.4	2.10
		3	2801.8	0.25
		4	2826.1	1.12
3000	5506.6	2	5532.7	0.47
		3	5517.2	0.19
		4	5537.4	0.55
6000	10917.8	2	10968.0	0.45
		3	10946.5	0.26
		4	10978.5	0.55
9000	16325.9	2	16370.6	0.27
		3	16370.6	0.27
		4	16374.6	0.29
12000	21744	2	21770.2	0.12
		3	21777.4	0.15
		4	21782.8	0.17
15000	27159.4	2	29106.6	7.16
		3	27196.9	0.13
		4	27184.4	0.09
18000	32564.3	2	33751.6	3.64
		3	32611.2	0.14
		4	32594.4	0.09

Table 2. Average Message Latency (AML)

pected values as the simulation progresses. The various deviations from the mean value (mainly due to causal violations) cancel out each other, thus stabilizing the observed data.

Table 2 shows a comparison of the actual versus the observed average message latency (AML) obtained by simulating the computer network model with NoTime. The NoTime kernel simulates to a high degree of accuracy as can be noticed from the table. The feed-forward nature of the model makes it suitable for unsynchronized simulation. For arriving at fast and approximate solutions, NoTime is certainly efficient. One of the limitations with the NoTime kernel is in case of bounded buffer sizes of routers where the results tend to be less accurate.

4.2 Performance of NoTime

One of the compelling reasons for experimenting with an unsynchronized simulator was to achieve significant performance gains over the synchronous simulators. Synchronization costs for distributed simulation forms a major overhead with the result sequential simulation is faster than parallel simulation. This is especially true for fine grained applications where the communication and synchronization costs are high Several experiments with varying granularity and message traffic were performed to characterize the performance of the NoTime simulation kernel. The results obtained were compared against the WARPED kernel. The WARPED kernel was configured with *infrequent state saving* [2] for all the simulations to minimize the time taken. This configuration reduces the memory requirements drastically especially for high message traffic networks.

Figures 1 and 2 shows a comparison of the time taken for the simulation of a simple queuing model in NoTime and WARPED kernels. It can be noticed from the graph that the scalability of the NoTime kernel is almost linear. On the other hand, the WARPED simulator take almost a quadratic/exponential time. The memory requirements of Time Warp simulators increases exponentially with the size of the application. This is especially true for some configurations of the queuing model where the system is unstable. A queueing system is said to be unstable when the net inflow into the system is more than the net outflow, resulting in a queue overflow. In such cases the state of the queue object is very large and the performance of time warp simulator degrades rapidly due to state saving costs.

Figures 3 and 4 shows a comparison of the time taken to simulate the computer network model with the NoTime and WARPED kernels. Again the speed up achieved is significant. The configuration for which the readings are shown, is a case of a high traffic network where router utilization is high. For such configurations, optimistic simulators pay heavy synchronization costs in terms of state saving and rollbacks. This set of experiments demonstrate the fact that for fast, approximate solutions, NoTime is ideal. In addition, for large simulations, Time warp simulators run into memory exhaustion problem as compared to NoTime which is highly scalable. The graphs for timing information clearly depict the fact that NoTime makes large scale simulation feasible.

5 Conclusions and Future Work

In this paper, we have presented the benefits of relaxing (or completely doing away with) strict causal adherence in parallel and distributed simulation of queueing systems. We have argued that it is not always necessary to synchronize and incur the overheads of synchronization. In our earlier study [13], we introduced the idea of unsynchronized simulation. In this paper we have added more experiments to demonstrate the feasibility of unsynchronized simulation. The experiments show the suitability of unsynchronized simulation to stochastic models.

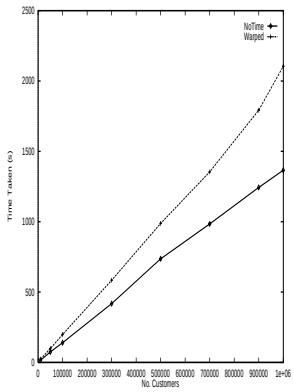

Figure 1. 2 LPs

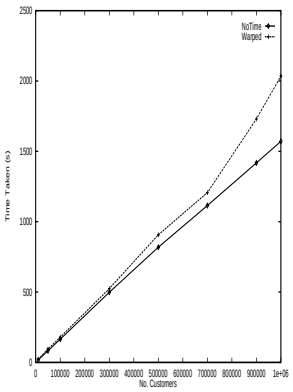

Figure 2. 3 LPs

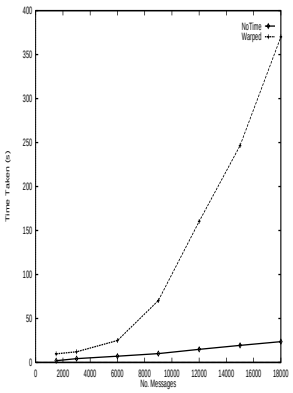

Figure 3. 2 LPs

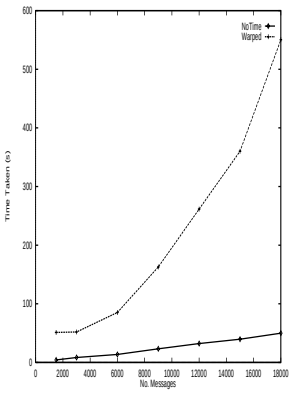

Figure 4. 3 LPs

The results presented in this paper clearly show the advantages of ignoring causality in simulations. Specifically, they are as follows: (i) faster simulation execution times, (ii) memory consumption is a fraction of what is needed for Time Warp simulation (as states are not saved), (iii) data obtained from unsynchronized simulation closely follow the data obtained from a Time Warp synchronized simulation (error rate is less than 2% on the average for our experiments), and (iv) no change in the modeling paradigm is required for such systems.

Of course, the unsynchronized simulation introduces errors in the simulation results. But our results show that this error is very small in many cases, sometimes even within the level of confidence for the correct results. The simulation is sensitive to the partitioning strategy which when carefully devised can overcome the dependence of simulation on load variations. In addition, by increasing the event granularity, the dependence on communication latencies can be reduced. More studies to control asynchronism and to reduce sensitivity to load variations is currently ongoing. Statistical techniques to recover from errors are also being investigated.

References

[1] K. M. Chandy and J. Misra. Asynchronous distributed simulation via a sequence of parallel computations. *Communications of the ACM*, 24(11):198–206, Apr. 1981.

[2] J. Fleischmann and P. A. Wilsey. Comparative analysis of periodic state saving techniques in Time Warp simulators. In *Proc. of the 9th Workshop on Parallel and Distributed Simulation (PADS 95)*, pages 50–58, June 1995.

[3] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, Oct. 1990.

[4] B. Groselji and C. Tropper. A deadlock resolution scheme for distributed simulations. *Transactions of the Scociety for Computer Simulation*, 6(2):89–132, Apr 1989.

[5] D. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):405–425, July 1985.

[6] L. Kleinrock. *Queueing Systems*. John Wiley & Sons, New York, NY, 1975.

[7] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of ACM*, 21(7):558–565, July 1978.

[8] P. Martini, M. Rümekasten, and J. Tölle. Tolerant synchronization for distributed simulations of interconnected computer networks. In *Proc of the 11th Workshop on Parallel and Distributed Simulation (PADS 97)*, pages 138–141. Society for Computer Simulation, June 1997.

[9] J. Misra. Distributed discrete-event simulation. *Computing Surveys*, 18(1):39–65, Mar. 1986.

[10] D. Nicol and X. Liu. The dark side of risk (what your mother never told you about Time Warp). In *Proc. of the 11th Workshop on Parallel and Distributed Simulation (PADS 97)*, pages 188–195, June 1997.

[11] R. Radhakrishnan, D. E. Martin, M. Chetlur, D. M. Rao, and P. A. Wilsey. An Object-Oriented Time Warp Simulation Kernel. In D. Caromel, R. R. Oldehoeft, and M. Tholburn, editors, *Proceedings of the International Symposium on Computing in Object-Oriented Parallel Environments (ISCOPE'98)*, volume LNCS 1505, pages 13–23. Springer-Verlag, Dec. 1998.

[12] R. Rajan, R. Radhakrishnan, and P. A. Wilsey. Dynamic cancellation: Selecting Time Warp cancellation strategies at runtime. *VLSI Design*, 1999. (forthcoming).

[13] D. M. Rao, N. V. Thondugulam, R. Radhakrishnan, and P. A. Wilsey. Unsynchronized parallel discrete event simulation. In *Proceedings of the 1998 Winter Simulation Conference*, Dec. 1998.