Proceedings of the 1998 Winter Simulation Conference
D.J. Medeiros, E.F. Watson, J.S. Carson and M.S. Manivannan, eds.

UNSYNCHRONIZED PARALLEL DISCRETE EVENT SIMULATION

Dhananjai Madhava Rao
Narayanan V. Thondugulam
Radharamanan Radhakrishnan
Philip A. Wilsey

Computer Architecture Design Laboratory
P.O. Box 210030, Cincinnati, Ohio 45221–0030, U.S.A.

ABSTRACT

Distributed synchronization for parallel simulation is generally classified as being either *optimistic* or *conservative*. While considerable investigations have been conducted to analyze and optimize each of these synchronization strategies, very little study on the definition and strictness of causality have been conducted. Do we really need to preserve causality in all types of simulations? This paper attempts to answer this question. We argue that significant performance gains can be made by reconsidering this definition to decide if the parallel simulation needs to preserve causality. We investigate the feasibility of *unsynchronized parallel simulation* through the use of several queuing model simulations and present a comparative analysis between unsynchronized and Time Warp simulation.

1 INTRODUCTION

Parallel discrete event simulators are used in today's high performance and parallel computing world for architecture design and application development. They are necessary because prototypes of the applications are time-consuming to build and difficult to modify. While sequential simulators have traditionally been used for this purpose, they tend to be slow and inadequate for detailed simulation of large systems. Consequently, parallel processing is often used in an attempt to speed up simulations, as well as to meet the larger memory demands of simulated applications. Despite its benefits, parallel simulation requires periodic synchronization between the simulating entities for maintaining causality in the simulation. This synchronization introduces overheads that often dominates simulator execution time.

The notion of causality (or synchronization) is embedded in almost every aspect of traditional parallel discrete-event simulation (PDES). Most simulationists prescribe to one of the two widely known distributed synchronization techniques, namely *optimistic* (Jefferson 1985; Steinman 1991) and *conservative* (Bryant 1979; Misra 1986). 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 (Fujimoto 1990). Traditionally, conservation techniques have had to contend with the overheads of lookahead and deadlock avoidance and recovery (Misra 1986) while optimistic techniques, such as Time Warp, have had to contend with rollback and state saving overheads. Several of these overheads can be attributed to the synchronization (or maintaining causality) requirement of the simulation technique. To overcome this problem, several researchers have proposed methods for relaxing the strict causality requirement in distributed simulation. Some preliminary insight into this relaxation possibility can be found by reviewing the Time Warp optimization called *rollback relaxation* (Wilsey and Palaniswamy 1994). Rollback relaxation proposes the use of a relaxed recovery mechanism when memoryless logical process receive straggler events. The memoryless property can be detected using standard optimizing compiler techniques (live variable analysis) and results in a relaxed definition of causality. More precisely, it redefines the recovery process to require the re-processing of only a subset of the events in the input queue when rollback occurs. Similarly, logical processes representing stateless functions mapping a single input to a single output can be completely implemented without synchronization (Wilsey and Palaniswamy 1994).

In this paper, we investigate the notion of ignoring causality violations (and thereby avoiding synchronization altogether) and study the consequences of such a step. In particular, we study the effect of unsynchronized simulation on simple queueing models in an attempt to analyze

and characterize the simulation outputs. To verify the correctness of the simulation outputs, we compare the results with results obtained from WARPED (Martin et al. 1995), a Time Warp based parallel discrete event simulator. The remainder of this paper is organized as follows. Section 2 overviews parallel discrete event simulation. Section 3 reviews some related approaches. In Section 4, the justification for unsynchronized parallel simulation in distributed environments is discussed. Section 4.1 describes the design and construction of NOTIME, the unsynchronized simulator. In Section 5, we present the queueing model, and evaluate the unsynchronized simulator using this model. Finally, Section 6 presents some concluding remarks.

2 BACKGROUND

In PDES, the model to be simulated is decomposed into *physical processes* that are modeled as *simulation objects* and assigned to a *Logical Process* (LP). The simulator is composed of a set of concurrently executing LPs. The LPs communicate by exchanging time-stamped messages. In order to maintain causality, LPs must process messages in strictly non-decreasing time-stamp order (Jefferson 1985; Lamport 1978). There are two basic synchronization protocols used to ensure that this condition is not violated: (i) *conservative* and (ii) *optimistic*. Conservative protocols (Bryant 1979) strictly avoid causality errors, while optimistic protocols, such as Time Warp (Fujimoto 1990; Jefferson 1985) allow causality errors to occur, but implement some recovery mechanism.

Conservative protocols were the first distributed simulation mechanisms. The basic problem conservative mechanisms must address is the determination of "safe" events. The conservative process must first determine that it is impossible for it to receive another event with a lower timestamp than the event it is currently trying to execute. Such events are deduced to be safe and can be executed without any causality violation in the system. Processes containing no safe events must block; this can lead to deadlock situations if no appropriate precautions are taken. Several studies on conservative mechanisms and optimizations to conservative protocols have been presented in the literature (Bryant 1979; Misra 1986).

In a Time Warp simulator, each LP operates as a distinct discrete event simulator, maintaining input and output event lists, a state queue, and a local simulation time (called *Local Virtual Time* or LVT). Each LP processes events optimistically and moves ahead in LVT. As each LP simulates asynchronously, it is possible for an LP to receive an event from the past, a *straggler* (some LPs will be processing faster than others and hence will have local virtual times greater than others), — violating the causality constraints of the events in the simulation. On receipt of a straggler message, the LP must rollback to undo some work that has been done. Rollback involves two steps: (i) restoring the state to a time preceding the time-stamp of the straggler and (ii) canceling any output event messages that were erroneously sent (by sending *anti-messages*). After rollback, the events are re-executed in the proper order.

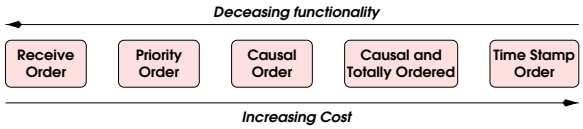


Figure 1: Message Ordering Schemes

3 RELATED WORK

Parallel discrete event simulation (PDES) has gained much attention during the last decade (Fujimoto 1990). A reflection of this is the number of researchers who have used either conservative or optimistic synchronization to successfully simulate their applications. With the extensive refinement and optimization of these techniques, synchronization has emerged to be the single largest overhead in a distributed simulation. The overhead of a synchronization protocol lies in its message ordering and delivery service (Fujimotoi and Weatherly 1996). Message ordering characteristics specify the order and time at which messages may be delivered. For example, in the High Level Architecture (HLA) Time Management services, Fujimoto (Fujimotoi and Weatherly 1996) has identified five types of message ordering services. Figure 1 illustrates the five different types of message orderings. These message ordering schemes provide, in turn, increased functionality but at increased cost. The most straightforward (least functionality), lowest latency ordering mechanism is the *Receive Order* scheme. Depending on what type of synchronization is required, the developer can select a message ordering category that best suits the purpose. The approaches for alleviating the synchronization overhead have fallen chiefly along two lines: those that work to reducing the number of times synchronization is required and those that work to eliminate the need to synchronize or maintain causality.

Nicol and Heidelberger (1995) have shown that continuous time Markov chains (CTMCs) can be simulated in parallel more efficiently than most other discrete event systems. Their methods exploit the randomization approach for CTMCs, which allows the precomputation of event times. Using this approach, synchronization points are predicted a priori and this knowledge is used to increase the possible parallelism. Their approach is viable

for both optimistic and conservative schemes. Buchholz (1997) went one step further than Nicol and Heidelberger. In addition to the precomputation of the communication times, he overlapped numerical analysis with simulation to improve the parallel conservative simulation of CTMCs. Buchholz called this new technique for the analysis of CTMCs, "hybrid simulation".

Martini, Rümeskasten and Tölle (1997) propose a novel synchronization protocol called "tolerant synchronization" in which a conservative synchronization protocol is allowed to optimistically process events that are within a tolerance level. As the protocol optimistically processes events, causality violations may occur but these violations are ignored and no recovery process is initiated to rectify the errors. In detailed simulations of interconnected computer networks, Martini *et al* noted that although the introduction of optimistic execution without recovery in a conservatively synchronized simulation introduces errors in the results, the errors are almost negligible and for certain cases, the error can be recalculated with the help of statistical methods.

4 UNSYNCHRONIZED SIMULATION

Is synchronization overrated? This is precisely the question we seek to investigate and study. Nicol (Nicol and Liu 1997) reminds us of the dangers of allowing "risk" when synchronizing a parallel discrete event simulation. While Nicol (Nicol and Liu 1997) reports how this problem may occur and the damage it may cause, this paper presents a different spin to this problem. What if the modeler was aware of the occurrence of inconsistent messages in a simulation and chose 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(1997) 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.

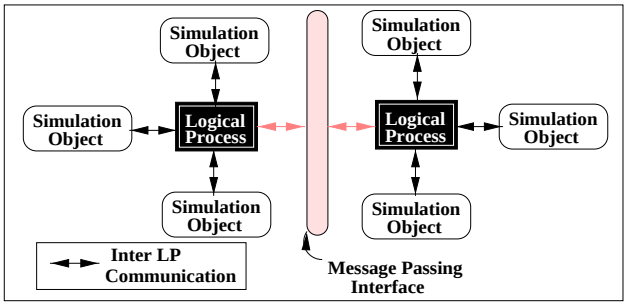


Figure 2: Architecture of NoTime

In this paper, we investigate the notion of relaxing causality in more detail and study the effects of ignoring causality in simulations. For this purpose, an unsynchronized simulator (NoTime) was developed along with a library of queuing models. The following subsections detail the architecture of the simulator and the queuing model library.

4.1 The NoTime Unsynchronized Simulator

The NoTime simulation kernel provides the functionality to develop applications modeled as discrete event simulations. The standard programming interface of WARPED (Martin et al. 1995) was adopted to provide an uniform interface to the application. It keeps the underlying mechanisms of different simulators transparent to the modeler. In NoTime, objects are grouped into entities called logical process or LPs (Figure 2). Processor parallelism occurs at the LP level and each LP is responsible for communication management and scheduling the simulation objects that it contains. In addition, communication between simulation objects within the same LP is performed by direct insertion of the event into the input queue of the receiving object. Communication between nonlocal objects is achieved through the use of the Message Passing Interface (MPI) (Gropp et al. 1994). MPI was chosen to keep the system portable so that shared memory systems as well as a network of workstations can be used. Since the parallelism occurs at the LP level, simulation objects that execute relatively independent of each other can be placed on different LPs to maximize parallelism (Figure 2). Conversely, simulation objects that frequently communicate with each other should be placed on the same LP to benefit from fast intra-LP communication. The NoTime system is composed of a set of C++ classes and libraries which the user accesses using *inheritance* or *method invocation.*

An LP, in the NoTime system, acts as a wrapper to a set of *simulation objects*. Each simulation object has a notion of an "input queue" and a set of state variables

that are associated with each object. The structure and interface of the state class that uses a simple C++ class to wrap state variables is provided. To optimize the activities associated with scheduling the objects and handling intra-LP communication, the different input queues of the objects on a LP are combined together into a single structure. The common input queue, is a simple First in First output (FIFO) data structure used to hold the unprocessed events of all simulation objects [1].

The simulation cycle on each LP proceeds in the traditional fashion. The scheduler (present at the LP level) checks and schedules simulation objects depending on the events present. Scheduling is based on the order in which events arrive into the queue. In essence, the scheduler's data structure is a simple FIFO queue, consistent with that of the simulation object. The NoTime simulation kernel uses a simple termination detection algorithm whereby a token is circulated between the LPs to collect information on each LP's status. Leader election is arbitrary. The termination detection assumes the underlying communication system is FIFO (which is guaranteed by MPI). In addition, a simple interactive simulation environment was built to help monitor the progress of the simulation on the various LPs.

4.2 Queueing Library

A reconfigurable queueing library with the essential components needed to model queueing systems was built on top of the NoTime simulation kernel. The queueing library consists of *source*, *queue*, *server* and *statistics collector* objects. The model and the various parameters of the source, server and queue objects can be set by the modeler through simple configuration files. Different scenarios can be modeled by specifying the desired layout in the configuration file. A number of such models were used to experiment and analyze the results and evaluate the performance of NoTime. Since a consistent interface was used, the same models were simulated on WARPED, a Time Warp simulator, and the results obtained were used for comparisons. The models used and the results obtained are illustrated in the following section.

5 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 (Kleinrock 1975). A set of random variables, X_n forms a Markov chain if the probability that the next value (or state) is X_{n+1} depends only upon the current value (or state) X_n and not upon any previous values. The term "memoryless processes" are attributed to systems with this property and is required of all Markov chains. One of the most important and interesting subset of these Markovian chains are the Birth-Death processes. Birth-Death processes can be imagined as random processes where the birth & death (or arrival & departure) of the different elements are random processes. Queueing 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". Queueing systems play a central role in a number of important physical systems. Many of the systems that we often encounter can be modeled using queueing theory. Simulations have often been employed when mathematical analysis of complex queueing systems becomes intractable. Many mathematical analysis techniques use the inherent memoryless property of the queueing models to approximate their behaviors, a classic example being *Little's Law*. We exploit this property of the queueing systems in our unsynchronized simulations.

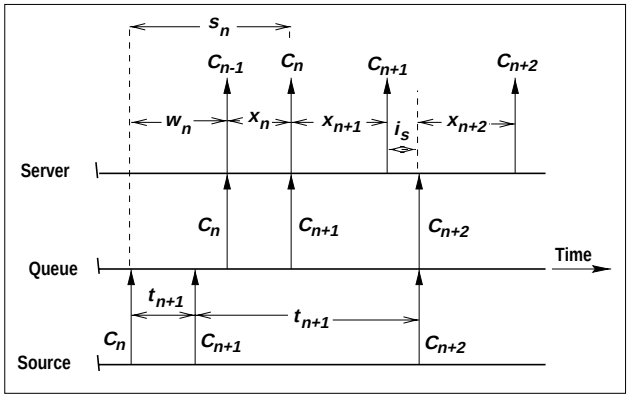


Figure 3: Time-diagram Notation for Queues

5.1 The Queueing Model

The queueing model we consider is a very general queueing $G/G/m$ system (Kleinrock 1975). This is a system whose inter-arrival time distribution $A(t)$ is completely arbitrary and whose service time distribution $B(x)$ 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 queueing library developed is based on these queueing systems. A number of parameters of the queueing system can be specified in the library. Figure 3 illustrates the important parameters used in modeling the systems. Let C_n represent the n^{th} customer in the

[1] In a Time Warp based simulation kernel, the input queue is required to be a sorted list of events.

Table 1: High Server Utilization Configuration(Source : Poisson distribution with mean 15, 9 Servers each using a Normal service time distributions with mean 150 and variance of 15) with an *average utilization = 0.9924*

S.No	Clients	No. LPs	Avg. Queue Length		% Error	Avg. Waiting Time		% Error
			TW	NoTime		TW	NoTime	
1	20000	1	1180.4	1141.1	3.33	17664	17137	2.98
		2	1187.6	1143.0	3.75	17768	17149	3.48
		3	1144.8	1114.5	2.64	17185	17762	-3.35
2	40000	1	2342.5	2290.2	2.23	35092	34379	2.02
		2	2349.4	2305.0	1.88	35144	34563	1.65
		3	2278.2	2275.3	0.12	34217	34169	0.14
3	80000	1	4647.3	4573.1	1.59	69649	68630	1.46
		2	4694.9	4606.6	1.88	70298	69077	1.73
		3	4573.7	4510.4	1.38	68618	67775	1.22
4	100000	1	5805.2	5711.2	1.61	87013	85730	1.47
		2	5851.4	5767.0	1.44	87685	86448	1.41
		3	5722.2	5640.8	1.42	85847	84739	1.29

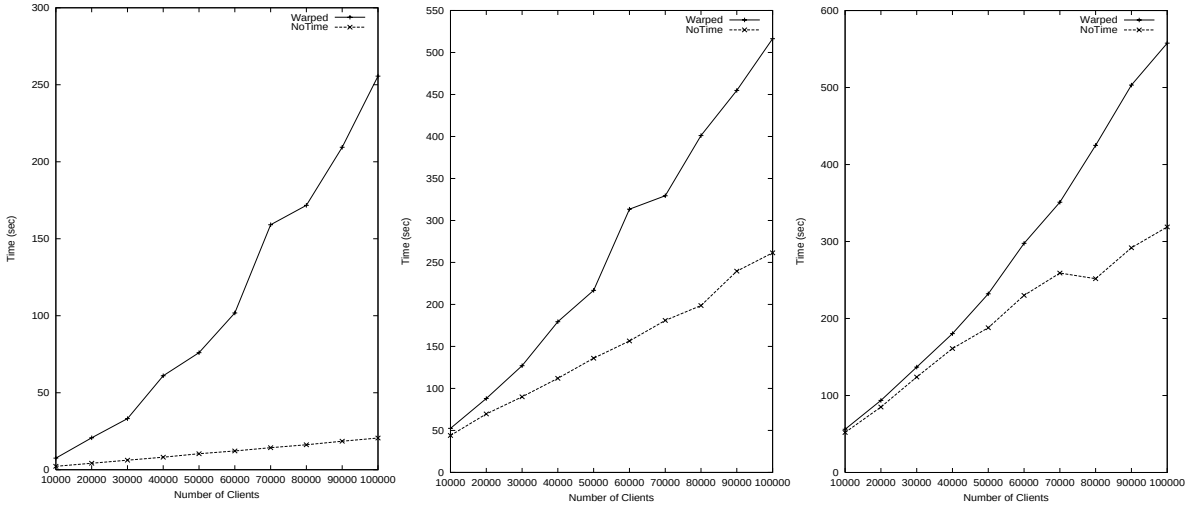
system. The parameter t_{n+1} represents the inter-arrival time of the customers. This parameter is modeled using a distribution (which can be specified by the *source object*). The parameter x_n (specified with a random distribution) models the time taken to serve a customer. Some of the parameters of interest are the average of wait times (average of w_ns), average queue length, average of server idle times (average of i_ss) and average server utilization.

5.2 Experiments and Results

The queueing library developed for NoTime was used to model different queueing systems. A generic example of a queueing system having a source feeding a simple FIFO queue served by a number of servers was used in the experiments. A self activating source that schedules itself based on a random distribution is used to generate the arrival of customers into the system. The queue manages the en-queueing and dequeuing of the customers as they arrive and are serviced by the servers. The queue maintains a table of busy/idle servers in the system. The servers communicate the processing times to the queue for the various customers. The processing time of different customers by the different servers is modeled using a probabilistic distribution. The flow of customers in the system is communicated to the statistics class that generates the statistics of the system at the end of simulation. Table 1 and 2 tabulate results obtained from simulating some of the models. The readings of the simulation were taken on a Sun Sparc Shared Memory Multi-Processor (SMP) machine with 4 processors, running SunOS version 5.6. Experiments are conducted on NoTime as well as WARPED, the Time Warp based parallel simulation kernel. The following subsection presents a comparative analysis between the statistics obtained from the two simulators.

5.3 Statistics

Table 1 tabulates the data obtained from a saturated queueing system run on one, two and three LPs configurations where server utilization is close to 100%, and the Queue length does not stabilize (average of three separate runs is shown). It keeps growing as the demand is more than what the servers can handle. This is an unstable queueing system which does not reach a steady state as the inflow is more than the outflow. Since the average service time is lower than the inter-arrival times of clients, this is expected. This kind of a queue would be helpful to study queue overflows. Such queueing systems are simulated with a fair degree of accuracy by NoTime (the error as compared to Time Warp as shown in Table 1). The interesting thing to note in such a system, is that NoTime is much faster than WARPED due to the overhead paid in terms of state saving in Time Warp. Also as illustrated in Figure 4, NoTime is quite scalable for such configurations. Finally, the tables establish the accuracy with which NoTime can simulate such a system. For single LP cases, it is very accurate (as it is pretty much a sequential execution). But when we simulated in parallel, we observed that NoTime is sensitive to load changes on the machine. As long as the load remained consistent, the simulation results were consistent. When the load changed during simulation, the data got skewed. Figure 4 present a comparative picture of the time taken by NoTime and WARPED to simulate the models (the average of three separate runs is shown).



Single LP Two LPs Three LPs

Figure 4: Timing information obtained with High Server Utilization Configuration (Source : Poisson distribution with mean 15, 9 Servers each using a Normal service time distributions with mean 150 and variance of 15)

Table 2 tabulates the results obtained from a configuration, where the server utilization is not high (The average of three separate runs is shown). NOTIME is roughly 3 times faster for single LP case but for the 3 LP case, WARPED and NOTIME are equally good. This is due to high communication overheads of MPI that dominate simulation time.

The intuition about the consistency and accuracy of the results obtained from unsynchronized simulation is presented in Figure 5. In accordance with the *Strong law of large numbers*, the various random parameters of the queueing system smoothen out and converge to the expected values as the simulation progresses. The deviations from the mean value (mainly due to causal violations) cancel out each other, thus stabilizing the observed data. Figure 6 provides insight into how the standard deviations in the data obtained decreases as the length of simulation increases. The deviations were obtained by simulating the model a number of times and maximum and minimum values of the parameters are plotted. The data is representative of a "stable" queueing system. By a "stable" queueing system we mean that the rate of service is proportional to the rate of arrival.

Clearly, unsynchronized simulation can be useful in situations where quick and less accurate results are preferred to time-consuming and highly accurate simulation results. Considering that NOTIME is approximately three times faster than an optimized Time Warp simulator, it is clear that NOTIME will be helpful in making fast and more general simulation studies to find the most promising design alternatives without simulating them in great detail.

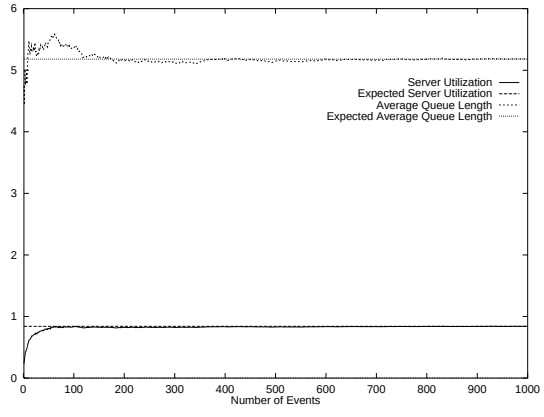


Figure 5: Stabilization of parameters

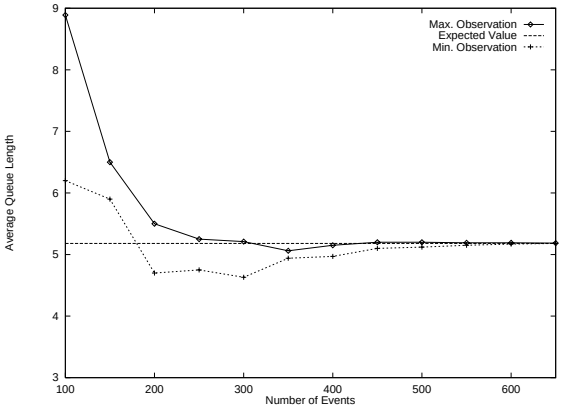


Figure 6: Deviations in observed data

Table 2: Low Server Utilization Configuration (Source : Poisson distribution with mean 15, 7 Servers each using a Normal service time distributions with mean 100 and variance of 15) with an *average utilization = 0.94*

S.No	Clients	No. LPs	Avg. Queue Length		% Error	Avg. Waiting Time		% Error
			TW	NoTime		TW	NoTime	
1	20000	1	7.28	7.16	1.64	108.9	107.5	1.28
		2	7.26	7.22	0.55	108.7	108.4	0.27
		3	7.15	7.33	-2.51	107.4	109.6	-2.05
2	40000	1	7.23	7.20	0.41	108.4	108.2	0.18
		2	7.29	7.23	0.82	109.1	108.4	0.64
		3	7.16	7.17	-0.14	107.5	107.7	-0.18
3	80000	1	7.22	7.21	0.13	108.3	108.2	0.09
		2	7.24	7.21	0.41	108.4	108.3	0.09
		3	7.18	7.23	-0.69	107.7	108.7	-0.93
4	100000	1	7.21	7.21	0.00	108.1	108.2	-0.09
		2	7.22	7.21	0.13	108.2	108.3	-0.09
		3	7.18	7.33	-2.08	107.7	110.2	-2.33

6 CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the benefits of relaxing (or completely doing away with) strict causal adherence in the parallel and distributed simulation of queueing systems. We have argued that it is not always necessary to synchronize and incur the overheads of synchronization. While we have just started to scratch the surface of this problem, recent research focus has started moving in this direction as is evident from the literature.

The results presented in this paper clearly show the advantages of ignoring causality in simulations to be (i) The simulation execution times can be considerably reduced, (ii) The memory consumption is a fraction of what is needed for Time Warp simulation (as states are not saved), (iii) The 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) There is no change in the modeling paradigm for such systems.

Of course, the unsynchronized simulations 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. Studies to control asynchronism and to reduce sensitivity to load variations are currently ongoing. Statistically techniques to recover from errors are also being investigated.

ACKNOWLEDGEMENTS

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.

REFERENCES

Bryant, R. E. (1979). Simulation on a distributed system. In *Proc. of the 16th Design Automation Conference*, pp. 544–552.

Buchholz, P. (1997). A distributed numerical/simulative algorithm for the analysis of large continuous time markov chains. In *Proc. of the 11th Workshop on Parallel and Distributed Simulation (PADS 97)*, pp. 4–11. Society for Computer Simulation.

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

Fujimotoi, R. M. and R. M. Weatherly (1996). Time management in the dod high level architecture. In *Proc. of the 10th Workshop on Parallel and Distributed Simulation (PADS 96)*, pp. 60–67. Society for Computer Simulation.

Gropp, W., E. Lusk, and A. Skjellum (1994). *Using MPI: Portable Parallel Programming with the Message-Passing Interface*. Cambridge, MA: MIT Press.

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

Kleinrock, L. (1975). *Queueing Systems*. New York, NY: John Wiley & Sons.

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

Martin, D. E., T. McBrayer, and P. A. Wilsey (1995). WARPED: A time warp simulation kernel for analysis and application development. (available on the www at `http://www.ece.uc.edu/~paw/warped/`).

Martini, P., M. Rümekasten, and J. Tölle (1997). Tolerant synchronization for distributed simulations of interconnected computer networks. In *Proc of the*

11th Workshop on Parallel and Distributed Simulation (PADS 97), pp. 138–141. Society for Computer Simulation.

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

Nicol, D. and X. Liu (1997). 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)*, pp. 188–195.

Nicol, D. M. and P. Heidelberger (1995). A comparative study of parallel algorithms for simulating continuous time markov chains. *ACM Transactions on Modeling and Computer Simulation 5*, 326–354.

Steinman, J. S. (1991). SPEEDES: A unified approach to parallel simulation. In *6th Workshop on Parallel and Distributed Simulation*, pp. 75–84. Society for Computer Simulation.

Wilsey, P. A. and A. Palaniswamy (1994). Rollback relaxation: A technique for reducing rollback costs in an optimistically synchronized simulation. In *International Conference on Simulation and Hardware Description Languages*, pp. 143–148. Society for Computer Simulation.

AUTHOR BIOGRAPHIES

DHANANJAI MADHAVA RAO is a Ph.D. student in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received his Bachelor's degree in Computer Science and Engineering from the University of Madras in 1996. His research interests include parallel discrete event driven simulation, parallel processing, and web-based simulation of active networks.

NARAYANAN V. THONDUGULAM is an MS student in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received a B.Tech degree in Electrical and Electronics Engineering from the Indian Institute of Technology, Madras in 1996. His research interests include parallel discrete event simulation, computer architecture, and unsynchronized simulation of discrete queuing models.

RADHARAMANAN RADHAKRISHNAN is a Ph.D student in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received a BE degree in Computer Science and Engineering from the University of Madras in 1993. His current research interests include parallel discrete event driven simulation, adaptive control, parallel processing, application of formal methods to distributed systems and active networks.

PHILIP A. WILSEY is an Assistant Professor in the Department of Electrical and Computer Engineering & Computer Science at the University of Cincinnati. He received PhD and MS degrees in Computer Science from the University of Southwestern Louisiana and a BS degree in Mathematics from Illinois State University. His current research interests are parallel and distributed processing, parallel discrete event driven simulation, computer aided design, formal methods and design verification, and computer architecture. He is a member of both the IEEE Computer Society and the ACM.