CONF-980448--

Title: An Adaptive Synchronization Protocol
For Parallel Discrete Event Simulation

Author(s): Keith R. Bisset

Submitted to: 1998 Advanced Simulation Technologies Conference
31st Annual Simulation Symposium
April 5-9, 1998, Boston MA

Los Alamos
NATIONAL LABORATORY

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

An Adaptive Synchronization Protocol for Parallel Discrete Event Simulation

Keith R. Bisset

TSA 5, MS F602

Los Alamos National Laboratory

Los Alamos, NM 87545

kbisset@lanl.gov

Abstract

Simulation, especially discrete event simulation (DES), is used in a variety of disciplines where numerical methods are difficult or impossible to apply. One problem with this method is that a sufficiently detailed simulation may take hours or days to execute, and multiple runs may be needed in order to generate the desired results. Parallel discrete event simulation (PDES) has been explored for many years as a method to decrease the time taken to execute a simulation. Many protocols have been developed which work well for particular types of simulations, but perform poorly when used for other types of simulations. Often it is difficult to know a priori *whether a particular protocol is appropriate for a given problem. In this work, an adaptive synchronization method (ASM) is developed which works well on an entire spectrum of problems. The ASM determines, using an artificial neural network (ANN), the likelihood that a particular event is safe to process.*

1 Parallel discrete event simulation

Simulation, especially discrete event simulation (DES), is used in a variety of disciplines where numerical methods are difficult or impossible to apply. One problem with this method is that a sufficiently detailed simulation may take hours or days to execute, and multiple runs may be needed in order to generate the desired results. In general there are three ways to speed up the execution of a DES: the algorithms in the simulation can be improved (both in the model and in the simulation framework), the simulation can be executed on a faster machine, and the simulation can be run in parallel on several processors. The last method, parallel discrete event simulation (PDES), is the focus of this research.

The typical method to parallelize a discrete event simulation is to divide the model that is being simulated into separate sub-models, called physical processes. For example, in a manufacturing plant simulation each machine could be considered a separate physical process. Each physical process is then simulated by a logical process (LP).

An LP communicates with another LP by transmitting a message that contains an event. An event consists of an execution time (the timestamp), which indicates the simulation time at which the event is to be executed on the destination LP, the model-specific actions to be performed, and the source (i.e., transmitting) and the destination (i.e., receiving) LPs. It is possible for the source LP and destination LP to be the same. When an LP receives a message, the event that is contained in the message is scheduled for execution. The source LP is said to have generated the event that is scheduled on the destination LP. Each LP maintains its own independant simulation clock, called the local virtual time (LVT). The LVT is the timestamp of the last event that has been executed. There is also a global virtual time (GVT), which is defined as the minimum of all the LVTs and any messages in transit in the simulation. The GVT records the global progress of the simulation.

The behavior of a PDES is influenced by many factors, including the model that is being simulated, the hardware on which the simulation is executing, the communication network that connects the processors, and the simulation framework (the infrastructure upon which the model is built). The situation becomes even more complex when the simulation is run on a heterogeneous collection of machines with varying computation power.

Reynolds [14] has identified nine design variables that can be used to describe the spectrum of PDES synchronization. The two which are most directly related to this work are *aggressiveness* and *risk*.

Aggressiveness is the willingness of an LP to process an event that may turn out to be incorrect. An LP that never executes events out of timestamp order has zero aggressiveness. An LP that allows events to be executed out of order, but later corrects the errors that result, is aggressive.

Risk is the willingness of an LP to transmit messages that later may have to be retracted because an event was processed incorrectly (i.e., out of order). Nonzero risk im-

plies that the simulation has some method to undo the effects of erroneous messages (e.g., anti-messages).

1.1 Conservative PDES

The conservative simulation protocol [3, 2] prevents out of order execution of events. Using the terminology of Reynolds [14], it is non-aggressive and non-risky. An event is not executed by an LP unless it can be guaranteed that no other event with a smaller timestamp (a straggler) will be received. Such an event is called a safe event. If there are no known safe events, the LP will block until a safe event becomes available. A safe event can become available in one of two ways: a new event is received and determined to be safe, or a previously received event is determined to be safe, usually because an event was received from another LP.

In general, messages from one LP to another must be sent in increasing timestamp order. A message from LP_1 to LP_2 with timestamp t is a guarantee by LP_1 to LP_2 that LP_1 will not send another message with a timestamp less than t. In this way, an LP can determine if an event is safe to process. If an LP has received a message from every other LP with a timestamp larger that the timestamp of the event in question, the event is safe. To prevent deadlocks, LPs send out *null messages*, messages with no corresponding event, only a timestamp. These messages simply serve to update the minimum timestamp on messages from a particular LP.

1.2 Optimistic PDES

In contrast to conservative simulation, optimistic simulation [8] assumes that every event is safe. Again using Reynold's [14] terminology, optimistic simulation is aggressive and most optimistic protocols are risky. When the assumption that an event is safe to process is proved wrong, by the receipt of a straggler (i.e., an event which should have been processed in the past), events which have already been processed and have a timestamp greater than that of the straggler must be rolled back. This rollback may include the sending of anti-messages to cancel messages transmitted in error.

An anti-message is simply a copy of the original message with the sign reversed. If the positive message to be canceled has not yet been executed by the destination LP, it is simply removed from the message queue of the destination LP. If the message has been processed, the destination LP must be rolled back to a time prior to that of the timestamp of the canceled message, in order to undo any effects of the incorrect message.

Message cancellation may either be aggressive (every message transmitted by a rolled back event is canceled immediately), or lazy (only events which are not regenerated when the rolled back event is reexecuted are canceled) [6, 9]. Aggressive cancellation increases the speed at which messages are canceled, at the risk of canceling some messages unnecessarily. Lazy cancellation only corrects truly erroneous messages, at the risk of more erroneous messages being executed, possibly increasing the number of rollbacks.

2 Adaptive simulation

Many protocols have been developed to reduce the aggressiveness and risk of optimistic protocols (e.g., [13, 16]) and to increase the aggressiveness and risk of conservative protocols (e.g., [4, 15]). These protocols do not answer the basic question: "Is a particular event safe to process?" The approach taken in this research is to determine, using an Adaptive Synchronization Method (ASM), the likelihood that a particular event is safe to process.

Other attempts at adaptively controlling PDES (e.g., [1, 11, 5, 7, 10] have used adaptively controlled parameters such as blocking windows (the amount of wall–clock time to block after each event is processed). These protocols do not treat each event individually. As a (simplified) example, take a simulation with two types of events, t_1 and t_2. Events of type t_1 almost always are rolled back, while events of type t_2 are almost never rolled back. As ASM would execute events of type t_2 immediately, but block for some amount of wall–clock time before executing events of type t_2 in order to reduce rollbacks. An adaptive protocol that works on an per LP basis would have to determine a balance between executing events of type t_1 immediately, and blocking before executing events of type t_2, making a choice which is not optimal for either type of event.

In a complex simulation, the appropriate action may depend not only on the type of event, but possibly on the events source, destination, the amount of aggressive processing on the destination LP, or the type of LP as well. Given this complex (and possibly changing) set of dependencies, using an ASM to decide, on a per event basis, how much time to wait prior to executing an event makes sense.

The ASM takes as input several parameters describing the state of the LP and the event in question (e.g., event type, LP type, the difference between the local and global virtual times, or the number of rollbacks by this LP). All of these parameters are local to the LP, thus do not incur any communication overhead. Given these parameters, the ASM will return a prediction of the likelihood that the event is safe to process (i.e., will not roll back). This likelihood is similar to a probability, but may have some differences, depending on the implementation of the ASM. For instance, it may not be limited to the range 0 to 1. Given this likelihood, a decision will be made whether or not to execute the event. In this way aggressiveness can be controlled on a per event basis.

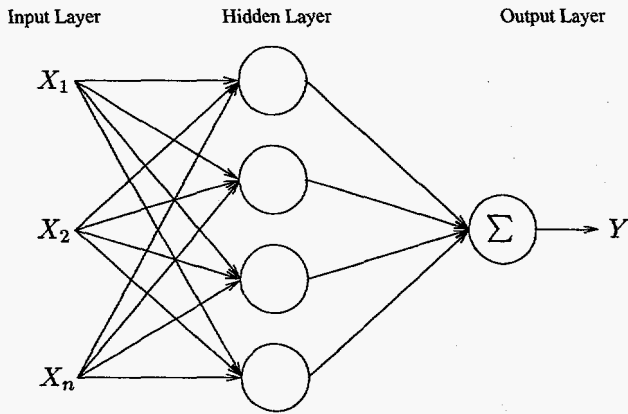The ASM can be implemented using any adaptive

Figure 1: Neural network architecture with inputs $X_1, X_2, \ldots, X_n$, output Y, and one hidden layer.

Figure 2: Detail of an ANN unit.

method, including an artificial neural network (ANN), fuzzy logic, or a genetic algorithm. For this work, an ANN was chosen. The reason for this choice is a ANN's ability to deal with a high-dimension input space, and the ability to interpolate between known data points.

3 Artificial neural networks

An in-depth treatment of artificial neural networks (ANN) is beyond the scope of this paper, more complete descriptions can be found in [12]. A brief introduction is presented below.

An ANN is a simplified model of neural processing of the brain, consisting of simple interconnected processing units. It is able to learn to approximate any function, by using example data. An ANN is arranged into layers: an input layer, an output layer, and one or more hidden layers, so called because they are not directly connected to the network's inputs or outputs. An ANN with one hidden layer is shown in Figure 1. The X_i are inputs to the network, and Y is the output of the network.

One of the most common ANN architectures is the multi–layer perceptron (MLP). An MLP unit is shown in Figure 2. The output of a unit is a function of the weighted sum of its inputs, as shown in Equation 1. The network learns by adjusting the weights on the inputs to achieve the desired output. The output function is a non–linear function which maps its input to the range $(0, 1)$. A common output function is the sigmoid function shown in Equation 2.
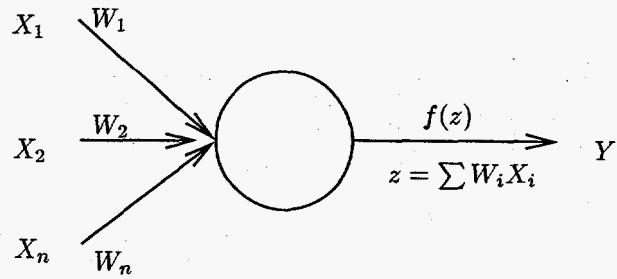
$$Y = f\left(\sum W_i X_i\right) \qquad (1)$$

$$f(z) = \frac{1}{1 + \exp(-z)} \qquad (2)$$

The network is trained by presenting example inputs and adjusting the weights of the network to minimize the error. Training can be either supervised or unsupervised. In supervised training, the desired output is presented with each set of inputs and the network is trained to produce the desired output. Unsupervised training only presents the inputs to the network. It is up to the network itself to group similar sets of inputs to produce the same output. This is useful in classification, where the actual output is not important, only that inputs which belong to the same class produce similar outputs. Only supervised training is considered in this work.

A common learning algorithm for ANNs is back propagation. The difference between the actual output, Y, and the target output, $Y^{(t)}$, is computed and propagated from the output layer, back through the hidden layers to the input layer, adjusting the weights on the inputs. The weight change between unit j and unit i is shown in Equations 3–5, where η is the learning rate, which controls how fast the network learns.

$$\Delta w_{ji} = \eta \delta_j X_i \qquad (3)$$

$$\delta_j = \begin{cases} \left(\frac{\partial f}{\partial \text{net}_j}\right)\left(y_j^{(t)} - y_j\right) & \text{if } j \text{ is an output unit} \\ \\ \left(\frac{\partial f}{\partial \text{net}_j}\right)\sum_q W_{qj}\delta_q & \text{if } j \text{ is a hidden unit} \end{cases} \qquad (4)$$

$$\text{net}_j = \sum_q W_{jq} X_q \qquad (5)$$

Training can be either off-line or online. In off-line training, all of the training examples are collected before training starts, and presented to the network as a group. Online training presents training examples individually, and can be interleaved with network evaluation. Usually online training examples are generated by the application as it executes.

4 Simulation framework

An object oriented PDES framework, object oriented parallel simulation system (OOPSS), was written to develop and test new simulation protocols. Through the use of inheritance and polymorphism, parts of the simulation framework can be replaced (e.g., substitute an optimistic protocol for a conservative one) without changing the rest of the framework, or the models being tested. This allows different protocols to share much of the code, allowing fair comparisons to be made between them.

OOPSS also features an abstract parallel machine interface, which currently has two implementations: message passing interface (MPI), and a distributed simulation simulator (DSS). The MPI interface allows OOPSS to run on any parallel machine or network which supports MPI, which includes many recent parallel computers as well as networks of workstations. DSS allows OOPSS to run on a single processor, but simulate a run on multiple processors. DSS uses threads to interleave the computation from each simulated processor, and simulates the interconnection network using a fixed delay for each message, but does not simulate network contention.

5 Models used for testing

The model used to evaluate optimal simulation is a simple queuing network that can be configured to make it more efficiently simulated under the conservative or optimistic protocol. The LPs are arranged into two rings, the odd numbered LPs in a slow ring and the even numbered LPs in a fast ring. LPs in the fast ring process events ten times faster than LPs in the slow ring. Two types of events are used: intra-ring and inter-ring. The LP configuration is shown in Figure 3.

An intra-ring event executed at time t causes a intra-ring event to be scheduled at time $t + 2$ on the next LP in the ring. Additionally, an inter-ring event is scheduled on the corresponding LP in the other ring at time $t + 1$, with probability p. An inter-ring event does not cause any other events to be scheduled. Each LP starts with a single intra-ring event scheduled at time $t = 1$.

An intra-ring event takes 10000 μsecs on a slow LP and 1000 μsecs on a fast LP. An inter-ring event takes no time to process on either ring. In this way the amount of time taken by event processing is independent of p, and any differences in execution time for the same size prob-
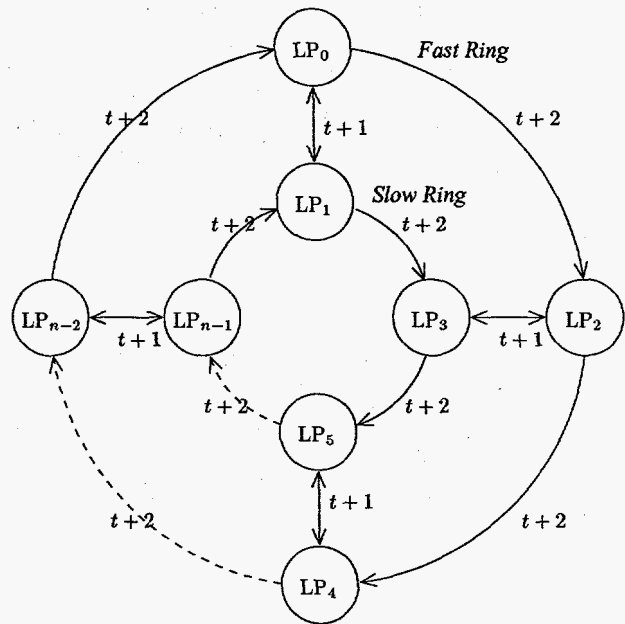

Figure 3: Model used for testing.

lem are due to simulation overheads.

The adjustable parameter, p, changes the behavior of the simulation so that it is more efficiently simulated by conservative or optimistic protocols. If $p = 0$, no inter-ring events are scheduled. The model is more efficiently simulated with an optimistic protocol. Each LP in a ring has the same number of events to process, so the LVT of LPs within a ring stay fairly close. Since there is no communication between rings, they can be simulated independently. Under an optimistic protocol, the cost of rollbacks is small, since no LP simulates far into the future of an LP from which it received events. Under a conservative protocol, after a fast LP finishes executing an intra-ring event, the LP must block until the corresponding LP in the other ring has finished executing its intra-ring event and sends a null-message. This forces the two rings to stay synchronized.

If $p = 1$, an inter-ring event is sent by every intra-ring event. The model is more efficiently simulated with a conservative protocol. Under a conservative protocol, after a LP on the fast ring executes an intra-ring event, the LP blocks until it receives a inter-ring event from the corresponding LP in the other ring. Under an optimistic protocol, after a intra-ring event is executed by a fast LP, the LP proceeds to execute additional intra-ring events until the inter-ring event is received. At this point the LP must rollback the additional intra-ring events just executed.

Five different methods for choosing p are used for testing. The first three methods use the same value of p
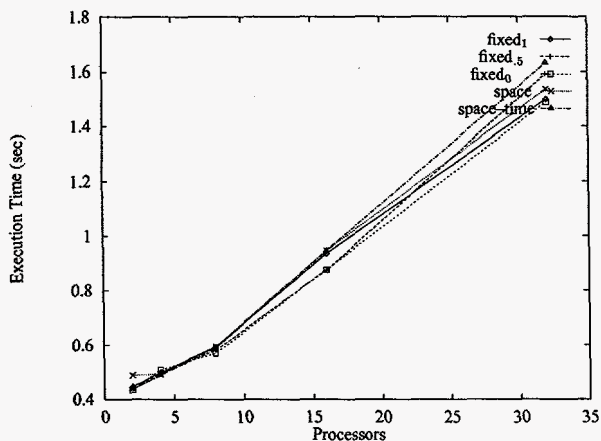
Figure 4: Comparison of different models under conservative synchronization.

Figure 5: Comparison of different models under optimistic synchronization.

for each LP in the system, 0, 0.5, 1, and are referred to as $fixed_0$, $fixed_{.5}$, and $fixed_1$, respectively. The fourth method chooses p from a uniform distribution for each LP. In this method, p remains fixed for the duration of the simulation. This method is referred to as *space*, since p varies in space (across LPs). The fifth method changes p after every ten events are executed, and is referred to as *space-time*, since p varies in space and time.

Each combination of synchronization protocol and p is run on varying problem sizes, ranging from 2 to 32 LPs. Each LP is run on its own processor. The simulation is run until a simulation time of 300 seconds is reached. For each simulation run, the execution time, overhead due to blocking and overhead due to rollbacks are measured.

Figure 4 shows the effect of varying p under the conservative protocol. The execution time remains fairly constant no matter what value of p is used, however the efficiency is very different. This is due to the protocol's conservative nature. After an intra-ring event is executed on a fast LP, it must block until the corresponding slow LP finishes executing. The slow LP will send either an intra-ring event (model $fixed_1$), or a null-message (model $fixed_0$). If an LP has blocked waiting for a null-message, the LP could have processed the next event, it simply lacked the information needed to make that decision. Any time spent blocked is simply wasted. If an LP is blocked waiting for a real message, the correct execution depends upon that message and the block is necessary.

Figure 5 shows the effect of varying p under the optimistic protocol. The $fixed_0$ model is executed efficiently, due to the absence of inter-ring events. Each LP receives events from only one other LP, in increasing timestamp order, so no rollbacks occur. Once inter-ring events are introduced into the simulation (i.e., $p > 0$), the cost of a
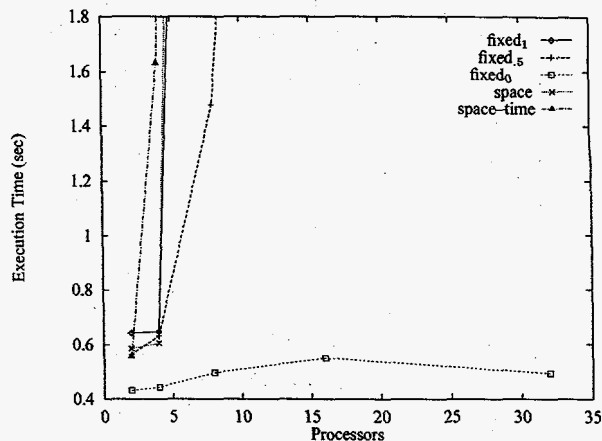
rollback becomes high. In the course of one event execution time on a slow LP, 10 intra-ring events can be executed on a fast LP, which will in turn lead to 9 intra-ring events executed on the next fast LP, and so on. At least 55 intra-ring events may have to be rolled back due to the rollback of the first event. This number may increase due to the propagation delays of anti-messages.

6 Adaptive synchronization method

The ASM protocol was implemented using the Stuttgart Neural Network Simulator (SNNS) [17], developed by the Institute for Parallel and Distributed High Performance Systems at the University of Stuttgart. SNNS features a graphical user interface for developing and training ANNs and a kernel which can be embedded in another application in order to use an ANN during runtime.

An MLP network with one hidden layer comprising of 21 units was chosen, using the standard backpropagation algorithm for training. The training set was generated from instrumented runs of the $fixed_0$ and $fixed_1$ models using the optimistic protocol. A training example was generated for each event that was executed, along with an indication of whether the event rolled back or not. The collected data was processed to to remove duplicates, creating a training set consisting of 3019 examples, 1240 for events which were rolled back, and 1779 for events which were not rolled back.

The input to the ANN consisted of seven parameters: event type, source LP type, destination LP type, source aggressiveness, destination aggressiveness, rollbacks per time, and rolled back events per time. The source is the LP that generated an event, and the destination is the LP on which the event is to be executed. The type of an LP is the ring which contains the LP (fast or slow). The aggres-
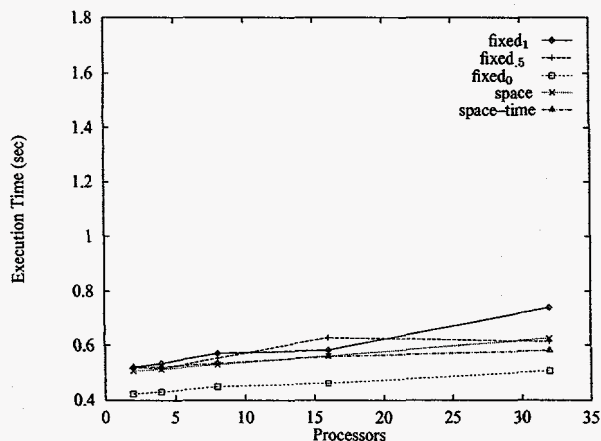
Figure 6: Comparison of different models under adaptive synchronization.

Figure 7: Amount of time spent rolling back under optimistic synchronization.

Figure 8: Amount of time spent rolling back under adaptive synchronization.

siveness of the source is measured by the difference between the source's LVT when the event was generated and the GVT when the event is executed. The aggressiveness of the destination is the difference between the LVT on the destination and the GVT, when the event is executed. Rollbacks per time is a measure of how many times the destination LP has rolled back for each unit of simulation time it has executed. Rolled back events per time is a measure of the seriousness of those rollbacks. These parameters were chosen so that they each have a limited range (as opposed to LVT, which can increase without bound), and can be computed locally on the destination LP (except for LVT of the source, which is transmitted with the event).

The ASM protocol is similar to the optimistic protocol, with one major difference. Prior to executing an event, information about the event (the input parameters) is given to the ANN, which returns l, an estimate of the likelihood that the event will have to be rolled back. If l is less than an adjustable threshold (0.5 was used in this work), the event is executed immediately. Otherwise the event is not executed and control is returned to the simulation framework. Other outstanding work is completed (e.g., adding newly arrived events to the event queue, or updating GVT). When this work is done, the process repeats. The event evaluated by the network may be different, if an event with an earlier timestamp was received.

Figure 6 shows the effect of varying p under the adaptive protocol. The use of the ASM combines the strengths of the conservative and optimistic protocols while controlling their weaknesses. It allows the fast LPs to block where necessary (waiting on a real message) while proceeding where possible (waiting on a null-message).

Figures 7 and 8 compare the amount of time spent rolling back under the optimistic and adaptive protocols.
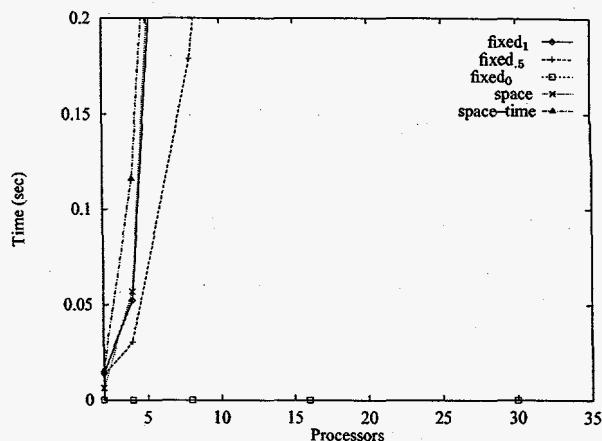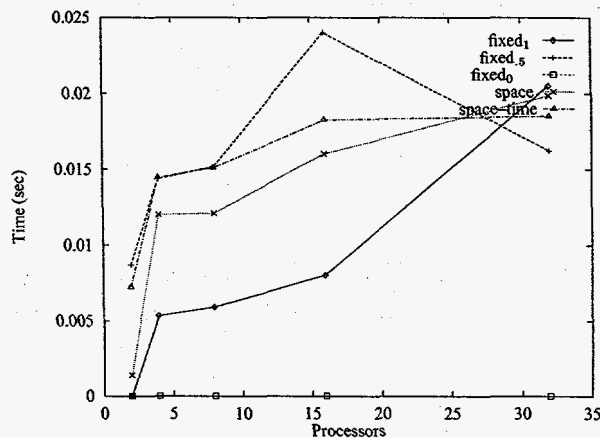
The rollback time increases greatly as more LPs are added, showing that the number of rollbacks is increasing, dwarfing the amount of time needed to process the events. Under the adaptive protocol, the time spent rolling back remains fairly constant, showing that the protocol allows some aggressive processing, but limits the amount of aggressiveness.

Figures 9 and 10 compare the amount of time spent blocked under the conservative and adaptive protocols. The amount of time spent blocked under the conservative protocol increases slightly as the number of LPS increases, due to the increased number of null messages which must be sent, but the time spent blocked remains fairly close under each model. The adaptive protocol only blocks when the number of rollbacks start to increase, so more time is spent blocked under model $fixed_1$ than model $fixed_0$.
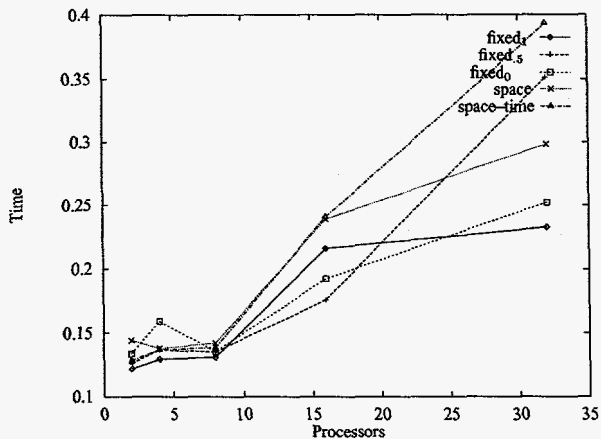
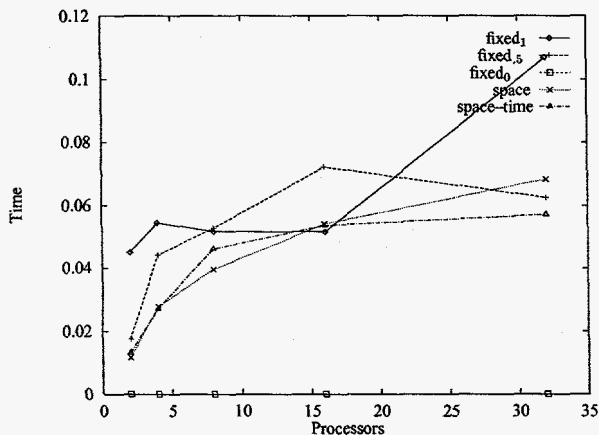Figure 9: Amount of time spent blocked under conservative synchronization.

Figure 10: Amount of time spent blocked under adaptive synchronization.

7 Future work

There are several areas left to explore. Perhaps the most exciting is adding the capability to do online training. As the simulation is running, the ANNs for each LP can be trained against the events that are being processed. This method has the capability to fine-tune each network for the specific LP it is running on, as well as adapting to changing behavior in the simulation. An example of such changing behavior is a battlefield simulation moving from the alert phase to the conflict phase. In the alert phase, units are mostly moving and observing, without much interaction. When the simulation moves into the conflict phase, units are actively engaging each other, with a much greater amount of interaction. A possible drawback to online learning is the amount of processing time needed for training. This problem can be alleviated by using one or

more dedicated processors to do the training, or only training when the effectiveness of a network falls below some threshold.

Another area to be explored is using ANNs to make other decisions in the simulation framework. Anyplace an arbitrary number is used, an ANN can adaptively select that number, possibly increasing performance. Examples of this include the periodic state saving interval and the GVT update interval. ANNs can also be used to select between different methods during the course of the simulation, such as the choice between periodic state saving and incremental state saving.

There are also many different ANN architectures and training methods which can be explored to attempt to increase the accuracy of the network. Using the simple methods presented in this paper, it has been shown that using an ANN to control the aggressiveness of an optimistic simulation has the potential for increasing the efficiency of PDES, and should be explored further.

References

[1] Duane Ball and Susan Hoyt. The adaptive time-warp concurrency control algorithm. In David Nicol, editor, *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 174–177, 1990.

[2] R. E. Bryant. Simulation of packet communications architecture computer systems. Technical Report MIT-LCS-TR-188, Massachusetts Institute of Technology, 1977.

[3] K. Mani Chandy and Jayadev Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, 5(5):440–452, 1979.

[4] Phillip M. Dickens and Paul F. Reynolds, Jr. Srads with local rollback. In David Nicol, editor, *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 22, pages 161–164, 1990.

[5] Alois Ferscha and G. Chiola. Self-adaptive logical processes: The probabilistic distributed simulation protocol. In *Proceedings of the 27th Simulation Symposium*, pages 78–88. IEEE Computer Society, 1994.

[6] Anat Gafni. Rollback mechanisms for optimistic distributed simulation systems. In Brian Unger and David Jefferson, editors, *Proceedings of the SCS Multiconference on Distributed Simulation*, volume 19, pages 61–67. SCS, 1988.

[7] Donald O. Hamnes and Anand Tripathi. Feedback based adaptive risk control protocols in parallel discrete event simulation. In *International Conference on Parallel Processing*, volume III, pages 93–96, 1995.

[8] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[9] Yi-Bing Lin and Edward D. Lazowska. A study of Time Warp rollback mechanisms. *ACM Transactions on Modeling and Computer Simulation*, 1(1):51–72, 1991.

[10] Bradley L. Nobel and Roger D. Chamberlain. Predicting the future: Resource requirements and predictive optimsim. In *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation*, volume 25, pages 157–164, 1995.

[11] Avinash C. Palaniswamy and Philip A. Wilsey. Parameterized time warp (PTW): An integrated adaptive solution to optimistic PDES. *Journal of Parallel and Distributed Computing*, 37(2):134–145, 1996.

[12] Duc Truong Pham and Xing Liu. *Neural Networks for Identification, Prediction and Control*. Springer-Verlag, 1995.

[13] P. Reiher, F. Wieland, and D. Jefferson. Limitation of optimism in the time warp operating system. In E.A. MacNair, K.J. Musselman, and P. Heidelberger, editors, *Proceedings of the 1989 Winter Simulation Conference*, pages 765–770, 1989.

[14] Paul F. Reynolds, Jr. A spectrum of options for parallel simulation. In M. Abrams, P. Haigh, and J. Comfort, editors, *Proceedings of the 1988 Winter Simulation Conference*, pages 325–332, 1988.

[15] Jeff S. Steinman. Interactive SPEEDES. In *Proceedings of the 24th Annual Simulation Symposium*, pages 149–158, 1991.

[16] Jeff S. Steinman. Breathing time warp. In Rajive Bagrodia and David Jefferson, editors, *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation*, volume 23, pages 109–118, 1993.

[17] Andreas Zell et al. Snns user manual, version 4.1. Technical report, Institute for Parallel and Distributed High Performance Systems, University of Stuttgart, 1995.

[8] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3):404–425, 1985.

[9] Yi-Bing Lin and Edward D. Lazowska. A study of Time Warp rollback mechanisms. *ACM Transactions on Modeling and Computer Simulation*, 1(1):51–72, 1991.

[10] Bradley L. Nobel and Roger D. Chamberlain. Predicting the future: Resource requirements and predictive optimsim. In *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation*, volume 25, pages 157–164, 1995.

[11] Avinash C. Palaniswamy and Philip A. Wilsey. Parameterized time warp (PTW): An integrated adaptive solution to optimistic PDES. *Journal of Parallel and Distributed Computing*, 37(2):134–145, 1996.

[12] Duc Truong Pham and Xing Liu. *Neural Networks for Identification, Prediction and Control*. Springer-Verlag, 1995.

[13] P. Reiher, F. Wieland, and D. Jefferson. Limitation of optimism in the time warp operating system. In E.A. MacNair, K.J. Musselman, and P. Heidelberger, editors, *Proceedings of the 1989 Winter Simulation Conference*, pages 765–770, 1989.

[14] Paul F. Reynolds, Jr. A spectrum of options for parallel simulation. In M. Abrams, P. Haigh, and J. Comfort, editors, *Proceedings of the 1988 Winter Simulation Conference*, pages 325–332, 1988.

[15] Jeff S. Steinman. Interactive SPEEDES. In *Proceedings of the 24th Annual Simulation Symposium*, pages 149–158, 1991.

[16] Jeff S. Steinman. Breathing time warp. In Rajive Bagrodia and David Jefferson, editors, *Proceedings of the SCS Multiconference on Parallel and Distributed Simulation*, volume 23, pages 109–118, 1993.

[17] Andreas Zell et al. Snns user manual, version 4.1. Technical report, Institute for Parallel and Distributed High Performance Systems, University of Stuttgart, 1995.