SRADS WITH LOCAL ROLLBACK

Phillip M. Dickens
Paul F. Reynolds, Jr.

IPC-TR-90-003
January 22, 1990

Institute for Parallel Computation
School of Engineering and Applied Science
University of Virginia
Charlottesville, VA 22903

This research was supported in part by Jet Propulsion Laboratory Contract #957721.

SRADS WITH LOCAL ROLLBACK

Phillip M. Dickens and Paul F. Reynolds, Jr,
Institute for Parallel Computation
School of Engineering and Applied Science
University of Virginia
Charlottesville, VA 22903
804/924-1039

ABSTRACT

There is reason to believe bounded aggressive processing (limiting the degree to which processes act on conditional knowledge) may be a good alternative to unbounded processing. Simulations characterized by substantial variance in logical process processing times can lead to conditions where, without bound, some processes may cause frequent repairs (e.g. rollbacks) to occur. We present an algorithm, SRADS/LR, in which we bound aggressiveness and discuss its expected impact on performance.

INTRODUCTION

Most of the literature in the field of parallel discrete event simulation partitions protocol types into two basic categories: *conservative* [ChMi79] [PeWo79] [Reyn82] [Nico89] and *optimistic* (Time Warp [Jeff85]). We have shown that this is an overly simplistic taxonomy [Reyn88], instead choosing to classify protocols based, among other things, on how much aggressiveness and risk they exhibit. Typically, protocols called conservative block when they do not have enough information to proceed. This blocking can occur when there is inadequate information to process incoming events (lack of aggressiveness) or when there is inadequate information to send events on to other processes (lack of risk). A blocking process may not proceed until it has enough information to guarantee that no other process can affect the accuracy of its computations. Since blocking is used to guarantee accuracy, deadlock can occur. Much research has been concentrated on mechanisms to prevent deadlock.

If a parallel synchronization protocol exhibits aggressiveness or risk then processes tend to be allowed to proceed at their own rate determined by the presence of any actions to be performed. In this case a rollback mechanism is typically used to guarantee accuracy. The rollback mechanism prevents a process from having to block and thus prevents deadlock.

The major criticism of blocking protocols is the use of blocking to guarantee accuracy. There are two major problems associated with blocking mechanisms. First, a processor may be required to remain idle even when there is useful work that it could perform. Second, blocking can potentially lead to deadlock and solutions to the deadlock problem are expensive.

Protocols such as Time Warp alleviate the need to use blocking by allowing a process to perform computation based on incomplete information. As mentioned above, this aggressive processing solves the deadlock problem since a process is never required to block for synchronization purposes. There are, however, two primary costs associated with this aggressiveness. One cost is a large memory requirement necessitated by state and anti-message saving. The second is the time required to periodically save state information and to rollback to an earlier state when it is discovered that some computation is incorrect. The costs of rollback can be quite large as one rollback can lead to a cascade of others.

In [Reyn88] Reynolds suggests allowing aggressive processing, but placing bounds on this aggressiveness. A process could perform computations based on incomplete information, but not pass on the results of this computation until it can be guaranteed that the results will not have to be rolled back. This would ensure that all rollbacks are strictly local and thus there would be no cascading rollbacks. We call this approach *local rollback.*

This paper introduces and explores the concept of a local rollback mechanism. We describe a modification to SRADS [Reyn82], an existing synchronization protocol, that allows for more locally aggressive processing. In particular, processing based on incomplete information is allowed, but the results of this computation are not passed along until it can be guaranteed they will never have to be rolled back. This allows controlled aggressive computation without the possibility of cascading rollbacks and with potentially less memory requirements than a completely aggressive approach.

We begin with an overview of the SRADS protocol.

SRADS

SRADS is a discrete event simulation protocol designed primarily for deterministic models where potential message arrival times are predictable. Examples of such models include logic networks, some queuing systems and generally any stochastic system that can be discretized. Studies performed by [ReKu86], [DaRe83], [Nico84] and [O'Hal83] have shown that SRADS performs quite well for this type of system.

In SRADS a process that may send a message to another process is termed a *writer* and a process that may receive a message is termed a *reader*. Note that a process may be both a reader and a writer. A reader is required to send a poll to its writers at regularly scheduled intervals. Polls are scheduled at the times the reader predicts it may receive a message from the writer. The polling mechanism is discussed in detail in the next section. As will be seen, the performance of the simulation is, to a large extent, dependent upon the ability of the reader to predict potential message arrival times. It is obvious that the polling mechanism requires that a reader know a priori the processes with which it may communicate. SRADS thus works

best with a static communication topology, although there are some cases in which it could work with a dynamic one.

Polling Mechanism

The polling mechanism in SRADS is used to both synchronize readers and writers, and to advance the simulation time of the reader in the absence of other available information. As mentioned above, a reader is required to poll its writers at regularly scheduled intervals. The timestamp of the poll event shows the current simulation time of the reader. After the poll has been issued, the reader blocks until the writer responds with a message (an acknowledgement) showing that it has advanced its simulation time to at least that of the reader. The poll is thus used to synchronize readers with their writers, ensuring that a reader does not get arbitrarily ahead of its writers.

Note that a reader may have enough information locally to be able to avoid a scheduled poll. This would be the case if it had received a message from a writer with a timestamp greater than the scheduled poll time. In this case the reader knows that the writer will not be able send a message with a timestamp earlier than the poll time and the reader may therefore proceed.

In addition to synchronizing processes, the polling mechanism also aggressively advances the simulation time of the reader in the absence of any other information. If the reader's next scheduled event is a poll, it will go ahead and advance its clock up to the poll time. The reader will then block and wait for a response from the writer. The reader *assumes* the writer has advanced its clock to at least this time, and blocks until it receives confirmation of this fact.

The use of the polling mechanism to both synchronize processes and aggressively advance simulation times can lead to a reader receiving a message in its logical past. This phenomenon is termed *time slip* and is caused by the aggressive assumptions made by the reader. The first assumption is that messages will only arrive at regularly occurring intervals. Using this assumption the reader only synchronizes with a writer at poll times. Between poll times readers and writers progress at their own rates. The second assumption is that it is acceptable to advance the clock of a reader to the next poll time. This is again based on assumptions about potential message arrival times. The reader is advancing its clock to the time of the next anticipated message arrival.

Note that SRADS does not quite fit into the "conservative"/"optimistic" dichotomy in that it does use blocking for synchronization but also allows for some aggressive processing. The polling mechanism is aggressive in that it makes assumptions as to the logical time of a writer and about potential message arrival times. It is conservative in that a reader must block until it knows for certain that the writer has progressed at least up to the time of the poll.

The following example should clarify how time slip can occur. In the next section we show how the addition of the local rollback mechanism can eliminate most of these problems.

Example 1

Consider a system consisting of three processes as shown in figure 1. Assume the application designer has determined with high probability that both processes A and B will send messages to process C at five time unit intervals. Process

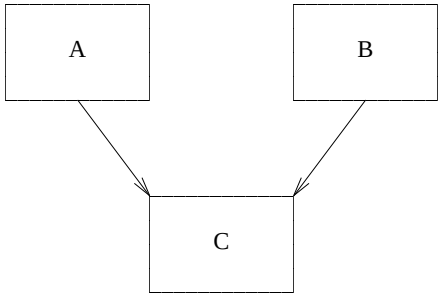

FIGURE 1. Example Network.

C will thus have a poll scheduled to both A and B every five time units.

Assume process C has sent a poll to both A and B at time 5 and is blocked waiting for their reply. Some time later A sends an acknowledgement to the poll showing that its logical clock is at time five. Before process B responds to the poll assume that process A sends an event message with a timestamp of nine. Note that process C cannot accept this message until it receives a response to the poll it issued to B.

Assume that some time later process B responds to the poll indicating that it has advanced its clock to time five. Process C will now go ahead and accept the event message it received from A. This is because C has completed the synchronization with B at time five, and does not have to synchronize again until time ten, the next scheduled poll time. It is now possible that B will send C a message between time five, their last synchronization time, and time nine, process C's current simulation time. In this case the message arrives in C's logical past. This late arrival is an example of time slip.

It should be noted that time slip does not occur if the protocol designer can accurately predict possible message arrival times. As discussed above, SRADS is expected to perform best in models where this assumption is met. There is evidence to suggest that even when the message arrival times cannot be predicted that time slip is not as severe a problem as one might imagine. In [Theo84] it was shown that in the case of queuing networks time slip had little effect on mean value statistics.

SRADS WITH LOCAL ROLLBACK (SRADS/LR)

SRADS, as presented above, requires a process to block at poll times unless it can be determined locally that a writer has progressed at least to the time of the poll event. If a poll must be issued the reader is forced to remain idle until the poll has been answered. Note that the process may receive messages from other writers while it is blocked on the poll. In the original version of SRADS these messages cannot be processed as there is no mechanism to correct any out of sequence computation that may result from this aggressiveness.

SRADS can be modified to allow aggressive processing while a reader is blocked on a poll. To facilitate this aggressiveness a rollback mechanism is employed. The results of aggressive processing are however not sent immediately. The process must wait until it has enough information to guarantee that any results that it passes along will have timestamps that are

monotonically non-decreasing. Thus it cannot process a message aggressively and immediately send the results of the processing if it is possible that another message may arrive at a later time that would cause the computation at the current process to be invalid. Thus any message sent will not, by itself, cause the receiving process to have to rollback (the receiving process may have to rollback if it processes the message aggressively). This guarantees that all rollbacks will be strictly local. The example below should clarify these ideas.

Example 2

Again consider the system shown in figure 1. Assume that process C has polls scheduled to A and B at five time unit intervals. Consider the same scenario discussed in Example 1.

Process C has a poll outstanding to both A and B. Process A responds to the poll and then subsequently sends an event message with a timestamp of nine. Without the local rollback mechanism C must wait to hear from B before it can process the message from A. With the local rollback mechanism C can now go ahead and aggressively process the message from A even though B has not yet responded to the poll. Process C will not, however, send the results of this computation along until it has enough information to ensure that the results will not have to be rolled back. Thus the results will not be sent out until process C can obtain more information as to the logical time of process B.

If process B sends C a message with a timestamp less than nine, then C can rollback to the earlier time and process the messages in the correct order. If process B sends C a response indicating that its logical clock is up to at least time nine, then the results of the aggressive computation can be sent out. If B responds with an acknowledgement showing its clock has advanced to some time between time five and nine, then C must wait for B to progress further in simulation time before it can send out the results of the computation.

Discussion

Note the basic difference between a local rollback mechanism and the more aggressive Time Warp. With a local rollback mechanism all rollbacks are strictly local and thus there are no cascading rollbacks. In Time Warp messages based on aggressive processing are sent out to other processes. If it is discovered that the computation is incorrect an anti-message is sent to cancel the original message. This anti-message may cause the receiving process to rollback which may in turn have to send its own anti-messages causing further rollbacks. In SRADS/LR a message that is sent will never have to be cancelled at a later point. Thus there are no cascading rollbacks in this approach. An interesting research question is how a local rollback scheme will perform compared to Time Warp.

One of the primary advantages of SRADS/LR is that the rollback mechanism is not required to prevent deadlock. SRADS by itself has been proven deadlock free [Reyn82]. This allows for two very important options. First the process can refuse to continue to process aggressively if it is running out of memory. Thus the process can determine locally the amount of memory it will require for state saving. Second, the process can modify its aggressiveness based on a history of the frequencies of rollbacks, for example. It can therefore make local decisions about the advantages of aggressive processing. It can, for instance, decide to continue to aggressively process messages from one source but not from another. These options are not available when the rollback mechanism is required for deadlock prevention.

It should also be noted that in the case where the protocol designer cannot predict potential message arrival times with complete accuracy that the local rollback can help control time slip. In Example 2 it was shown how the local rollback mechanism corrects messages arriving in out of order sequence between poll times.

The local rollback mechanism can, in certain circumstances, also correct time slip caused by a reader aggressively advancing its clock to the next scheduled poll event. Referring to Example 1, time slip can occur when a reader advances it clock to the next poll time and then receives a message in its logical past. The process can back up its clock and accept the earlier message *if* it has not answered a poll itself based on the time to which it had aggressively advanced its time. The reason that a process cannot rollback before a time with which it has answered a poll is that to do so would invalidate information it had sent to another process. This invalidation of information contradicts the basic premise of local rollback: A process only sends information to another process that it will never retract.

PERFORMANCE ISSUES

We present an informal assessment of the expected performance of SRADS/LR as compared to one exhibiting unbounded aggressiveness and risk (unbounded "optimism"). The following variables are essential to an assessment of this difference and are meant to represent the magnitudes of the *differences* in the costs between the two approaches. Each can be considered in units of wallclock time.

- LO: lost opportunity cost. This measures how much could be gained if aggressiveness and risk were unbounded as opposed to the way they are bounded in SRADS/LR.

- DA: deadlock avoidance cost. SRADS/LR employs a polling mechanism as described above. An unbounded approach has none.

- SS: state saving costs. SRADS/LR can control how much state saving is done on a continuous scale from none to as much as is required for unbounded *local* aggressiveness. An unbounded approach must have some state saving and part of the frequency of state saving will be dependent on how imbalanced the system is.

- RC: rollback costs. We expect a decrease in rollback frequency in SRADS/LR relative to the costs in an unbounded approach. This variable includes only those costs local to processes.

- IR: cost of inter-processor rollbacks. SRADS/LR would have none. Unbounded approaches are designed to accommodate them. This variable measures the cost of carrying out interprocessor rollbacks (e.g. an anti-message system as in Time Warp).

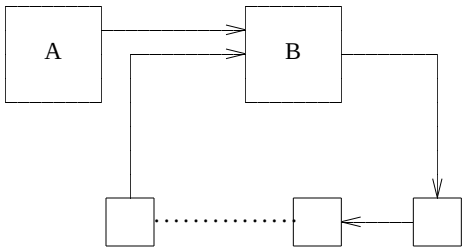

FIGURE 2. Network for Protocol Comparison

SRADS/LR will exhibit better performance when the following inequality holds:

$$LO + DA < SS + RC + IR$$

With sufficient activity level (see [Nico89]) we can argue that DA and IR can be negligible since they are primarily message passing costs and can be overlapped with processing. SS can be considered negligible if we assume a separate state saving device as proposed in [Fuji88]. Thus, the remaining, dominant costs are LO and RC. In figure 2 we demonstrate a network in which RC can be made arbitrarily larger than LO, assuming that state saving is done to secondary storage, thus creating non-negligible state restoration costs.

In figure 2 we assume that process A sends late messages to process B with a high probability, thus creating a low lost opportunity cost. Given this, the difference in rollback costs, RC, is high for the following reason. With SRADS/LR the cost of recovering from a late message from A to B is the cost of doing a rollback at B only. Subsequent messages sent from B into the cyclic subnetwork will have processing costs associated with them, *but no rollback costs*. An unbounded approach will encounter a sequence of rollbacks in the subnetwork, given that B had processed messages from the subnetwork and passed the results of processing them back into the subnetwork. It is likely that some of the rollback activity will overlap with processing in processes that have already rolled back. However, if the cost of event processing is significantly less than the cost of rolling back, the recovery in the subnetwork will be dominated by a linear sequence of rollbacks. Thus, RC will be significant. The difference between RC and LO can be made arbitrarily large by increasing the percentage of late messages from A and/or increasing the number of processes in the subnetwork.

IMPLEMENTATION

We are in the process of implementing SRADS/LR on the SPECTRUM [ReDi89] Testbed at the University of Virginia. SPECTRUM is a testbed for parallel discrete event simulation protocols. The testbed provides a common environment in which to test various simulation protocols. We currently have implemented SRADS and Null Messages as well as many applications. We are in the process of comparing the performance of SRADS/LR to both original SRADS and Null Messages on a variety of applications. Future work includes a comparison with an approach similar to Time Warp.

REFERENCES

[ChMi79] Chandy, K.M. and J. Misra, "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs," *IEEE Trans on Software Engineering.*, SE-5,5, May, 1979, 440-452.

[DaRe83]] Davidson, D.L. and P. F. Reynolds, "Implementation and Performance Analysis of the SRADS Distributed Simulation Protocol" *DAMACS Report No. 83-13, University of Virginia*, December, 1983.

[Fuji88] Fujimoto, R., et al, "The Rollback Chip: Hardware Support for Distributed Simulation Using Time Warp," Proc. of SCS Distributed Simulation Conference, 19,3, Jan 1988, 81-86.

[Jeff85] Jefferson, D., "Virtual Time," *ACM TOPLAS*, 7,3, July, 1985, 404-425.

[Nico84] Nicol, D.M., "Synchronizing Network Performance", Master's Thesis, University of Virginia, March 1986.

[Nico89] Nicol, D.M., "The Cost of Conservative Synchronization in Parallel Discrete Event Simulations", unpublished manuscript, June, 1989.

[O'Hal83] O'Hallaron, D.R., "Analysis of a Model for Distributed Simulation", Master's Thesis, University of Virginia, January, 1983.

[PeWo79] Peacock, J.K., Wong, J.W. and E. Manning, "Distributed Simulation Using a Network of Processors," *Computer Networks*, 3, North Holland Pub., 1979, 44-56.

[Reyn82] Reynolds, P.F. "A Shared Resource Algorithm for Distributed Simulation," *Proc of the Ninth Annual Int'l Comp Arch Conf*, Austin, Texas, April, 1982, 259-266.

[Reyn88] Reynolds, P.F. "A Spectrum of Options for Parallel Simulation Protocols," *Proc of ACM Winter Simulation Conference*, Dec, 1988.

[ReDi89] Reynolds, P.F. and Dickens, P. M., "SPECTRUM: A Parallel Simulation Testbed",*Proc of the 4th Annual Hypercube Conference*, Monterey, Ca., March, 1989.

[ReKu86] Reynolds, P.F. and Kuhn, C.S., "Three Variations on the SRADS Simulation Protocol," *Proc of SCS Eastern Multi-conference*, Orlando, April, 1986.

[Theo84] Theofanos, M. "Distributed Simulation of Queueing Networks", Master's Thesis, University of Virginia, Jan. 1984.