Proceedings of the 1983
Winter Simulation Conference
S. Roberts, J. Banks, B. Schmeiser (eds.)

ACTIVE LOGICAL PROCESSES AND DISTRIBUTED SIMULATION: AN ANALYSIS*

Paul F. Reynolds, Jr.
Department of Applied Math and Computer Science
The University of Virginia
Charlottesville, Virginia 22901

RESEARCH SUMMARY

1. INTRODUCTION

We discuss a deadlock-free distributed simulation algorithm that improves upon the performance and resource requirements of previously proposed algorithms. Our approach is based on the notion of active logical processes. A more detailed discussion of active logical processes can be found in Reynolds (1983).

Distributed discrete event simulation algorithms typically consist of networks of cooperating asynchronous logical processes, where a logical process (LP) is a representation of a corresponding physical process (Chandy, 1979). A distributed simulation algorithm determines how a network of LP's representing an underlying physical system will interact so that the order of events in the underlying physical system is maintained. A given algorithm will determine such key factors as the final completion time (performance), whether the simulation will deadlock (even if the underlying physical system would not), and the resource requirements in the system that supports the simulation. No distributed simulation algorithm proposed to date can be considered optimal for all possible weightings of these factors, and, indeed, such an algorithm may not exist.

Algorithms which preserve the proper sequencing of events throughtout the course of a simulation have been characterized as "conservative" (Jefferson 1982). Conservative algorithms include Bryant's (1977) infinite buffers algorithm, Peacock's (1979) blocking condition and link time algorithms, Chandy's (1979) null messages algorithm and Chandy's (1981) distributed deadlock detection and recovery algorithm. All but the last of these algorithms is deadlock free a priori; the last uses detection and recovery. More recently Jefferson (1982) has proposed the time warp algorithm, which allows LP's to block only when they have no work to do, thus preventing deadlock.

*This research was supported in part by IBM Corporation.

In each of the algorithms mentioned the prevention or detection of deadlock can adversely affect performance. The blocking algorithm requires a potentially high degree of connectivity among processors, and is likely to incur high communication costs (Peacock, 1979). The link time, null message and infinite buffer algorithms allow an LP to communicate only with immediate predecessors about its blocked state, thus reducing communication costs, but adding the requirement that at least one LP in each directed cycle of LP's be able to predict a future interval of time in which the LP will not send a message to any other LP's. The performance of the deadlock detection and recovery algorithm depends on the frequency of deadlocks; a high frequency would adversely affect finsihing time. The time warp algorithm must maintain input and state histories which can grow in proportion to the number of events simulated.

We regard low connectivity and bounds on required resources that are independent of the number of events being simulated as important criteria for measuring the worth of a distributed simulation algorithm. These goals tend to enhance performance as well. The null message and link time algorithms meet these criteria best, but their performance suffers as a result of the frequent transmission of unnecessary messages. The SRADS algorithm (Reynolds, 1982) also meets our stated criteria and it reduces the number of unnecessary messages transmitted. We discuss the expected behavior of this algorithm next.

2. ACTIVE LOGICAL PROCESSES

The link time and null message algorithms employ passive LP's. A passive LP is driven by messages from other LP's. Until it receives these required messages it is blocked. The generation of null messages and predictions is done by LP's for each of the links that they may write to, whether or not the LP's that read from those links need them. The result is the generation and transmission of many unnecessary messages. See (Chandy, 1981).

Active logical processes (ALPs) employ an alternate view: a writing LP should generate a prediction or a null message only when a reader indicates that it is blocked. Also, when feasible, readers should perform predictions in order to spare the cost of having writers generate and communicate them. We consider these alternatives.

An ALP-based network consists of LP's and shared facilities (SF's), where an SF is a single cell repository for messages. LP's communicate exclusively through SF's; any ALP-based network can be represented as a bipartite graph of LP's and SF's. An LP is connected to that set of SF's from which it can read or write. In order to read from or write to an SF, an LP must access it.

Consider the following sequencing requirement which has been derived from (Lamport, 1977). A write (read) access to SF by LP_i must satisfy:

SR: For any reader (writer), LP_j, connected to SF, it is required that $C_i \leq C_j$, where C_i is LP_i's current simulation time.

Implicit in this algorithm is the assumption that an LP can predict the appropriate times to read from an SF. This is the case when exact write times, or exact potential write times, are known. If exact write times are known, all intermediate synchronizations (unnecessary messages) at potential write times are discarded, leading inevitably to improved performance. However, even if a reading LP must "poll" at potential write times, whenever the reading LP applies SR and finds the simulation time of the writing LP ahead of its own, it can discard all polls in the interval up to the writing LP's simulation time. We have determined both in analytic and in simulation studies that a significant number of polls can be discarded, leading to a significant improvement in performance. In effect we have found a method for discarding a significant number of the unnecessary messages that would be transmitted in the null message and link time message algorithms.

If potential write times cannot always be predicted then the algorithm given above will not always give correct results; it is possible for an LP to read from an SF at a simulation time that is greater than the simulation time at which the last write was performed. This time slip can result in an LP simulating events out of sequence. To prevent this we must put the burden of prediction onto the LP's that write into these SF's.

For those SF's for which a reading LP cannot predict potential write times, we can require the reading LP to request future potential write times from writing LP's. By placing the burden of synchronization on reading LP's we have created once again the potential for many predictions and many potential synchronizations to be skipped. We have compared the null message and link time algorithms with this new method and found that the new method has smaller finishing times unless writing LP's write very frequently and at every potential write time.

Finally, it is possible to combine the two approaches we have proposed. Clearly, the approach where all predictions are incorporated into reads yields better performance than the approach where all writers are called upon to make predictions. However, the combined approach yields performance that falls between the performance of the other two. Thus, if we must employ the combined approach, we know we can get improved performance over both passive and active LP algorithms that require writers to perform all predictions, while still guaranteeing the proper sequencing of events (i.e. a correct simulation). We have found the combined approach to be useful for the simulation of networks of logic.

3. CONCLUSION

We have investigated placing the responsibility for synchronization in LP's at the point where they need to read a message from another LP. If the element of prediction can also be incorporated at the point where reads are performed then significant improvements in simulation finishing times can be realized. Even in the case where such predictions cannot be made by a reading LP, simply allowing predictions to be made on demand, rather than as a matter of routine, can lead to performance improvements. We have demonstrated that our approach is superior, on the average, to the link time and null message algorithms. Lacking data on the performance of other approaches, we can draw no further conclusions about comparative performance.

REFERENCES

Bryant RE (1977), Simulation of Packet Communication Architecture Computer Systems. MIT/LCS/TR-188, MIT, Cambridge, Mass.

Chandy KM, Misra J (1979), Distributed Simulation: A Case Study in Design and Verification of Distributed Programs, IEEE Transactions on Software Engineering, SE-5, pp. 440-452.

Chandy KM, Misra J (1981), Asynchronous Distributed Simulation Via a Sequence of Parallel Computations. Comm ACM, 24,4, pp. 198-206.

Jefferson D, Sowizral H (1982), Fast Concurrent Simulation Using the Time Warp Mechanism, Part I: Local Control. A Rand Note, N-1906-AF.

Lamport L (1978), Time, Clocks, and the Ordering of Events in a Distributed System, Comm ACM, 21,7, pp. 558-565.

Peacock JK, Wong JW, Manning E (1979), Distributed Simulation Using a Network of Processors. Computer Networks, 3, North Holland Publishing Co., pp. 44-56.

Reynolds PF (1982), A Shared Resource Algorithm for Distributed Simulation. In: Proc. of the Ninth Annual Int'l. Computer Architecture Conference, Austin, Texas, April, pp. 259-266.

Reynolds PF (1983), The Active Logical Process Approach to Distributed Simulation. DAMACS Technical Report # 83-12. Dept. of Applied Math and Computer Science, Univ. of Virginia, Charlottesville, Virginia.