Proceedings of the 1984
Winter Simulation Conference
S. Sheppard, U. Pooch, D. Pegden (eds.)

ASYNCHRONOUS SIMULATION OF SOME DISCRETE TIME MODELS

Jay B. Ghosh
Decision Sciences, University of Dayton
Dayton, OH 45469

ABSTRACT

Traditionally, simulations are carried out in a synchronous manner in that most simulators advance their clocks from the time of one event to that of the most imminent event in order to execute the simulations. This synchronization is achieved at the expense of maintaining a data-structure called the event-list. An alternative to this situation is asynchronous simulation. Asynchronous simulators do not always process events in their natural order of occurence. The event-list and the associated costs may thus be eliminated. While asynchronous simulation has found considerable exposure in the literature of distributed processing, little attention has been paid to assess its applicability in the conventional environments. Some research in the past have considered simulating without the event-list. But such research have inevitably lacked generality. This paper intends to broaden the scope of asynchronous simulation in the conventional environment. It does so by presenting asynchronous solutions to a set of disparate problems and by hoping that the commonality underlying these solutions shall be captured and eventually lead to a general framework for simulation.

INTRODUCTION

The word "asynchronous" in the context of discrete event simulation has dual connotations. Some (viz., Bratley et. al. [1], p. 16-17) prefer it to mean variable time-increment procedures as opposed to "synchronous" or fixed time-increment procedures for simulation control. Others (viz., Chandy and Misra [2]) maintain that "asynchronicity" comes about when within phases of execution of a simulation parts of the simulator are allowed to be far ahead of others in simulated time. It should be evident that according to this posture both procedures of conventional simulation are inherently "synchronous". This paper will subscribe to the latter view in its use of the words "synchronous" and "asynchronous", and from this point on, will abandon the quotes surrounding them.

BACKGROUND

There are three prevalent world views for conventional discrete event simulation. They differ from each other on the basis of their individual orientations: activity, event, and process (Kiviat [5].) Though these different orientations warrant differences in implementation, synchronicity remains essestial to all of them. This is normally achieved through the maintenance of a clock and a list of pending events. The cost associated with processing of the list is often prohibitive. Over the years, a number of data-structures have been advocated for this list (McCormack and Sargent [6]), but the debate about their optimality still continues. Of interest is a recent paper by Comfort [3].

Early departures from synchronous simulation and the list of events can be found in Hordijk et. al. [4] and Norton [8]. Norton provides simulation solutions to a class of problems in the asynchronous sense. Nance [7] puts the issue of such solutions in perspective, calling them state-sequenced simulations. More recently, Bratley et. al. [1] suggests (p. 263-264) in passing that it may be profitable to simulate using local clocks and local list of events in those parts within the structure of a simulated system where the processes do not interfere with each other. To some extent this paper reflects similar thoughts.

In contrast to conventional simulation, the idea of asynchronicity comes quite naturally to distributed simulation. A good amount of work on asynchronous simulation is reported in its literature, notably that by Chandy and Misra [2].

CONCEPTS AND ISSUES

This paper does not present a unified methodology for asynchronous simulation. It considers, on the other hand, a number of disparate problems and shows how these problems can be solved in an asynchronous manner. There is a typical theme to all the solutions in that every simulated system is conceived as a network of blocks through which entities, generated internal or external to the system, may traverse over time.

Asynchronous simulation does not require an event-list. A global clock is not necessary either. Normally, clocks are assigned to the various blocks and are maintained locally. Entities are also allowed to carry their own clocks.

This does not imply, however, that an entire simulation can be carried out asynchronously. The degree to which asynchronicity can be effected depends on the attributes of the system under study and the objective of the simulation experiment. The system attributes are the structure of the system, the protocals governing the dynamics of the system, and the level to which resources are shared across various processes comprising the system. In particular, pure serialism would hardly warrant any synchronization whereas any parallelism would inevitably demand it. Resource sharing, pre-emptive entities, and some queue disciplines would also force asynchronous processing to degenerate considerably. Objectives of a simulation experiment concern statistics collection and simulation run control. These tasks assume nontrivial proportions in asynchronous simulation.

In the next section a number of simulations are handled asynchronously. The examples have been chosen to highlight the attractiveness of the asynchronous approach as well as its limitations. Most of the major issues mentioned above are addressed.

SOME EXAMPLES

Norton [8] introduced the concept of entity-based processing which, in her case, relied on the state-space representation of systems. This paper resorts to a similar approach, but is far less formal and much more intuitive. Emphasis is placed on resolving situations where an approach like Norton's would either become too complicated or simply fail. The examples follow.

Single-server Queues

A single-server queue (fig. 1) is considered. Queue discipline of first-in-first-out (FIFO) is assumed. This system has the nice property that preserves the

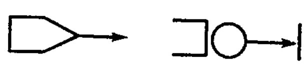

Fig. 1.

order in which entities enter the system even as they exit. It is thus possible to simulate the system from one entity to the next. The departure time for an entity is simply the maximum of its arrival time and the departure time of the preceeding entity plus its service time. Going back to fig. 1 it may be instructive to think of the arrival time as the local clock time at the source and the departure time as the local clock time at the queue.

Single-server Queues in Tandem

The same approach can be extended to a tandem of single-server queues (fig. 2). Here an entity arriving

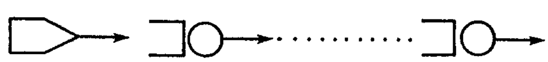

Fig. 2.

at a queue simply checks its own clock with that of the queue. Its departure time is computed like before to reflect whether it has to wait or not.

Multiple-server Queues

The situations considered above have been purely serial. A multiple-server queue (fig. 3) brings in parallelism. Entity-based processing collapses as that much

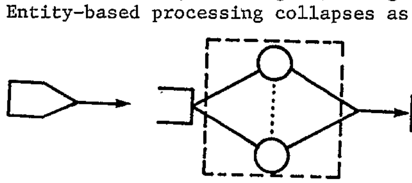

Fig. 3.

desired ordering of entities according to their arrival times is no longer maintained. Synchronization becomes necessary immediately following service. This can be done by using the following steps.

i. Keep an ordered list of service completion times.
ii. Check arrival time of entity with the minimum of service completion times. If arrival time is greater than or equal to the minimum time, then the entity does not have to wait. Otherwise it does. Compute the corresponding service completion time and post in the list.
iii. Identify the entity with the minimum service completion time and release it for further processing.
iv. Note that if there are m servers, the first entity will be released only after m service completion times have been computed.

Queues with Feedback

Feedback like parallelism hampers the performance of entity-based processing. Synchronization is required as entities with different times contend to enter the same queue (fig. 4). This can be ensured if the fol-

Fig. 4.

lowing is done.

i. Process an entity all the way along the feedback loop to the queue.
ii. Hold it there until all entities arriving from the source are processed.

Combination of Queues

Combination of single-server and multi-server queues (fig. 5) require occasional synchronization. Entities

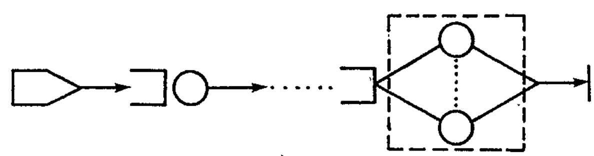

Fig. 5.

are processed until they cannot progress any further i.e., they either reach a sink or are "blocked". Notice that the "blocking" mentioned here is caused by a time condition rather than a state condition. A "blocked" entity is unblocked at a later time by some other entity arriving at the point of the "block".

Non-FIFO Queue Disciplines

Non-FIFO queue disciplines like LIFO (last-in-first-out) and SIRO (service-in-random-order) destroy the arrival order of entities. Synchronization is thus called for. A scheme for LIFO queues is given below.

i. "Block" any entity that has to wait at the queue.
ii. If an entity does not have to wait, check the LIFO queue. If queue is empty proceed with current entity. If not, "block" it and "unblock" all entities that can enter service before it.

SIRO works very much the same way except that the arriving entity gets "blocked" and an entity is randomly picked from the queue and released for further processing.

Resource Sharing

Resources are global variables that may couple two or more processes. Asynchronous processing without regard to appropriate value of these variables may impair the integrity of a simulation. One solution is to tag each resource with a time or a list of times. On other occasions, it may be necessary to bring about synchronization between processes sharing the same resource.

Statistics Collection

Statistics collection on observation-based variables (e.g., time in queue) is straight forward in asynchronous simulations. Not so with time-based variables (e.g., number in queue). The problem can be handled by keeping past information at each process. On arrival of an entity at a process these information can be scanned to develop the necessary statistics. The same information can also be used if state-conditioned processing is required. The above solution holds for local collection of statistics. Global collection (e.g., number in system) of statistics may need some synchronization as well.

Terminating a Simulation

Simulations are normally stopped after a specified number of entities have been processed or a specified time has been reached. Terminating an asynchronous simulation based on time needs synchronizations both at the source and the sink ends of the simulated system. Termination criteria other than number of entities and time may also require some synchronization so that statistics are collected properly.

CONCLUSIONS

As demonstrated in the examples above, asynchronous solutions can be found for a wide variety of problems. But the strategies employed so far have been specific to problems. For the asynchronous methodology to be a viable alternative to its synchronous counter-part this problem-dependence must be removed. Experience suggests that this may not be very difficult to achieve at least for systems without complex interactions. Adapting asynchronous methods to general problem solving remains the task for future research.

ACKNOWLEDGEMENTS

The research leading to this paper was partially supported by a small grant from the School of Business Administration at the University of Dayton.

REFERENCES

1. Bratley, P., Fox, B. L., and Schrage, L. E., A Guide to Simulation, First edition, Springer-Verlag, New York, 1983.

2. Chandy, K. M. and Misra, J., "Asynchronous distributed simulation via a sequence of parallel computations", Comm. ACM, Vol. 24, Iss. 4, 198-206, April, 1981.

3. Comfort, J. C., "The simulation of a Master-Slave event set processor", Simulation, Vol. 42, Iss. 3, 117-124, March, 1984.

4. Hordijk, A., Iglehart, D. L., and Shassberger, R., "Discrete time methods for simulating continuous time Markov chains", Rept. No. 35, Dept. of Operations Research, Stanford Univ., Stanford, California, 1975.

5. Kiviat, P. J., "Digital Computer Simulation: Computer Programming Languages", Memo. No. RM-5883-PR, Rand Corporation, Santa Monica, California, 1969.

6. McCormack, W. M. and Sargent, R., "Analysis of future event set algorithms for discrete event simulation", Comm. ACM, Vol. 24, Iss. 12, 801-812, December, 1981.

7. Nance, R. E., "The time and state relationships in simulation modeling", Comm. ACM, Vol. 24, Iss. 4, 173-179, April, 1981.

8. Norton, S. R., "Schedule independent simulation: a new technique of discrete event simulation suitable for small computers", Proc. Winter Simulation Conference, vol. 2, 499-508, 1976.