Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Heterogenous distributed simulation

Paul F. Reynolds, Jr.
Department of Computer Science and
Institute for Parallel Computation
The University of Virginia
Charlottesville, Virginia 22903

ABSTRACT

Simulations can often be substituted for expensive or difficult experiments involving real physical resources. Heterogenous distributed simulation, a fairly recent phenomenon, is an example of simulation made possible by low cost, high performance processors and medium-to-high-bandwidth, (potentially long distance) communication networks. Heterogenous distributed simulation brings together heterogenous, physically distributed resources for simulation purposes, thus supporting experiments that could otherwise be expensive or difficult. Heterogenous distributed simulation is sufficiently new that many issues regarding its use remain. These issues include fault tolerance, dynamic reconfiguration, real time requirements and optimal approaches to process synchronization and communication. We discuss these issues and others and propose some directions for research.

INTRODUCTION

Distributed simulation, sometimes called heterogenous or geographically distributed simulation, is distinguished from *parallel simulation* (to confuse things, often called distributed simulation in the literature) on the basis of inter-process communication times and goals for employing multiple processors. Distributed simulations tend to incur communication delays on the order of seconds, and are generally employed for the purpose of bringing physically separated, heterogenous resources together for simulation purposes. Examples include SIMNET [PoMi87] and the National Testbed [Word88]. Distributed simulations may not be concerned with processor utilization or minimum finishing time, although real-time requirements may bring these into play. Parallel simulation research, on the other hand, has focused primarily on maximizing processor utilization and/or minimizing simulation finishing time. Inter-processor communication times have generally been presumed to be relatively small (e.g. ~ milliseconds). Parallel simulation research has not been concerned with real-time issues, human-in-the-loop or hardware-in-the-loop.

Examples of approaches to parallel simulation include the pioneering work of Chandy and Misra [ChMi79] and Peacock, et al. [PeWo79] as well as SRADS [Reyn82], and time warp [JeSo82].

Distributed simulation, as defined here, is barely a science. There are critical issues revolving around it that simply have not been addressed in the literature. There are systems such as SIMNET, but they have been developed without the benefit of an underlying body of theory and without the benefit of testbeds for studying ideas for synchronization, fault tolerance or reconfiguration.

Parallel simulation is in its infancy. It is characterized by isolated efforts to develop synchronization protocols (the papers cited above are products of this sort of research). A recent paper [Reyn88] demonstrates that parallel simulation researchers have only begun to explore all of the possibilities, and that without analytic and rapid experimental support, the task could could be monumental at best.

Distributed and parallel simulation have more in common than not. The demands placed on each will become demands placed on the other as scientists turn to them as the means for supporting low cost experimentation; e.g. real time requirements are becoming a major consideration in parallel simulations. An exploration of these two areas together will lead to a better understanding of each of the areas alone.

In the remainder of this paper (and the attendant tutorial) we make the concept of heterogenous distributed simulation clearer by reviewing design options, and we discuss issues facing those who would wish to use distributed simulation based on sound scientific principles. Because of the similarities between distributed and parallel simulation, we also discuss the relevance of parallel simulation research to distributed simulation research.

DEFINITIONS

Timestepped simulations assume that simulation time advances in (generally) fixed increments, where the chosen increment is sufficiently small that no two significant events that should occur at distinct simulation times are simulated in the same time interval. *Discrete event simulation* is suitable for simulations which incorporate event sequences that occur on widely disparate time and/or space scales.

Real time (a.k.a. *faster than real time*) simulations are required to run at least in real time. There are a number of interpretations of this requirement. One is that a simulation must at all times not fall behind real time. A second is that a simulation must reach certain milestones without falling behind. This latter interpretation allows for lapses between milestones in the event processing requirements momentarily outpace processing resources. Two activities associated with simulation, distributed simulation in particular, that are likely to make it have real time requirements are human-in-the-loop and hardware-in-the loop. As their names imply, this means the simulation incorporates a real time process (human and/or hardware) and to meet that process's requirements the simulation itself must be real time.

Fault tolerance has its usual meaning from software engineering literature. If there is a failure in the system can the system recover and proceed, perhaps at reduced capability? We shall see that fault tolerance is important because distributed simulations involving human- and/or hardware-in-the-loop are often unrepeatable, at least without great expense. Fault tolerance under these conditions is mandatory.

Load balancing is concerned with load sharing in order to meet performance requirements. In distributed simulation, performance requirements are likely to be real time requirements. Load balancing takes on new meaning when communication times are on the order of seconds, as they may be in distributed simulation.

DESIGN CHOICES

We review here the choices a designer of a distributed simulation may be able to make. In some cases some of the choices may be dictated by the application, for example real time requirements.

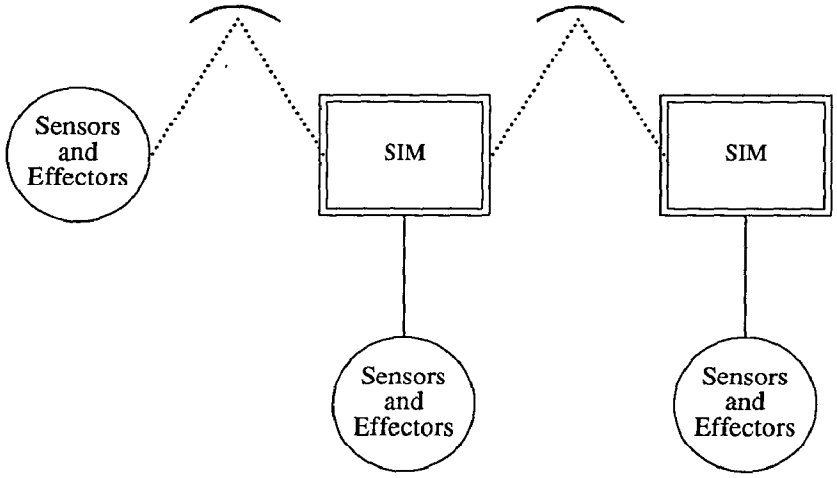

Figure 1. Example of Distributed Simulation

In many instances we refer to local and remote sites. In figure 1 the two entities labeled "sim" are assumed to be pieces of a distributed simulation, each located at a specific site. The dashed communication line between them represents a (relatively) slow communication link; e.g. satellite communications. Sensors and effectors are optional, and can be represented by hardware and/or humans. As shown in figure 1, communications between sensors and effectors and local sites may be either high speed or low speed, depending most likely on the remoteness of the sensors and effectors from the site.

Real time heterogenous distributed simulations can be a challenge, particularly if pieces of the simulation are geographically distant, or if their communication link time is on the order of seconds. Real time takes on a modified meaning here: consider the case where we have two interacting humans at physically remote sites. Each of them interacts with a local piece of the distributed simulation. If one of them acts, it could be a half of a second or more, using current satellite communication links between the sites, before the second human has any idea that the first one acted. Clearly, "real time" has a modified meaning here if the half second delay is not representative of the delay that would occur between the two humans in the physical system being simulated. Thus, when we say that real time may be a design choice, attendant with that choice is compensation for those instances where lapses occur in the maintenance of a real time rate in the simulation because of communication delays.

Dead-reckoning is one option used to cope with this phenomenon; i.e. local sites predict behavior of remote entities (simulated or real) based on past history. It is then the responsibility of the remote sites to inform others of changes that would make dead-reckoning inaccurate (in accordance with some threshold). This technique is descirbed in [PoMi87].

A choice of timestepped vs discrete event time management in a distributed simulation is another design choice. The choice is not dichotomous; combining the two, some sites doing one type of management and others doing the other, is entirely feasible. There are times when timestepped simulations are appropriate. This includes cases where the generally low overhead of the timestepped approach may make it a better method than an equivalent discrete event simulation. Discrete event simulations are clearly more appropriate in cases where a timestepped approach would have many intervals where no significant events occurred.

We must view time management from at least two perspectives: what is done at individual sites and what is done among sites. We see no restrictions on the choices made, e.g. discrete event at local sites but timestepped among sites, etc. The choice would be dictated by characteristics of the simulation at individual sites, including the kind of computations being performed, and whether processing at a local site is sequential or parallel. Similarly, the choice of time management technique among sites could be dictated by the presence or absence of real time requirements, communication bandwidth, etc.

It is in the area of time management that the research done in parallel simulation can be most beneficial to distributed simulation. An extensive amount of work has been done in parallel simulation on the problem of ensuring that a discrete event parallel time management algorithm is deadlock-free. Also, there is an emerging understanding of what is required to make parallel simulation algorithms work reasonably efficiently. Where one of the significant challenges lies is in the determination of the impact of significant communication times on algorithms that have heretofore assumed relatively small communication times.

Fault tolerance is a choice that may or may not be dictated by the application being simulated. Fault tolerance would addres the issue of what to do if, for example, a sensor in a distributed simulation failed, or if the communication link between a sensor and the portion of the simulation receiving its output failed, say, intermittantly. Fault tolerance could be desirable simply because of the length of time the simulation requires to execute. We can imagine some remotely connected sites cooperating on a common (distributed) simulation which requires more time than the expected mean-time-between-failure for the aggregation of processors involved in the simulation. In this latter case, check-pointing is a reasonable option, if the characteristics of the simulation are such

that the possible rollback that goes with checkpointing is feasible.

Load balancing is perhaps the design choice that is least determined by the application being simulated. There are often alternatives to load balancing, e.g. adding more processing power at bottlenecks. Recently, there have been a number of papers on dynamic load balancing, mostly heuristics for performing it. There is much to be learned from load balancing experiments done for parallel simulation.

Distributed simulation poses new challenges for dynamic load balancing, primarily because the large communication costs typically incurred by distributed simulations keep code or data transfers for balancing purposes from being cost effective. A good dynamic load balancing strategy would most likely be a distributed one, where individual sites monitored some subset of the other sites, and acting on knowledge of what those sites had to do, picked up some of their load if they were observed to have fallen behind. The distributed nature of this algorithm comes in part from the requirement that the processor being helped would have to recognize that another processor had taken a portion of its load; all this to avoid the long communication delays that would be incurred if a more cooperative effort were employed.

Both fault tolerance and dynamic load balancing may require dynamic reconfiguration of a distributed simulation. As with fault tolerance and load balancing, reconfiguration would work best if a minimum of inter-site communications were required.

Dynamic reconfiguration may also be required as a result of dynamic reconfiguration of the simulation itself. For example, it may be planned that a piece of a simulation may "drop out" part way through. If there are other sites dependent on its presence, then reconfiguration is called for.

In addition to the choices outlined above, most of the design variables described in [Reyn88] apply as well. For example, a distributed simulation could exhibit aggressive processing, in the sense that local sites make assumptions about remote sites in order to allow their own processing to advance. To a degree, the dead-reckoning approach used in [PoMi87] is an example of aggressiveness. Similarly, partitioning - using different time management techniques at different sites - is another option.

We have outlined some of the design choices the designer of a distributed simulation may face. These choices, in turn, reflect a broader need to understand distributed simulation well enough to know what choices should be made under what conditions. In the next section we discuss research issues that underlie the design choices we have identified.

SOME ISSUES

There is a need for research in the following areas if we are to develop a good understanding of how to design efficient and correct distributed simulations.

(1) Development of underlying formal models to, among other things, keep a rein on an unmanageably large number of possible approaches. The number of possible design choices is sufficiently large that we need models to identify those choices that would not meet application-specific requirements.

(2) Development of testbeds for testing the approaches predicted as viable by the models. It would be best to avoid the direction research has taken in parallel simulation, namely, the development and isolated analysis of parallel simulation algorithms (protocols). A testbed allows for the testing of different approaches in a *common* environment, which is the best way to do comparative analyses.

(3) Gaining an understanding of the interplay between the somewhat different areas: distributed simulation and parallel simulation. As we have indicated above, the two areas have much in common. Many of the load balancing and protocol analysis results done for parallel simulation are applicable to distributed simulation as well.

(4) Integration of real time requirements. Given sufficient processing resources, this can be relatively simple. However, by nature of their characteristically long communication times, distributed simulations must compensate for delays that would not be present in underlying physical systems being simulated.

(5) Understanding dynamic reconfiguration and load balancing. We can approximate the problem to be considered in the following manner: for a set of processing resources, a set of communication delays between processing resource pairs, a set of reconfigurations of the processing resources, and a set of constraints on the reconfigurations, can we determine if a given simulation can meet its performance requirements? For a given simulation state, what is the optimal partitioning considering anlysis and reconfiguration costs?

(6) Fault tolerance. Many distributed and parallel simulations are real time, one-shot simulations. Users would pay a lot for reliability and fault tolerance, because the price of a failure could be very high. What are effective fault tolerance strategies? Fault tolerance is often implemented using redundancy. That could be expensive or impossible with distributed simulation in the sense that "hot spares" may not be available. Without redundancy of this sort, what other options exist?

CONCLUSIONS

This paper establishes the framework for our tutorial in that it identifies issues we perceive as relevant to the successful understanding of the science of doing distributed simulation. The issues and design choices outlined here indicate the areas we view as the ones requiring a better understanding before distributed simulation will be well understood. We expect the tutorial to probe more deeply into the issues and choices identified here, as well as to identify some new areas.

ACKNOWLEDGMENTS

This research was supported in part by the Department of Defense, through the Jet Propulsion Laboratory, contract number 957721.

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.

[FJLO88] Fox, G., et al., "Solving Problems on Concurrent Processors," Prentice Hall, 1988.

[HaDo88] Hartrum, T.C. and B.J. Donlan, "Distributed Battle-management Simulation on a Hypercube," *Proc., SCS Multi-conference,*, San Diego, CA, Feb., 1988.

[JeSo82] Jefferson, D. and H Sowizral, "Fast Concurrent Simulation Using the Time Warp Mechanism," *A Rand Note*, N-1906-AF.

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

1985, 404-425.

[Misr86] Misra, J., "Distributed Discrete Event Simulation," *ACM Computing Surveys,* 18,1, March, 1986, 39-65.

[PeWo78] 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.

[PoMi87] Pope, A.R. and D.C. Miller, "The SIMNET Communications Protocol for Distributed Simulation," BBN Technical Report, BBN Laboratories Incorporated, Cambridge, MA, 1987.

[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, Jr., P.F. "A SPECTRUM of Options for Parallel Simulation", *Proc. ACM Winter Simulation Conf.*, San Diego, CA., Dec 1988.

[Word88] Worden, J. "National Testbed Program," *Proc of SCS Multi-Conference: Aerospace Simulation III,*" February, 1988, San Diego, CA.

BIOGRAPHY

PAUL F. REYNOLDS, JR. is an Associate Professor of Computer Science, as well as the Director of the Institute for Parallel Computation at the University of Virginia. He is a member of the Simulation Engineering Group, an oversight group for the National Testbed. His research interests include parallel and distributed simulation, and, in general, parallel language and algorithm design. He has been a consultant to numerous corporations and government agencies in the systems and simulation areas. His Ph.D is from the University of Texas, 1979.

Institute for Parallel Computation
Thornton Hall
The University of Virginia
Charlottesville, VA 22901
(804) 924-1039
pfr@uvacs.cs.virginia.edu