Distributed Systems Project Report

Distributed Simulation

Members:
Arpit Kumar
Sachin Kumar
Siddharth Rakesh
Manav Sethi

Supervisor:
Prof. Arobinda Gupta

April 14, 2015

Contents

1 Introduction

Simulation is the imitation of the operation of a real-world process or system over time. The act of simulating something first requires that a model be developed; this model represents the key characteristics or behaviors/functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time.

Computer Simulations of real world process provides a means to test out different strategies in order to determine which will be the most effective. Modeling complex real world systems has become very essential in many fields especially in engineering, health, math, military, social and telecommunication sciences. It is very low cost method of information gathering before decision making.

Computer Simulation are primarily of two types,

Continuous Simulation: Continuous Simulation refers to a computer model of a physical system that continuously tracks system response according to a set of equations typically involving differential equations. Typical use cases include rocket trajectory mapping, electrical circuits and robotics.

Discrete Event Simulation: A discrete-event simulation (DES), models the operation of a system as a discrete sequence of events in time. Each event occurs at a particular instant in time and marks a change of state in the system. Between consecutive events, no change in the system is assumed to occur; thus the simulation can directly jump in time from one event to the next.

DES consists of a collection of techniques which, when applied to the study of a discrete event dynamical system, generates sequences called sample paths that characterize its behavior.

Computer simulations are very attractive as they can compress time simulating years of activity in a matter of minutes or even seconds.

They also help us to replicate experiments. To replicate means to rerun an experiment with selected changes in parameter values or operating conditions made by the investigator.

Despite the sophisticated techniques used, some simulation may take hours to complete in real time due to limited availability of resources (processor or memory) in a single computer. This leads to slow simulations while the decisions may be required to be made in minutes in some cases.

To deal with this parallel and distributed simulations came into being.

2 Parallel/Distributed Simulations

This is a technology that enables a simulation program to be executed on parallel/distributed systems, namely systems composed of multiple interconnected computers. As the definition suggests, there are two key components to this technology (1) simulation and 2) execution on parallel or distributed computers. Although parallel and distributed simulations differ from each other in many regards they offer similar benefits. Parallel simulation executes on a set of computers confined to a single cabinet or shelf whereas distributed simulation executes on systems that are geographically distributed across a building, a university campus or even the world.

Following are the benefits of executing a program on multiple computers:

- **Faster Execution**: By subdividing a large simulation computation into many sub-computations, and executing the sub-computations concurrently across, say, ten different processors, one can reduce the execution time up to a factor of ten.

  In computer simulations it may be necessary to reduce execution time so that an engineer will not have to wait long periods of time to receive results produced by the simulation. Alternatively, when used to create a virtual world into which humans will be immersed, multiple processors may be needed to complete the simulation computation fast enough so that the simulated world evolves as rapidly as real life. This is essential to make the computer-generated world "look and feel" to the user just like the real thing

- **Geographical Distribution**: Executing the simulation program on a set of geographically distributed computers enables one to create virtual worlds with multiple participants that are physically located at different sites. Participants in such a simulation can interact with each other as if they were located together at a facility at a single site, but without the time, expense, and inconvenience of traveling to that site.

- **Integrating simulators that execute on machines from different manufacturers**: If we have multiple simulators designed by different manufacturers, rather than porting these programs to a single computer, it may be more cost effective to "hook together" the existing simulators, each executing on a different computer, to create a new virtual environment. Again, this requires the simulation computation to be distributed across multiple computers.

- **Fault tolerance**: Another potential benefit of utilizing multiple processors is increased tolerance to failures. If one processor goes down, it may be possible for other processors to pick up the work of the failed machine, allowing the simulation computation to proceed despite the failure. By contrast, if the simulation is mapped to a single processor, failure of that processor means the entire simulation must stop.

2.1 Underlying technologies

- The first key ingredient is an inexpensive computer, thereby making systems composed oftens, hundreds, or even thousands of computers economically feasible.

- *Inter-computer communications.* There are two flavors of technology at work here. On the one hand, high-speed switches enable one to construct systems containing tens to hundreds or even thousands of processors that reside within a single cabinet or computer room. On the other hand, advances in fiber optics technology is fueling a revolution in the telecommunications industry, making possible computing systems distributed across continents. These advances enable one to consider developing computer applications utilizing many geographically distributed machines.

- *Modeling and simulation.* The final ingredient are technologies to enable construction of models of actual or envisioned real-world systems that can (1) be represented in the internal storage of a computer, and (2) be manipulated by computer programs to emulate the evolution of the actual system over time.

Distributed Simulation over Parallel Simulation

Distributed computers cover a much broader geographic area. Their extent may be confined to a single building, or may be as broad as across an entire nation or even the world. Unlike parallel computers, each node of a distributed computer is usually a stand-alone machine that includes its own memory and I/O devices. Commercial off-the-shelf personal computers or engineering workstations, often from different manufacturers, are usually used.

2.2 Notions of time in simulation

There are several different notions of time that are important when discussing a simulation.

- *Physical time* refers to time in the physical system.

- *Simulation time* is an abstraction used by the simulation to model physical time.

  It is defined as a totally ordered set of values where each value represents an instant of time in the physical system being modeled. Further, for any two values of simulation time T_1 representing physical time P_1 and T_2 representing P_2, if $T_1 < T_2$, then P_1 occurs before P_2, and $(T_2 - T_1)$ is equal to $(P_2 - P_1) * K$ for some constant K. If $T_1 < T_2$, then T_1 is said to occur before T_2, and if $T_1 > T_2$, then T_1 is said to occur after T_2.

  The linear relationship between intervals of simulation time and intervals of physical time ensures durations of simulation time have a proper correspondence to durations in physical time

- *Wall-clock time* refers to time during the execution of the simulation program.

  A simulation program can usually obtain the current value of wall-clock time (accurate to some specifiable amount of error) by reading a hardware clock maintained by the operating system.

3 Conservative Synchronization Algorithms

The physical system being modeled is viewed as being composed of some number of physical processes that interact in some fashion. Each physical process is modeled by a logical process (LP), and interactions between physical processes are modeled by exchanging time-stamped messages between the corresponding logical processes. The computation performed by each LP is a sequence of event computations, where each computation may modify state variables and/or schedule new events for itself or other LPs.

But, to ensure proper execution, Each logical process must process all of its events, both those generated locally and those generated by other LPs, in time stamp order. Failure to process the events in time stamp order could cause the computation for one event to affect another event in its past, clearly an unacceptable situation.

Errors resulting from out-of-order event processing are referred to as causality errors, and the general problem of ensuring that events are processed in a time stamp order is referred to as the *synchronization problem.*

Conservative synchronization protocols ensure that each LP strictly avoids processing events out of time stamp order. In the next section, we describe an alternative approach called *optimistic synchronization* where errors are detected during the execution, and some mechanism is used to recover from them.

Synchronization algorithms do not need to actually guarantee that events in different processors are processed in time stamp order but only that the end result is the same as if this had been the case.

The state of the entire simulator must be partitioned into state vectors, with one state vector per LP. Each logical process contains a portion of the state corresponding to the physical process it models, as well as a local clock that denotes how far the process has progressed in simulation time.

A conservative algorithm solves the problem of when it is "safe" to execute an event. More precisely, If a process contains an unprocessed event E with time stamp T (and no other with smaller time stamp), and that process can determine that it is impossible for it to later receive another event with time stamp smaller than T , then E is said to be safe because one can guarantee that processing the event now will not later result in a violation of the local causality constraint.

3.1 A Deadlock Avoidance Conservative algorithm

Let us assume that one statically specifies the links that indicate which logical processes may communicate with which other logical processes. Further assume that (l) the sequence of time stamps on messages sent over a link is non decreasing, (2) the communications facility guarantees that messages are received in the same order that they were sent (software to re-order messages is necessary if the network does not guarantee this property), and that (3) communications are reliable (i.e., every message that is sent is eventually received). This implies that the stream of messages arriving on a given link will have non-decreasing time stamp values. It also guarantees that the time stamp of the last message received on an incoming link is a lower bound on the time stamp of any subsequent message that will later be received on that link.

Because messages in each FIFO queue are sorted by time stamp, the LP can guarantee adherence to the local causality constraint by repeatedly processing the message containing the smallest time stamp, so long as each queue contains at least one message If one of the FIFO queues becomes empty, the LP must wait until a new message is added to this queue because a message could later arrive that contains a time stamp as small as the message it just removed and processed from that queue. This will never break causality constraint but may lead to deadlock as some process may keep on waiting for each other forever to send a message but neither will send any message.

We can either avoid the deadlock or may have a separate routine which detects the deadlock and breaks it. A deadlock avoidance method will require additional messages called null messages. It is required that an LP indicates to other LPs a lower bound on the time stamp of messages it will send that LP in the future. Null messages can be used for this purpose. Null messages are used only for synchronization, and do not correspond to any activity in the physical system.

Algorithm 1 Conservative Null Message Deadlock Avoidance Pseudo Code

```
1: procedure SimulateSystem(d1, . . . , dN)
2: while simulation is not over
3:     wait until each FIFO contains atleast one message
4:     remove smallest time stamped message M from its FIFO Queue
5:     clock ← time stamp of M
6:     process M
7:     send NULL message to all LPs with time stamp equal to lower bound on
   time stamp of future messages (current time stamp + look-ahead)
```

An alternative approach to sending a null message after processing each event is a demand-driven approach. Whenever a process is about to become blocked because the incoming link with the smallest link clock value has no messages waiting to be processed, it requests the next message (null or otherwise) from the process on the sending side of the link. The process resumes execution when the

response to this request is received. (**We have implemented this method in our framework for simulating a network as described in later sections**, where it is described in more detail)

4 Optimistic Synchronization Algorithms

Conservative synchronization algorithms avoid violating the local causality constraint, whereas optimistic algorithms allow violations to occur but provide a mechanism to recover. The term optimistic execution refers to the fact that logical processes process events, "optimistically" assuming there are no causality errors.

These algorithms also have the same priliminaries

5 Dataset

- A small synthetic dataset
- A random graph of 100 nodes
- Co-authorship network with 1000 nodes
- Co-authorship network with 100000 nodes

6 Approach

6.1 Distributing the graph

6.2 Establishing communication between the nodes

6.3 Developing the message handling and processing framework

6.4 Designing system assessment methods

7 Interface Description

8 Results Obtained

References

[1] Parallel and Distributed Discrete Event Simulation: Algorithms And Applications by Richard M. Fujimoto, Proceedings of the 1993 Winter Simulation Conference
http://dl.acm.org/citation.cfm?id=256596

[2] Distributed Discrete-Event Simulation by Jayadev Misra
http://dl.acm.org/citation.cfm?id=6485

[3] Integrated Fluid and Packet Network Simulations by George F. Riley, Talal M. Jaafar and Richard M. Fujimoto
http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1167114

[4] Parallel Simulation of Telecommunication Networks http://titania.ctie.monash.edu.au/pnetsim.html

[5] Introduction to Discrete-Event Simulation and the SimPy Language by Norm Matloff
http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf

[6] The OMNET++ Discrete Event Simulation System by Andrs Varga
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.331.1728&rep=rep1&type=pdf

[7] EpiSimdemics: an Efficient Algorithm for Simulating the Spread of Infectious Disease over Large Realistic Social Networks by Christopher L. Barrett, Keith R. Bisset, Stephen G. Eubank, Xizhou Feng, Madhav V. Marathe, Network Dynamics and Simulation Science Laboratory, Virginia Tech, Blacksburg
http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5214892

9 Feedback