8. Anon. (1994). Virtual reality and training, *Government Executive*, June.
9. Kleijnen, J. P. C. (1993). Verification and validation of simulation models, *Technical Report 320.93.186*, Tilburg University, Tilburg, The Netherlands.
10. Law, A. M., and W. D. Kelton (1991). *Simulation Modeling and Simulation*, 2nd ed., McGraw-Hill, New York.
11. Mowbray, D. W., J. W. Wallace, A. L. Herrman, and E. S. Hirschorn (1995). An architecture for advanced distributed simulation, *Phalanx*, Vol. 28, No. 2, pp. 12–15.
12. National Simulation Center (1995). *Training with Simulations: A Handbook for Commanders and Trainers*, Combined Armed Center, Ft. Leavenworth, Kansas.
13. Pate, M. P., and G. G. Roussos (1996). JLINK: a distributed interactive Janus, *Phalanx*, Vol. 29, No. 1, pp. 12–15.
14. Piplani, L. K., J. G. Mercer, and R. O. Roop (1994). *Systems Acquisition Manager's Guide for the Use of Modeling and Simulation*, Defense Systems Management College, Ft. Belvoir, Va.
15. Sargent, R. G. (1994). Verification and validation of simulation models, in *Proceedings of the 1994 Winter Simulation Conference*, J. D. Tew, S. Manivannan, D. A. Sadowski, and A. F. Seila, eds., IEEE, Piscataway, N.J., pp. 77–87.
16. Shiflett, J. E., W. H. Lunceford, and R. P. Willis (1995). Application of distributed interactive simulation technology within the Department of Defense, *Proceedings of the IEEE*, Vol. 83, No. 8, pp. 1168–1178.
17. Sikora, J., and P. Coose (1995). What in the world is ADS? *Phalanx*, Vol. 28, No. 2, pp. 1–8.

CHAPTER 20

Discrete-Event Simulation of Computer and Communication Systems

ALFRED HARTMANN AND HERB SCHWETMAN
Mesquite Software, Inc.

20.1 INTRODUCTION

In this chapter we discuss techniques useful for constructing discrete-event simulations of computer and communications systems. While analytical methods such as queueing theory are sometimes employed (Lazowska et al., 1984) and can offer valuable insights, their scope is limited to problems for which analytical solutions methods are possible. Computer-based simulation is very broadly applicable, and with the availability of many kinds of simulation software packages and the rapid proliferation of low-cost computing power to every desktop, simulation-based analysis has moved to the fore. Today, most computer and communications system designers are familiar with the C/C++ programming language and with the notion of computing processes. This makes process-based simulation in C/C++ a natural and convenient choice for many modeling projects.

20.2 FUNDAMENTAL CONCEPTS

A computer system or communication network consists of a collection of resources. Entities (jobs, programs, tasks, transactions, messages, etc.) compete for use of these resources. Models of these systems possess, at least at some level, constructs that represent both resources and entities. Almost any simulation model of almost any kind of system will have similar constructs. In this chapter we explore the issues that make modeling computer systems and communication networks different from simulation models of other kinds of systems.

Handbook of Simulation, Edited by Jerry Banks.
ISBN 0-471-13403-1

20.2.1 Goals

The major goal of most system modeling projects is to provide estimates of system performance. With computer systems, the most important performance measure is task response time. With communication networks, the most important measure is message latency time. Thus the goal of many projects is to provide accurate estimates of these measures. Other goals include providing insight into the operation of the system and guidance for reducing the impact of performance bottlenecks.

20.2.2 Resources and Entities

As mentioned above, simulation models typically have entities (processes, customers, messages, transactions, etc.) and resources (processors, memory, buses, channels, communications links, etc.). In these models there can be several different types of entities and there can be multiple instances of each type of entity, all active at the same and at different points in simulated time. In many situations, the entities are competing for use of some of the resources of the system model. For example, in a model of a computer system, several simulated jobs can all be competing for use of the CPU resource.

Similarly, there are usually several types of resources in a system model. These can be classified into two major categories: (1) active resources, and (2) passive resources. The distinction depends on what entities do after obtaining access to an element of a resource. An element of an *active* resource is typically used or occupied for a specific interval of time and then released. An element of a *passive* resource is obtained by an entity; at this point the entity proceeds to perform other actions, including gaining access to other resources.

Examples of active resources in a computer system include CPUs, disk drives, and controllers, and examples of passive resources include main memory and buses. In a communications network, message processors and internode links might be active resources, while buffers are examples of passive resources.

The entities of these models depend on the level of the model. In a high-level model of a computer system, the entities might be programs or transactions. In a high-level model of a communications network, the entities might be messages. In a low-level model of a computer system, the entities might be individual computer instructions, I/O requests, and data transfers to and from main memory. In a low-level model of a network, the entities might be data packets or cells.

20.2.3 Workload

Every system model includes a representation of the workload for the system. Here *workload* refers to the sequence of demands by entities for the resources of the system. Obtaining an accurate characterization of the system workload is one of the most important steps toward building an accurate and useful model. With computer systems this task is, in some cases, made easier by the presence of system tools which can be used to automatically collect some of the necessary data. For example, if the entities of the model are on-line transactions, the transaction monitor may keep transaction logs that can be used to build a characterization of the input transaction stream. Similarly, in a computer network, there may be message logs at some of the nodes which can be useful in constructing profiles of the message traffic in the network.

As with most models, it is necessary to have an accurate characterization of the workload in terms of the demands for system resources. With computer systems, this may present some difficulties. For example, a critical input for a model may be the service-time distribution at a disk drive, but a system accounting package may report response times at this drive (and the mean response time at this drive would be an output of the model). In a similar vein, the accounting package might report characters per transfer to a disk drive, but it may be difficult to relate the number of characters transferred to the service time.

Typically, what is required for a model of a computer system is a profile of the kinds of jobs or tasks being modeled. This profile would consist of a stratification of the tasks into different classes, and then for each class, a summary of the demands for service for the different resources of the system. As an example, a workload for an on-line transaction processing system might be classified according to different types of transactions being submitted, and then for each type of transaction, collecting the following information:

- Number of transactions of each type
- Amount of CPU time per transaction
- Number of disk accesses per transaction
- Amount of data sent back to the requester per transaction

For a communications network, the workload characterization might consist of the following kinds of information:

- Number of messages generated at each node
- Message length distribution
- Information about the destination of each message (e.g., the distance to the destination)
- The percentage of messages requiring an acknowledgment

The end result of this workload characterization is a set of parameter values that are required to model accurately the workload in the model.

20.2.4 Output: Measures of Performance

As stated earlier, models of systems are often used to obtain estimates of the average system response time for elements of the workload. Similarly, in models of networks, estimates of the average message latencies and/or delivered bandwidth are of interest. Here *response time* means the time from when a request for service is made (starts or arrives) until the time that the request is completed (departs). *Latency* is the term used for response time in an input–output subsystem, a memory subsystem, or in a communications network. Response times (latencies) are expressed in terms of time units per request. The reciprocal of response time is throughput rate (e.g., transactions per second), while the reciprocal of latency is bandwidth (e.g., megabytes per second). All of these items are usually outputs of simulation models.

If a model is being used to gain insight into performance problems (where performance problems are often indicated by unacceptably long response times or latencies), more output data are required, to pinpoint causes of problems. These additional data

usually consist of statistical summaries of times spent at individual resources of the system model or times spent in the individual activities that make up a complete workload element. In some cases it may be necessary to insert special instrumentation to collect the data necessary to pinpoint causes of performance problems.

20.2.5 Duality of Models

Process-oriented models of computer and communications systems tend to differ in their assignment of processes to simulation entities. In computer system simulations, processes tend to be assigned to the *workload* (e.g., to represent user processes or applications), and these processes acquire and release system *resources*, for which they typically compete. In communication system simulations, processes tend to be assigned to *resources* (e.g., switches, or channels), and the processes handle *workload* messages, for which they typically do not compete. These process assignments tend to keep the number of processes in the simulation to a reasonable number, since the simulation system incurs context switch overhead in dealing with multiple processes. It would normally be prohibitive to model each message as a process that competes for communication system resources, due to their vast number.

These different ways of conceptualizing models leads to a type of model duality between models of computer systems and models communication systems:

Simulation Entity	Computer System	Communication System
Workload item	Process	Token (structure)
Resource item	Structure	Process

This duality can result in a difference in perspectives between computer system simulation and communication system simulation, and perhaps a difference in choice of simulation tools.

Queue service disciplines appropriate to competitive resource acquisition in a computer system simulation may or may not satisfy the needs of message workload processing in a communication system simulation. First come, first served (FCFS) is a simple service discipline common to both computing and communications, while the common round-robin processor scheduling discipline would probably not be applicable to communications simulations. A communication system simulation may need to employ a wide variety of buffer management strategies that may not be reflected in the selection of resource allocation strategies available in a computer system simulation facility. As examples, the "leaky bucket" (Bae and Suda, 1991) buffer management strategy used for ATM network source control is unique to communications, as are combination timeout and priority-based cell disposal methods for congestion reduction in ATM networks.

20.2.6 Object-Oriented Models of Systems

Object-oriented software design methods have much to offer the simulation developer, as discussed in Chapter 11. We may consider object-oriented software design to be based on the principles of:

- *Encapsulation:* the ability to combine data and the functions that manipulate the data into abstract data types
- *Inheritance:* the ability to derive new subtypes from existing abstract types
- *Polymorphism:* the ability for functions to manipulate objects of different types that share certain type relations

The advantages that object-oriented design offers to all types of software development apply also to simulation development. Simulation entities can be encapsulated into objects, classes of simulation entities can be grouped into base object classes and differentiated using inheritance into arbitrary subclasses, and polymorphic functions can be defined appropriately as methods of the object classes. All these object-oriented techniques can serve to organize and simplify simulation model development.

Simulation development systems and programming languages provide predefined base classes for common types of simulation processes and resources. These can be inherited by newly defined subclasses in a particular simulation effort. Polymorphic functions can be predefined that perform standard manipulations on user objects that are instances of subclasses derived from the base class repertoire.

20.2.7 Quasi-parallelism in System Models

Process-oriented simulation uses multiple processes to simulate the parallel activities in the system being simulated. However, it is typical for the simulated execution of these multiple processes to be interleaved sequentially on a single processor. This interleaved execution is generally done deterministically so that results are repeatable from simulation run to simulation run. Such forms of parallelism are referred to as *quasi-parallelism*, to distinguish them from *true parallelism*, which is nonsequential and often nondeterministic. True parallelism may be exploited *underneath* the quasi-parallel simulation environment, to speed simulation, but this is transparent to the simulation design. Quasi-parallelism is not transparent to the simulation design but is an integral part of the simulation design. Building simulation models that exploit true parallelism is discussed in Chapter 12. It can be noted that some of the most successful applications of parallel simulation techniques have involved models of large communications systems.

Quasi-parallelism must provide facilities for simulated processes to *block* (i.e., waiting for simulated time to pass, for resources to become available, or for a simulated event to occur). At the appropriate point in discrete simulated time, processes must *unblock* and resume simulated execution. Multiple processes need some form of synchronized interprocess communication, which can be provided by predefined synchronization classes implementing events, signals, mailboxes, semaphores, and so on.

20.3 COMPUTER SYSTEMS

A simulation model of a computer system (MacDougall, 1987) must capture the essential parts of the real system. These essential parts consist, in broad terms, of (1) workload elements, (2) system resources, and (3) system policies (which govern allocation of resources to elements of the workload). In this section we discuss some of the issues involved in modeling these parts (Jain, 1991).

20.3.1 What Is the Workload?

Computer systems exist to process elements of the workload, where, depending on the system being modeled, a workload element could be one of the following items: (1) job, program, or task; (2) transaction or query; or (3) I/O request or request for an item in main memory. Each of these kinds of workload elements is a sequence of demands for service at the resources of the system. As an example, consider a job or task as being representative of a workload element. A job (a program) is a sequence of alternating requests for time on a CPU (CPU bursts) and input/output service (transfers of blocks of data to and from I/O devices such as disk drives). In addition, the job will request use of blocks of main memory. In most modern systems, multiple jobs are executing simultaneously, so there is competition for all of these resources. System performance (job response times) reflects the ability of the system to satisfy these conflicting requests for use of the system resources.

20.3.2 Modeling System Components

Computer systems consist of both hardware and software components. In addition, the operating system manages access to these components; it embodies the control mechanisms which are in place to guarantee correct and efficient operation of the system. In a simulation model of such a system, all of the important components, as well as the policies that control access to these components, should be represented if accurate estimates of system performance are to be achieved.

Hardware Components. The most important hardware components of a computer system are the CPU and the I/O devices. Main memory is critical to the operation of the system, but in most modern systems, main memory is quite large and is usually not a major factor in determining system performance. However, incorporating main memory into a model is usually not a difficult task. The other important and often neglected kind of component is the interconnection hardware (called a *bus*), used to tie the CPU and the I/O devices to the main memory. Modeling these hardware components is usually fairly straightforward because they can be represented by static resources. The key is to accurately model the kinds of serial or parallel accesses that are possible in each of these components.

Software Components. The software components can be more difficult to represent in a system model. Typically, a software component can embody lengthy sequences of actions and implement complex operating policies. Furthermore, some of the components are provided by third-party vendors, and their inner workings will not be disclosed to a system modeler.

One class of software component that is found in systems consists of those that offer service to user programs. An example is a database management system (DBMS). Many applications in a system could require service from the DBMS. However, there are limits to the number of requests for service that the DBMS can handle simultaneously. Thus such a server can be a limitation to the performance (job response time) for jobs that must access critical data via the DBMS. Other software components that can affect the performance of jobs in a computer system include transaction processing monitors, network access systems, and remote file servers.

If a software component is determined to be a potential performance bottleneck in a system being modeled, a special effort may be required to ensure that an accurate representation of the component is available. For example, if a commercial DBMS is an important component in a system, the system model may require a simulation model of the DBMS. To incorporate this component into the model, the modeler may have to take one of the following steps:

- Obtain additional information from the vendor or other sources.
- Perform some black-box measurement studies, so as to infer the behavioral characteristics of the DBMS.
- Obtain a prebuilt simulation model of the DBMS.

The other aspect of software components that is of importance in a model is the set of control and management policies provided by these components. These policies and rules appear in the operating system (which controls the operation of the entire system) as well as in the components mentioned above. These policies specify priorities, scheduling rules, resource constraints, and rules for dealing with congestion and overflow (e.g., maximum queue lengths at an I/O device). Depending on the level of detail in the model, these policies may be an important factor in the development of an accurate system model.

20.3.3 Asynchronous Operation

In an attempt to improve the utilization of system resources, modern systems encourage an asynchronous style of behavior. The basic principal is that a program is able to initiate I/O actions that can take place in parallel with continued use of the CPU. A model of a system must capture this parallel, asynchronous style of operation. A process-oriented simulator provides a base for implementing models in which the major components can operate asynchronously.

20.3.4 Trade-offs: Detail Versus Cost

The purpose of a system model, in the simplest terms, is to produce response intervals for workload elements. These intervals can then be used to provide the estimates that are needed to make judgments about system performance.

A response interval typically consists of one or more subintervals, in which each subinterval has a delay interval and a service interval at some resource. A very high level system model has only a few resources, and as a result, the response interval has only a few subintervals. Such a model could execute very quickly, but the resulting estimates of response times might be inaccurate.

To improve accuracy, the high-level resources can be redefined as collections of lower-level resources. When this is done, what was formerly a single service interval at a resource is now a sequence of delays and service intervals for these lower-level resources. The gain is a more accurate representation of the resource; the cost is an increased number of simulated events, which translates to a longer-running model.

As an example, in a high-level model, an input–output operation could be modeled as a single service interval. In the model there could be a single input–output resource, and programs get, in a sequential manner, service at the resource. The real input/output device and associated connection components is more complicated; capturing more of

this complexity in a model might result in better estimates of the time required by a program to complete its input/output requirements. A more complex model could have a bus, a controller, and several disk drives. This more complex model could be processing multiple requests in parallel and could provide more accurate estimates of response times at the input/output resource.

All system models represent trade-offs between increased levels of detail and increased execution. Good models strike a balance between these conflicting goals.

20.3.5 Model Results

A model of a system can be used in two distinct ways [LaKe91]: (1) to provide estimates of the output values for the system operating in "steady state," and (2) to provide estimates of the output values for the system operating over a specified interval of time (e.g., for an 8-hour shift). The problems of providing accurate estimates in both of these situations are well known, as are techniques for dealing with these problems (Law and Kelton, 1991).

Models of computer systems can present some additional problems to the task of providing reliable estimates of the output variables. These issues stem from the fact that real computer systems can be large and complex. Furthermore, some of the important components are not readily visible, and producing accurate representations of their behavior may be difficult. Thus validation of the structure and operation represented in a model may be a significant issue as it is developed.

In some cases it is possible to obtain output results for the test workloads used to parameterize the model of the workload. In these cases, the results from the model can be thoroughly validated. In other cases, the model developers may be forced to review the design and implementation of the model itself, to verify the validity of the results.

20.4 COMMUNICATIONS SYSTEMS

Any project to develop a simulation model of a communications system can be divided into two main branches of work: (1) construction of the workload model, and (2) construction of the network model.

20.4.1 What Is the Workload?

A communications system workload model is, primarily, a quantitative description of the messaging traffic within a communications network. The description may be either explicit, as obtained from instrumented trace studies of a real system, or statistical, making random draws from a set of probability distributions. Regardless of the form, the workload description should accurately represent the messaging traffic for the intended network.

Workload validation is the process of ensuring appropriate accuracy in the formulation of the workload description. Even the most faithful network model will be useless if it is exercised with an invalid workload: "garbage in, garbage out." Workload construction and validation is a major portion of the effort in building any communications simulation model.

A workload driver (model) presents messages to the communications system for delivery through the network. We are concerned with the course of the message traffic through network space and time. A message may be part of a larger message stream

with a given point of origination and one or more points of destination. Messages to be sent to only a single point of destination are referred to as *unicast traffic*, while messages sent to multiple destinations are *multicast traffic*, with *broadcast traffic* being intended for all destinations. A message can be sent in one piece to its destination(s), or be decomposed into smaller units, or be composed with other messages into larger units, or the pieces of multiple messages can be variously combined and separated in their course through the network.

What happens to a message *after* its entry to the network is the province of the network model. The workload model describes the kinds of messages that *enter* the network and the arrival times for these messages. In this sense the workload serves to provide the *stimulus*, while the network serves to provide the *response*. Part of the overall design of the simulation is determining the most appropriate type of stimulus that will generate the responses of interest.

Since real networks tend to be constructed in layers, the character of the workload will be greatly affected by the layer chosen for direct stimulation. For example, if only the physical links are to be studied, the stimulus can be presented at a very low level (e.g., physical data unit) in the layering, while if the performance of a distributed database operating over the network is to be studied, the stimulus is presented at a very high level (e.g., database query). The higher the layer chosen for stimulation, the more layers must be present in the network model, so the choice of the layer greatly affects the total implementation effort. If lower layers are to be treated only in the abstract, experience is necessary to judge which details to ignore and which to model.

The logical data unit may be much larger than the physical data unit transferred over the link, or the reverse may be true. For example, in Ethernet over ATM, the Ethernet packets may be kilobytes in size while the ATM cells are only a few tens of bytes. On the other hand, in ATM over SONET, the SONET frames hold a large number of ATM cells. If the number of messages can be reduced by two orders of magnitude, that is desirable if this reduction can be accomplished without loss of accuracy in the simulation. The evaluation of this trade-off may depend on which network performance attributes are important to measure; thus specific rules are difficult to give.

The workload model injects messages into the network, and each message has the following principal attributes:

- Time of origin
- Point of origin
- Point(s) of destination
- Message characteristics (size, priority, etc.)

Usually, the origin times (and possibly the destination addresses) of the injected messages are drawn from probability distributions. Or rather, the time interval between message originations at a given point of injection is drawn from a probability distribution. *Proper selection of the distribution of interorigination time intervals is a major aspect of workload validation.*

Since communications networks are frequently analyzed as queueing networks, which are most mathematically tractable with exponentially distributed time intervals, the exponential distribution is a frequent, though not necessarily valid choice. It is applicable where message origination is a Poisson (or memoryless) process, but much multimedia traffic, such as compressed voice and video streams, tends to show Markov

process characteristics (or bursty behavior) (Bae and Suda, 1991). A Poisson process and a non-Poisson Markov process with the same average intermessage time interval may produce dramatically different network performance results due to traffic congestion at intermediate network nodes. Here a non-Poisson Markov process refers to a process in which some time intervals (e.g., intermessage intervals) are characterized as an exponential distribution with one of two means; the choice between these two means is determined by Markov state transitions with specified probabilities. The goal is to model a process accurately with two distinct modes of behavior.

Bae and Suda (1991) provide several examples of Markov process models for ATM network traffic of voice, video, and data content. The Markov transition probabilities typically cause state transitions that alter the message interarrival time significantly so as to produce the necessary burstiness found in the actual source. Each of the states may correspond to some simple probability distribution (such as exponential) but with different distribution parameters for each state.

All of these workload models are dependent on the types of data coding used, particularly if compression is employed, since this may alter the traffic characteristics considerably. Forward error checking (FEC) using error-correcting codes (ECC) can also alter traffic patterns by changing the incidence of re-sends of messages. Regardless of the distribution chosen, the random, uncorrelated time intervals between message originations argue for an asynchronous relation between the workload model and the network model.

To illustrate some of these issues, assume that we are developing an object-oriented simulation model in C++. Furthermore, let us propose a class named `Node` and a class named `Workload`:

```
class Node_c { . . . . };
class Workload_c { . . . . }
```

Each instance of Node could be given an instance of Workload, to supply it with new messages to send:

```
Workload_c work (. . .);
Node_c node (work, . . .);
```

Or it might be done the other way around:

```
Node_c node (. . .);
Workload_c work(node, . . .);
```

Either the Node_c instance, ?node, has to block until the next message origination time, or the Workload_c instance, ?work, has to block. This sort of quasi-parallel asynchrony in simulated time is not a feature of the C/C++ programming languages or of operating systems. Essentially the node and its corresponding work need to function as asynchronous coroutines in discrete simulated time, based on a simulation event queue, as discussed in the earlier section on quasi-parallelism.

20.4.2 Networks and Network Protocols

The network model is the other half of the effort, and together with the workload model constitutes a full communications system model. Networks are often represented as

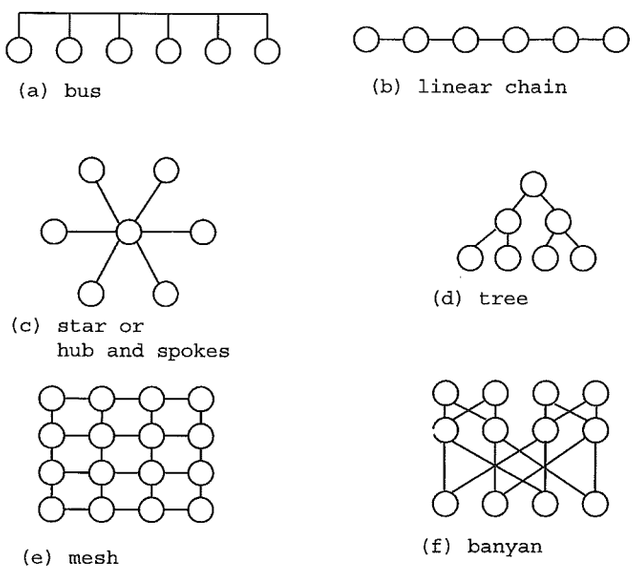

Figure 20.1 Simple and regular network topologies.

graphs, with nodes and edges (links). Multidrop links or buses can be viewed as edges in a hypergraph network representation. Usually, the terminal nodes serve as source and destination nodes, while the interior nodes serve as switch or router nodes. This graph structure becomes more elaborate when network protocols are included, especially those with stratified software layering, as in network architectures today.

The interconnection pattern of network nodes and links constitutes the *network topology*. Complex and irregular network topologies can be expensive to model, both in terms of design time and simulation run time, but cannot always be avoided. Well-known simple and regular network topologies are shown in Figure 20.1.

There need be no identity between the *logical* network topology and the *physical* network topology and the *physical* network topology. As an example we show in Figure 20.2 a logical ring network wired as either a physical ring, a linear chain (interleaved or noninterleaved), or as a star. The well-known token ring network in its usual realization is an example of the latter, since physically wiring it as a ring (Figure 20.2*c*) is often cumbersome and failure prone.

There may or may not be significance to the possible difference between logical and physical network topologies. If signal propagation delays are critical, the physical topology could be of concern in the network model. For example, in bused versions of Ethernet, the incidence of packet collisions is affected by the relative positions and separations of nodes on the bus and by the total end-to-end propagation delay.

20.4.3 Service Disciplines for Buffers, Channels, and Switches

Communications system performance can be greatly affected by choice of service discipline. Also, various service disciplines can vary in ease of implementation (depending

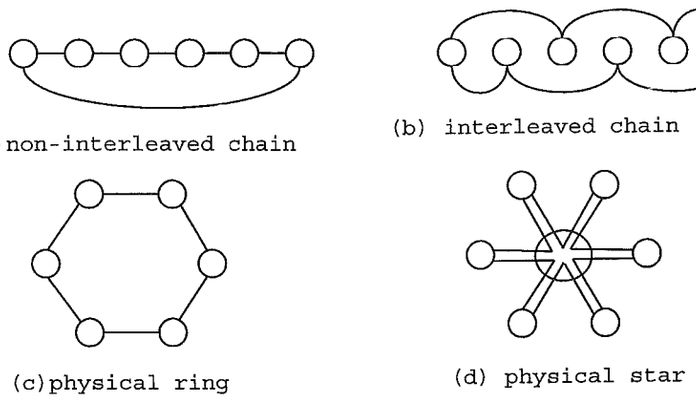

Figure 20.2 Logical versus physical network topologies.

on choice of tools and libraries) and in computational effort at simulation run time. Engineering trade-offs may argue against attempting theoretically optimal service disciplines, either in simulation or in actual network design.

Some of the more common queue service disciplines applicable to communications buffers are:

- First come, first served (FCFS), also called first in, first out (FIFO)
- Priority order, based on simple message priority or on some message attribute [e.g., time of origination, or some quality of service (QoS) parameter associated with the message or the channel or the destination]
- Random order, either purely random (uniform distribution) or skewed by some attributes of the message or time spent in the queue)

Essentially, FCFS is a special case of priority order that takes time of queue entry as the priority. Simularly, priority order can be viewed as a special case of deterministic ordering, as opposed to random ordering. The communications system architect is concerned with choosing service orders that maximize a particular figure of merit for the network. Since both network performance and network cost are multidimensional quantities, specification of an appropriate figure of merit is not simple.

In some cases there may be provably optimal service orders for a particular design goal, and frequently one desires to simulate the performance difference between optimal and nonoptimal orderings. This is especially true if the cost or complexity of implementing optimal ordering is high.

In the case of buffer management, one needs to consider (1) message entry to the buffer, (2) message retention in the buffer, and (3) message exit from the buffer (either normal or abnormal).

Message entry to the buffer may depend on the current state of the buffer (e.g., full or nonfull, priority of presently buffered messages versus incoming message priority, etc.) and the incoming message attributes. Message retention in the buffer might involve ongoing buffer content reorganization based on dynamic parameters, including the time-in-queue of the presently buffered messages. Message exit from the buffer may occur

either normally as the message is serviced, or abnormally as the message is dropped for service deadline expiration, preemption by a higher-priority message, or whatever.

Frequently, buffers are used in association with switches, to store messages in the process of passing through a switch from an inbound channel to one or more outbound channels. The delays and complexities associated with message switching can have a great impact on the switch buffer management strategy. Buffers may be employed at the inputs to the switch (input-buffered switch), at the output ports from the switch (output-buffered switch), at both inputs and outputs, or at any intermediate stage within a multistage switch.

20.4.4 Accounting for Transmission Times and Propagation Delays

Depending on the scale of the simulation model in space and time, transmission times and propagation delays may or may not be significant in a given situation. A simple back-of-the-envelope analysis is usually sufficient to determine if these times need to be accounted for in the model or if they may safely be treated as zero (i.e., instantaneous transmission and/or propagation).

Because communications channel capacity and node separation distances are often fundamental in communications system simulation, it is rare that both these model parameters are ignored. Transmission speed and separation distance produce significant delays in communication that usually must be accounted for in the model.

20.4.5 Size Issues

The simplest way to manage size issues is to parameterize the model with size factors and to raise these factors judiciously as experience is gained with smaller simulations. How the computational load of simulation varies with network size or workload quantity may vary, but often may be faster than linear growth. A binary progression of 1X, 2X, 4X, 8X, and so on, can be tested and plotted against execution time to estimate practical simulation limits. Since discrete-event simulation can employ arbitrarily large amounts of computational time and expense—and still never produce perfect accuracy—seasoned judgment must prevail.

If scaling factors can be measured so that simulation results from small simulations can be extrapolated to results for large systems, significant savings can be realized. Even if such extrapolations are not trusted, the vast multidimensional design space can be explored more quickly with small-scale simulations, and the large-scale simulations be reserved for choices proven on a smaller scale.

20.4.6 Tradeoffs: Detail Versus Cost

Pragmatic engineering practice usually argues in favor of simple approaches initially, both in design of actual communications systems and in construction of their corresponding simulation models. It is often simpler and safer to converge on a final design by iterating over a sequence of successively less abstract designs. This technique of successive refinement in design and modeling can reduce total engineering effort and time to completion significantly by proceeding through a sequence of stable intermediate points.

Another valuable method is to simulate at a simple abstract level whenever possible, even if it is thought to be unrealistic, and measure the deviation from a more complex

model. Only if the deviation is deemed significant should greater design or simulation effort be expended to use a more concrete model. A crude estimate in time is often far more valuable than an accurate estimate that arrives after the battle is lost. Rapid progress in communications system architecture and technology, and the swift obsolescence of existing communications systems make timeliness more of a virtue than n-digit accuracy.

20.5 EXAMPLES

20.5.1 Client–Server System

A client–server system typically is a collection of computer systems, with one system hosting a server application and the other systems hosting client processes. The systems are connected via a network (either a local area network or a wide area network). An example of this kind of system is a bank, with the server being the main accounting system and the clients being teller terminals located in the branch offices of the bank.

The clients all send requests for service to the server; the server processes each of the incoming requests and returns a reply to each requester. A major design issue for such a system is the size and processing capacity of the server, so that all of the requests for service can be handled promptly.

A process-oriented simulation model of this kind of system could be built from node objects, where each node has one or more CPUs, a collection of disk drives, main memory, and an input "mailbox" (a mailbox is a mechanism for receiving messages). One node is the server node, and the rest of the nodes are client nodes. A server process is assigned to the server node, and client processes are assigned to the client nodes.

The server process models the server application. Briefly, the server application (process) receives requests for service in its input mailbox. It decodes each request message and then assigns the request to a subprocess. The subprocess proceeds, in a parallel, asynchronous manner, to process the request and then formulate a response that is sent back to the client who originated the request. Processing this request by the server subprocess will require using a CPU for some amount of simulated time and possibly making several accesses to some of the disk drives, depending on the type of request. It is the competition for server node resources by these subprocesses that determines the time required to service requests.

Each client process is associated with a human operator (teller). This process formulates a request for service, sends it to the server's input mailbox, waits for a reply, evaluates the reply (representing a delay), formulates the next request, and so on. All of the clients operate in the same manner.

The key measure of performance is the average response time for service requests. The model can be used to configure the server node so that the average response time (or other measures of performance) meet the criteria for the system. The model can also be used to project system performance as the number of clients increases.

20.5.2 Modeling a Shared Memory Multiprocessor Computer System

A shared memory multiprocessor computer system has several CPUs, each with a private cache; each CPU (really each cache) is connected to a single bus, which in turn is connected to the main memory. This arrangement is attractive because processes execut-

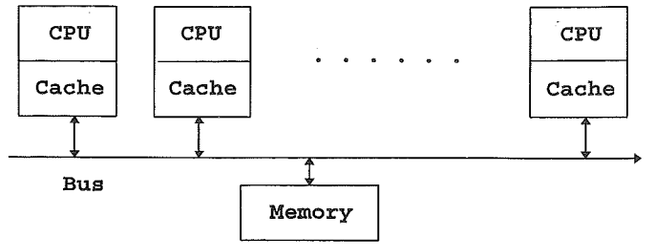

Figure 20.3 Shared memory, multiprocessor system.

ing on different CPUs (in parallel) can easily communicate with each other via common areas of the main memory (shared memory). Such a system is shown in Figure 20.3.

As could be expected, the bus between the processors and the main memory is a potential performance bottleneck. The model can be used to predict the number of processor nodes that can be accommodated on this bus. The number of processors, the instruction processing rate, the cache miss rate, and the speed of main memory all affect the performance of this system.

Another issue with such a system is the cache coherency protocol, used to make certain that all accesses to data (including modified shared data) is to the correct (latest) data values. The choice and efficient implementation of this protocol can have an impact on performance.

A simulation model was implemented using a process for each cache. The bus and the main memory are modeled as simulated resources. The cache processes decided when a cache miss occurs using a probability distribution based on the cache miss rate (an input parameter to the model). Each miss causes an access to the bus and to main memory. The competition for bus and memory accesses is modeled as competition for the simulated resources. The main performance measures are memory request latencies and bus utilization. This type of system can accommodate only a limited number of CPUs; using this kind of model can help determine this limit.

20.5.3 Modeling Ethernets and Token Rings

The two major varieties of local area networks are Ethernet (ANSI/IEEE, 1996) and token ring (ANSI/IEEE, 1995). Transmission rates are nominally 10 Mbit/s for Ethernet and 16 Mbit/s for token ring, although effective channel utilization tends to be rather low for such protocols. From a modeling standpoint the principal differences are transmission rate, network topology, and media access control (how permission to transmit is obtained).

Both the original Ethernet and token ring standards are becoming dated, and new local area network standards are being proposed for higher-bandwidth networks. The new standards are needed principally to accomplish two goals: higher transmission rates and support for multimedia traffic (mainly voice and video).

This *digital convergence* of traditional data traffic with historically analog voice and video traffic is the main impetus behind the need for higher-bandwidth and more inclusive protocols. In addition to local area networks, the telecommunications industry is supporting the Asynchronous Transfer Mode (ATM) (ATM Forum, 1995) standard as the wide area network solution for digital convergence. There is also a standards activ-

ity to bring ATM into the local area network environment as the logical layer between higher-level local area network protocols and the lower-level physical channel.

Modeling an Ethernet. While there are a number of different varieties of Ethernet, the typical transmission rate is 10 Mbit/s, or one bit every 100 ns. Access to the channel is gained by a probabilistic method of first waiting for the channel to become idle and then transmitting the message. The receiver listens during transmission, and if it should detect a collision with another sending party, a special collision signal is broadcast and then the message transmission is aborted. After a random-duration *backoff period*, the transmission is reattempted at the first idle interval following the backoff period. The random distribution from which the backoff period is drawn is adjusted according to the frequency of collisions experienced.

Collisions result in aborted message traffic and waste of channel utilization. Collisions result when two or more parties desire to send, observe the channel idle, and begin transmission within a critical interval determined by the propagation delay between senders. The greater the distance between senders, the longer the critical interval during which collision is possible. This interval is independent of transmission rate, so the cost of collisions in lost traffic only increases with increasing channel capacity.

Modeling a Token Ring. Nominal token ring transmission rates are 16 Mbit/s, or one bit every 62.5 ns. Media access is gained by acquiring a token that is passed around the logical ring topology of the token ring network. Elaborate protocols decide how the first token is created on network startup and how a new token is forged if the circulating token is lost. A sender is clear to send when the token is acquired, and the sender is guaranteed no collision (unless there is a duplicate token in the ring). It is the sender's responsibility to remove the message from the ring when it has circuited the ring back to the point of origination. MacDougell (1987) discusses simulation models of both token rings and ethernets.

20.5.4 Modeling High-Speed Networks

High-speed networks are generally considered to be networks with channel rates at or above 1 Gbit/s. At these rates, traditional methods of flow control via return messages to "slow down" are cumbersome. For example, at a signal velocity of $\frac{2}{3}c$ (about 2×10^8 m/s), Table 20.1 shows the quantity of "bits in flight" in the channel just as the leading bit hits the receiver. Thus the communications system modeler may encounter alternative flow control methods (e.g., channel capacity reservation, transmission metering, etc.) in dealing with high-performance networks.

Probably a greater concern for the network modeler confronted with simulating high-

TABLE 20.1 Channel Bit Storage

Transmission	Link Length	Bits in Flight
1 Gbit/s	1 km	5 kbit
1 Gbit/s	1000 km	5 Mbit
10 Gbit/s	1 km	50 kbit
10 Gbit/s	1000 km	500 Mbit

performance networks, though, is the tremendous simulation effort to model even a small amount of real time. While the propagation delays are unaffected by the high transmission rates, the volume of traffic in the network is increased dramatically. Other factors being equal, a 1-Gbit/s network would have 100 times the traffic in the network at any given time as a 10-Mbit/s network, assuming the same channel utilizations. If higher channel utilizations are anticipated, to utilize the costly high-performance channels more efficiently, a several-hundredfold increase in traffic could occur.

The modeler undertaking to simulate a high-performance network needs either a great deal of computing power or must be willing to wait a significant amount of time for the results. The alternative is to scale back the size and scope of the model, to compensate for the additional workload. This could reduce the simulation accuracy and workload range under study.

20.6 SUMMARY

In this chapter we have covered some of the basic techniques and issues associated with constructing and using simulation models of computer systems and communications networks. With both kinds of systems, a workable model consists of two equally important submodels: (1) the workload model, and (2) the system model.

The workload model is responsible for generating units of work that resemble the units of work for the real (modeled) system. The system model processes the stream of simulated workload units. In many cases the most important measure of system performance is the response time (latency) for these workload units as they enter and then eventually leave the system. Accurate simulation models of real systems must provide realistic estimates of these response times.

Models of computer systems and communications networks differ from other types of simulation models in some significant ways:

- *Time Units.* Most events in these types of models occur in the range of milliseconds and/or microseconds; this means that if significant periods of real time are to be spanned by the simulation time, thousands or millions of events will be simulated; this, in turn, means that simulation models could require extensive amounts of computer time.
- *Workload Models.* In many cases, existing systems can be used to generate the parameter values required to characterize the workload; also, the nature of these models means that without accurate profiles of the workload, it is difficult, if not impossible, to construct useful models of these systems.
- *System Models.* In many cases the systems being modeled cannot be readily viewed by a modeler; the real system is collection of resources and allocation policies that are controlled by software or implemented in software and are not easy to visualize. Assistance from system experts is often critical to the development of useful simulation models.

The field of simulation modeling of computer systems and communications networks covers a broad range of models, application domains, and uses. There are a number of commercially available packages that are tailored to implementing and using models in some of these domains. A person faced with the challenge of developing a model in this

area should evaluate some of these packages before launching a "build from scratch" implementation project. There are also several technical conferences and workshops that include sessions addressing topics in this area. A list and/or evaluation of some of these packages and conferences is beyond the scope of this chapter.

REFERENCES

ANSI/IEEE (1995). *Standard 802.5 (Token Ring)*, IEEE, Piscataway, N.J.

ANSI/IEEE (1996). *Standard 802.3 (Ethernet)*, IEEE, Piscataway, N.J.

ATM Forum (1995). *ATM User Network Interface (UNI) Specification Version 3.1*, Professional Technical Reference Series (Paper 0-13-393928-X), Prentice Hall, Upper Saddle River, N.J.

Bae, J. J., and T. Suda (1991). Survey of traffic control schemes and protocols in ATM networks, *Proceedings of the IEEE*, February, pp. 170–189.

Jain, R. (1991). *The Art of Computer Systems Performance Analysis*, Wiley, New York.

Law, A., and D. Kelton (1991). *Simulation Modeling and Analysis*, 2nd ed., McGraw-Hill, New York.

Lazowska, E., J. Zoharjan, G. Graham, and K. Sevcik (1984). *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*, Prentice Hall, Upper Saddle River, N.J.

MacDougall, M. H. (1987). *Simulating Computer Systems: Techniques and Tools*, MIT Press, Cambridge, Mass.

CHAPTER 21

Simulation and Scheduling

ALI S. KIRAN

Kiran and Associates

21.1 INTRODUCTION

In a job shop, a set of *orders* often referred to as jobs, parts, products, and so on, may require one or more *operations*, given by a process plan (job routing, etc.). The *process plan* specifies each operation and its requirements (i.e., resources and time required) as well as the *sequence of operations*. In general, there may be alternative *resources* (which can be referred to as machines, processors, workstations, and so on, for each operation. The operations themselves may be substituted by other operations based on availability and or performance considerations. Scheduling in practice usually refers to the determination of a set of orders, which will be processed by the resources during a short-term period (day, week, etc.). The selection of this period, the scheduling horizon, may be part of the decision.

A real-world scheduling problem could be stated simply as "a selection of five orders to run on Monday." In selecting the orders to be completed first, the shop supervisor uses performance measures, although in most of the cases, indirectly. For example, in the hope of reducing the number of late orders, he or she may give *priority* to orders with tighter due dates. Usually, there is more than one objective present in the practice of scheduling. In most cases the scheduling objectives cannot be even stated in terms of a quantifiable *scheduling criterion*. In addition, the concern of the scheduler is to reduce the negative impact of random events such as machine downtimes, absenteeism, scrap, and reworks.

The scheduling problem is the determination of the start and completion time for each operation of each order so that (1) no constraints are violated, and (2) some scalar function of the operation start and completion time is minimized (or maximized). The first constraint leads to the concept of a feasible schedule, whereas the second constraint defines an optimal schedule. These concepts are explored in the next section.

Scheduling problems generally include restrictive assumptions in order to be solved. A representative set is provided here for clarity [1].

Handbook of Simulation, Edited by Jerry Banks.
ISBN 0-471-13403-1