Matthew Jacob
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012.
Email: mjt@csa.iisc.ernet.in

Discrete Event Simulation

Computers can be used to simulate the operation of complex systems and thereby study their performance. This article introduces you to the technique of discrete event simulation through a simple example.

Introduction

Computers are playing an increasingly integral part in our daily lives. Communication, entertainment, finance, education, governance, health care … the list of areas in which our lives are impacted by computers is large and ever-growing. There are people who use computers every moment of their waking lives, others even when they are asleep.

In addition to this, computers play a key role in determining their own destiny. Scientists and engineers who investigate new computing and communication technologies routinely use enormous amounts of computational power in the design and evaluation of possible future products. This sometimes involves studying future systems by simulating their expected behaviour[1]. Discrete event simulation is a widely used technique in this context.

[1] See the two-part article: N K Srinivasan, Computer Based Modelling and Simulation, *Resonance*, Vol.6, Nos.3 and 4, 2001.

In discrete event simulation, a computer program is written to mimic the behaviour of the system under study. The system could be a single CPU or a cluster of them, a network switch or a large communication network, and so on. This program keeps track of the *state* of the system as time progresses. By 'state' we mean the condition or status of the system at a given time. System state will be represented in the program by a collection of state variables.

How is discrete event simulation different from other kinds of simulation? One key distinguishing factor is a restriction placed on how the system state can change. Specifically, in discrete event simulation, *a change in system state takes place instantaneously*, i.e., at an instant in time. The state of the system

Keywords
Simulation, modelling, computer programming.

can change only at such time instants; these are referred to as *events*.

Example: Railway Reservation Counter

Rather than an example from the world of computers or communications, we will choose an example system that is easy to understand. Consider a simple model of a railway reservation counter at a railway station. If you want to buy a ticket, you wait in a queue in front of the ticket seller counter; you leave with your ticket after getting service from the ticket seller. This is, of course, much simpler than the ground reality at the railway station. In our simple model, the railway reservation counter is abstracted down to a queue (of waiting customers) and a server (the ticket seller), as shown in *Figure* 1.

Suppose that we are interested in learning more about the characteristics of both the queue and the server. For example, the average time that customers spent in the system, the percentage of time that the ticket seller was busy, and you can think of other interesting possibilities. For this purpose, we could describe the state of the system at a given instant in time by (i) a queue of waiting customers, and (ii) a flag indicating whether or not the server is busy. We will refer to these two state variables as CustomerQueue and ServerFlag. Associated with each customer in the queue, the program would record the instant in time when that customer arrived as well as how much service time his request will require.

Let us next consider what events can cause the state of the system to change. We must consider both the state of the queue and that

CPU simulators of future processors simulate program execution to estimate execution time, energy consumption and even how hot the CPU will become.

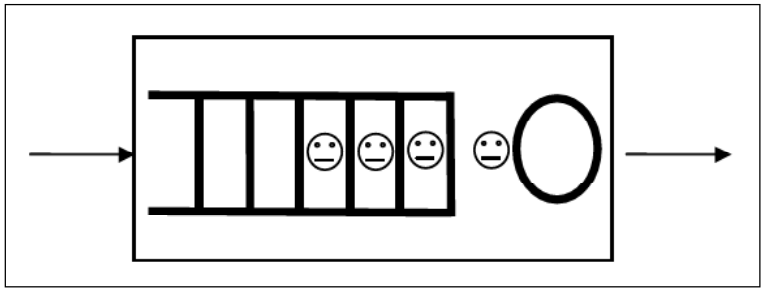

Figure 1. Simple model of Railway Reservation Counter.

Network simulators are useful in the design and configuration of computer communication networks, taking different kinds of network components into account.

of the server. How does the state of the queue change? The state of the queue clearly changes when a new customer arrives at the railway station to purchase a ticket and joins the queue. It also changes when a customer finishes getting service and leaves, making way for the customer at the front to leave the queue and be served. How does the state of the server change? The state of the server changes from idle to busy when a new customer arrives at the railway station and finds the queue empty; this lucky new customer will not have to wait in the queue and will get service immediately. The state of the server will change from busy to idle if the queue is empty when a customer leaves after getting service; that is, there are no waiting customers and the server becomes idle.

This quick analysis reveals that changes to the state of the queue and server happen in response to two types of events: (i) a new customer arrives, and (ii) a customer leaves after getting service. We will refer to these two types of events as CustomerArrival and CustomerDeparture.

Discrete Event Simulation Algorithm

After determining the appropriate state variables and events based on our understanding of the system under study and the objectives of the study, we can apply the discrete event simulation algorithm in our simulation program.

Pseudo-code for this algorithm is shown below.

```
Algorithm  DiscreteEventSimulation

1. Initialize the state variables
2. Initialize the 'collection of pending events'
3. Initialize the simulation clock
4. while (there are pending events to be handled)
       Remove the pending event (E) with the smallest timestamp (t)
       Set simulation clock to that time t
       Execute the event handler for event E
```

As you can see, the pseudo-code is made up of a few initialization statements and a while loop. It uses a simulation clock, a collection of pending events and event handlers for the different types of events used during the simulation.

A discrete event simulation mimics the state of the system as it progresses over time.

Recall that a discrete event simulation mimics the state of the system as it progresses over time. The simulation clock keeps track of the progress in time. It is typically initialized to 0 and increases during the execution of the simulation. During execution of a discrete event simulation, system state can change only at the distinct instants of time associated with events. The algorithm steps from event to event in increasing order of time, making appropriate changes to the state variables. The change in state is implemented through execution of an event handler function associated with the event type. It is the responsibility of the programmer to write an event handler function for each type of event, as well as to declare and initialize the required state variables.

What is the 'collection of pending events'? It is the data structure used by the discrete event simulation algorithm to keep track of the events that are yet to be mimicked as the discrete event simulation progresses over time. It may be possible to initialize this collection with all of the events that occur during the time interval under study. In some cases, this may not be possible; certain events may get added to the collection of pending events while the discrete event simulation program is running, *during the execution of event handlers*. The implementation of the 'collection of pending events' must include a function for this purpose. We will refer to this function as ScheduleEvent (eventType, timestamp), the two parameters specifying the type of the newly created event and its associated time. From the first step in the algorithm's while loop, it is clear that the implementation of the 'collection of pending events' must also provide a function to identify and delete the pending event with the smallest timestamp.

Discrete event simulation is also used to study systems in manufacturing, logistics, finance, etc.

It was mentioned that the simulation clock is typically initialized to 0 and increases during the execution of the simulation. Can it not move forward and backward in time? For example, you could write a simulation program with an event handler which, while processing an event with timestamp 15 generates an event with timestamp 10. When the simulation runs, the simulation clock would step back from 15 to 10 when this new event is handled! The question to ask ourselves here is: Would such an event handler be meaningful in a real world scenario? Is it possible for an incident that is happening right now to change the state of the world as it was yesterday? If not, such an event handler would not be written.

Return to Example: Railway Reservation Counter

Our simple model of the railway reservation counter uses two state variables, CustomerQueue and ServerFlag. We can initialize CustomerQueue to Empty and ServerFlag to Idle, reflecting the situation just prior to the counter opening in the morning. Our simple model involves two types of events, CustomerArrival and CustomerDeparture. We must now implement the event handlers for these event types.

Recall that associated with each arriving customer, we have two pieces of information – the time at which he arrives into the system and the amount of server time that will be required to satisfy his purchase requirements. The event handler for event type CustomerArrival could work as follows: if the server is idle at the time of arrival of the new customer, it can immediately provide service to this lucky customer. This customer starts getting service at time **t**, his arrival time at the counter. His service time requirement is known. So, we can schedule a CustomerDeparture event for his departure from the system as part of this event handler. On the other hand, if the server is busy, an entry for the new customer must be inserted into the CustomerQueue of waiting customers.

```
eventHandler CustomerArrival(time t)

       if (the server is idle)
              Set ServerFlag to Busy, serving the new customer
              Schedule the CustomerDeparture event for this customer
       at time t+service time required
       else
              Insert an entry (arrivaltime=t, serviceTime) for the new
                     customer into the CustomerQueue
```

The event handler for event type CustomerDeparture is equally simple: it checks whether the CustomerQueue is empty, in which case the status of the server must be set to Idle with the departure of the customer. On the other hand, if the CustomerQueue is not empty, then the next customer in the CustomerQueue can receive service starting at time **t**.

```
eventHandler CustomerDeparture(time t)

       if (the CustomerQueue is empty)
              Set ServerFlag to Idle
       else
              Delete C, the next customer in the CustomerQueue
              Customer C will receive service starting at time t
              Schedule the CustomerDeparture event for customer C
                     at time t+C.service time required
```

Our programming job is not done – the simulation program does not explicitly keep track of customer or server events; so, it does not meet our objectives of determining the average time a customer spends in the system and the percentage of time that the server was busy. This will require adding some accounting variables into our program, initializing and updating them appropriately in the event handlers. For example, to determine the average customer time in the system, we could add two more variables, TotalNumberOfCustomersServed and TotalCustomer TimeIn System, both initialized to 0 and incremented appropriately in the handler for events of type CustomerDeparture. Our

simulation program could then be written to print out the average time that a customer spends in the system, calculated as TotalCustomer TimeInSystem divided by Total NumberOf CustomersServed, before terminating. The percentage of time that the server was busy, or other measurements of interest that one can think of, can be handled in a similar fashion.

Generation of Pending Events

In our implementation of a discrete event simulation program for our study of the railway reservation counter, there are two types of events – CustomerArrival and CustomerDeparture. The events of type CustomerDeparture are generated dynamically within our event handlers. But, how will the events of type CustomerArrival be generated?

One possibility would be to gather and record the required information at the actual railway reservation counter. Our simulation program could then use this data to initialize its 'collection of pending events', say with all of the customer arrival events. This approach is widely used in the simulation of computer memory systems and is known as *trace driven* simulation; a trace is a record of the (timestamped) events observed on an actual system.

Another possibility would be to try and generalize the information contained in a trace. For example, an expert in railway reservation systems might tell you that *the time intervals between customer arrivals* (i.e., customer inter-arrival times) can be modeled as being exponentially distributed. Under this assumption, you could develop the discrete event simulation program using a random number generator to generate the next CustomerArrival event from within the CustomerArrival event handler itself. The use of random number generators in discrete event simulation is common and is known as *stochastic simulation*.

Commonly used random number generators produce sequences of values by applying carefully designed arithmetic expressions, and are therefore called pseudorandom number generators.

How the Railway Counter Simulation Progresses

Let us now consider a small simulation run to better understand

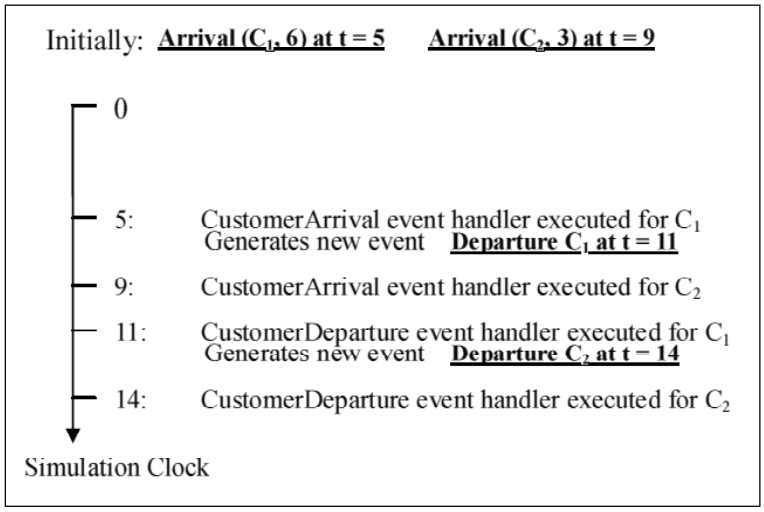

Figure 2. Example of how simulation progresses.

how our discrete event simulation of a railway reservation counter progresses. Suppose that the collection of pending events is initialized with 2 events as shown in *Figure* 2 – the arrival at time $t = 5$ of customer 1 (who will require a service time of 6 units at the counter) and the arrival at time $t = 9$ of Customer 2 (who will require a service time of 3 units at the counter). The figure shows the simulation clock line, running from top to bottom. Events are shown underlined and in bold.

Initially, ServerFlag is Idle and the CustomerQueue empty. The lowest timestamp event is that for the arrival of customer 1 at $t = 5$, as a result of which the CustomerArrival event handler is executed. In this handler, ServerFlag is changed to Busy and a CustomerDeparture event for customer 1 is generated with timestamp $t = 11$. The arrival event for customer 2 will be handled next. As ServerFlag is now Busy, this new customer gets inserted into the CustomerQueue. The Discrete Event Simulation loop next processes the departure event for customer 1 which has timestamp $t = 11$, and so on.

Observe that the Server was idle from $t = 0$ until $t = 5$ and busy from $t = 5$ until $t = 14$. Customer 1 spent time from $t = 5$ until $t = 11$ in the system, while customer 2 spent time from $t = 9$ until $t = 14$ in the system.

One will require adequate knowledge about the system in order to develop a model that is suitable to meet the objectives of the study for which discrete event simulation is being adopted.

Was Our Model too Simple?

A visit to your nearest railway reservation counter will tell you why our model was called 'simple' reality is more complicated. Some railway stations have a single queue leading to a number of service counters. Others have multiple queues each with an associated server. Yet others have counters that give priority to women, senior citizens or foreign tourists. In practice, customers move from queue to queue, or do not join the end of a queue. Further, our simple model makes very simplistic assumptions about service times; service time will vary depending on the efficiency of the server, the availability of tickets, the correct filling up of the reservation form and numerous other factors.

Our simple model was based on one queue with one server; it is often called the *single server queue*. Extending what we have learned so far, a model based on a single queue and multiple servers, or a model using a collection of single server queues, could easily be developed. Many of the other problems listed in the previous paragraph can be addressed by adding more detail to the model. One will require adequate knowledge about the system in order to develop a model that is suitable to meet the objectives of the study for which discrete event simulation is being adopted.

Simulation Packages

Open source and commercial software packages that support discrete event simulation are available. They provide the underlying discrete event simulation algorithm (including the implementation of the 'collection of pending events' data structure), random number generators (required for stochastic simulation), and some basic structures such as the single server queue from which one can construct a simulation model, often using a graphical user interface.

Suggested Reading

[1] J Banks, J Carson, B L Nelson and D Nicol, *Discrete-event System Simulation*, 5th Edition, Prentice Hall, 2010.