Computer Systems Modelling

Discrete Event Simulation

Simon Crosby

Part II, CST, 1996

Discrete Event Simulation

Analytical models must frequently make simplifying assumptions

- Multiple resource holding
- Blocking
- Parallelism (splitting of "customers")
- Assume exponential distributions
- Service disciplines (e.g. Earliest Due Date)
- Adaptive routing
- Concurrency constraints such as locks

We seek an alternative paradigm which can represent arbitrarily complex systems – simulation

- Able to capture the dynamic behaviour of systems
- Captures the dynamics of complex systems by imitation
- Tracks the evolution of the system over time
- Examples include communication network design, road traffic modelling, studying chemical reactions, fluid flow, etc.

Simulation

Execution of simulation model consists of a series of state space changes

These closely follow the chronological order of events in the system being modelled

Consider simulation as a "set of equations" describing evolution in time of system under study

The "equations" are "solved" by following their evolution in time

We **always** follow the **dynamic** evolution of the system, even if we only want a mean value

Simulation is of particular use study systems which are not in a steady state condition

Types of Simulation

- **Discrete event** (sequential) – changes in system state occur only at specific points in time
- **Continuous** – changes occur as a function of time, and evolve with small time increments

We are concerned only with **discrete event** simulation

State variable changes occur at points in time which are called event occurrences

Pros and Cons

The principal advantage of simulation is its extreme generality

There are 4 main problems with simulation:

- The design, coding and debugging of a simulation program is often time consuming and difficult to understand
- Generality can lead to complexity which can obscure understanding of the model
- Execution can be computationally expensive
- Statistical analysis of the output is difficult

Principles of DES

To implement a discrete event simulator we need only 2 data structures:

1. an event list containing all pending events
2. a clock variable to denote how far the simulation has progressed

Frequently most of the simulation time is spent on "housekeeping" to maintain chronological ordering

Events

Each event contains a time stamp denoting "when it occurs"

Denotes some change in the state of the system to be simulated e.g. "packet arrived"

Events are ordered in time in an **event list**

```
Initialize the clock to 0
Initialise the event list
WHILE termination criterion is not met
    remove a smallest tuple (t,m) from the event list
    update the clock to t
    simulate the effect of transmitting m at time t
```

It is **crucial** that the simulator always selects the event with the **least** time stamp

Physical systems

Concerned with the problem of representing so-called "physical systems"

A physical system consists of one or more physical processes

Physical processes are distinguished from one another by virtue of their mutual independence

Interaction between PPs is achieved via messages (events)

All state changes are summarized in the event exchanges

"Correctness"

We need to consider properties which affect the notion of correctness that we intuitively associate with real world systems

The conditions that are required are

Realisability: a message sent by a process at time t is a function of its initial state, t, and the messages it has received up to and including t

Predictability: requires that, given the initial state of a physical system, the output of every sub-process represented as an independent entity in the simulation, up to any time t, can be computed

These conditions also assist us to **construct** the model

An Example – Car Wash

Automatic car wash which is able to service one car at a time

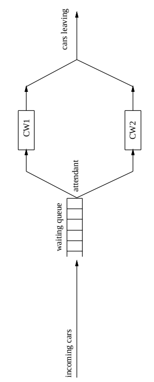
Is it viable to install a second car wash?

The inter-arrival time of cars that need to be washed is randomly distributed

A model of a car wash system consists of an attendant and two car washes, CW_1 and CW_2

If both car washes are busy, an arriving car joins the queue

When a car wash becomes non-busy then the car at the head of the queue is sent to it



Events:

- the arrival of a car at the attendant,
- its arrival at a car wash
- or the departure of a car from the car wash

Given that it takes CW_1 8 minutes and CW_2 10 minutes to wash a car, a *possible* sequence of events is

event number	time	event
1	6	car1 arrives
2	6	car1 arrives at cw1
3	11	car2 arrives
4	11	car2 arrives at cw2
5	12	car3 arrives
6	14	car1 leaves
7	14	car3 arrives at cw1
8	17	car4 arrives
9	19	car5 arrives
10	21	car2 leaves
11	21	car4 arrives at cw2
12	22	car3 leaves
13	22	car5 arrives at cw1
14	25	car6 arrives
15	30	car5 leaves
16	30	car6 arrives at cw1
17	31	car4 leaves
18	38	car6 leaves

The car wash model does not yet match the physical process paradigm

This requires the model to be divided into autonomous components that interact solely via events

The car washes must send events to the attendant stating when they become non-busy

We can then verify the conditions of realisability and predictability

message	event	time	sender	receiver	content
1	-	0	cw1	att	idle
2	-	0	cw2	att	idle
3	1	6	source	att	car 1
4	2	6	att	cw1	car 1
5	3	11	source	att	car 2
6	4	11	att	cw2	car 2
7	5	12	source	att	car 3
8	6	14	cw1	sink	car 1
9	-	14	cw1	att	idle
10	7	14	att	cw1	car 3
⋮	⋮	⋮	⋮	⋮	⋮

Modelling Stochastic Systems

To represent the stochastic nature of the systems being modelled, simulation requires the use of random variables

An input to the simulation may have a stochastic distribution e.g. arrivals at a queue

Simulation model needs to sample random variables from the given distribution to "re-create" the input process

Random variates can be generated for a wide range of theoretical and empirical distributions

This process requires only a sequence of independent random variates with a uniform distribution!

Random Number Generation

Many random number generators in use today are not particularly good

It is important **not** to generate random numbers with a method chosen at random

How can we generate a **random** sequence in a **deterministic manner**?

The answer is that the sequence is not random, but appears random as far as can be determined from statistical tests

In general a deterministic algorithm will repeat the sequence after some time

Important requirements include

- The algorithm should be fast
- The storage requirements should be low
- The random sequence should only repeat with a very long cycle
- The sequence generated should possess two important statistical properties: uniformity and independence

For ease of implementation and speed, most random number generators use integer representation over an interval m

Given a value in this range, a desired value in the range $(0, 1)$ can be obtained by dividing by m.

Midsquare Method

Suggested by von Neumann and Metropolis (1946)

Take the square of the previous random number and extract the middle digits

R_i	R_i^2	R_{i+1}
5772156649	33317792380594909291	7923805949

Sequences which are generated in a deterministic way are called **pseudo-random** sequences

This is a comparatively poor source of random numbers

The sequence tends to "get into a rut"

Linear Congruential Method

Most successful random number generators are special cases of the following scheme, introduced by Lehmer in 1951

It is based on 4 magic numbers:

The starting value X_0	$X_0 \geq 0$
The multiplier a	$a \geq 0$
The increment c	$c \geq 0$
The modulus m	$m \geq X_0, m > a, m > c$

The desired sequence of random numbers X_n is then generated by setting

$$X_{n+1} = (aX_n + c) \text{ modulo } m$$

An Example Sequence

Let $X_0 = a = c = 7$, and $m = 10$

This gives the pseudo-random sequence $\{7, 6, 9, 0, 7, 6, 9, 0, \ldots\}$

What went wrong?

The choice of the values is critical to the performance of the algorithm

Also demonstrates that these methods always "get into a loop"

Multiplicative Congruential Method

The special case where $c = 0$ yields this method

When $c = 0$ the period of the sequence is reduced

But, to our advantage, generation is a little faster

How do we choose the values?

Well, $a = 1$ can be rejected immediately

This would mean that
$X_n = (X_0 + nc)$ modulo m

The case $a = 0$ is even worse.

So $a > 1$ is necessary.

Choice of values

Since the period can never be greater than m this should be chosen to be as large as possible

The following values were chosen for efficient computation of random numbers on the IBM/370

$$\begin{aligned} a &= 16807 \\ m &= 2147483647 \\ c &= 0 \\ X_0 &= 314159 \end{aligned}$$

Knuth has observed that the value a should end in the digits $x21$ where x is even e.g. 31415821

Implementation problem: what about overflow?

Tests for Randomness

Various statistical tests exist

These include tests for uniformity (the Kolmogorov Smirnov test and Chi-Square test)

Runs tests, which examine the arrangement of numbers in a sequence (a run) to test the hypothesis of independence

These tests frequently check for the number of "up runs", the number of "down runs", and the runs above and below the mean.

Autocorrelation tests checks the correlation between every m numbers starting with the i^{th} number.

Kolmogorov Smirnov Test

Compares the continuous cumulative distribution function $F(x)$ of the uniform distribution

to

the empirical cdf S_n of the sample of n observations.

By definition $F(x) = x, 0 \leq x \leq 1$. Assume the sample sequence from the generator is $R_1, \ldots R_n$, then

$$S_n(x) = \frac{\text{number of } R_1, \ldots R_n \text{which are } \leq x}{n}.$$

As $n \to \infty, S_n(x)$ becomes a better approximation to $F(x)$ if the hypothesis is true.

The Kolmogorov Smirnov test is based on the largest absolute deviation between $S_n(x)$ and $F(x)$ over the range of the random variable. That is, based on the statistic

$$D = \max |F(x) - S_n(x)|$$

The sampling distribution of D is known and tabulated as a function of n. For testing a uniform cdf, the procedure is as follows:

1. Rank the data from smallest to largest. Let $R_{(i)}$ denote the i^{th} smallest element, so that $R_{(1)} \leq \ldots \leq R_{(n)}$.

2. Compute for $1 \leq i \leq n$

$$\begin{aligned} D^+ &= \max \left[\frac{i}{n} - R_{(i)} \right] \\ D^- &= \max \left[R_{(i)} - \frac{i-1}{n} \right] \end{aligned}$$

3. Compute $D = \max(D^+, D^-)$

4. Determine the critical value D_α from published tables for the specified significance level of the critical value (ie confidence level) α and the given sample size n.

5. If $D > D_\alpha$ then reject the sample. Otherwise the given sample of random numbers is a good approximation (subject to the critical value α) of the uniform distribution.

Random Variate Generation

We have a sequence of pseudo-random uniform variates. How do we generate variates from different distributions?

Random behavior can be programmed so that the random variables appear to have been drawn from a particular probability distribution

If $f(x)$ is the desired pdf, then consider the CDF

$$F_x(x) = \int_{-\infty}^{x} f(x)dx$$

This is non-decreasing and lies between 0 and 1

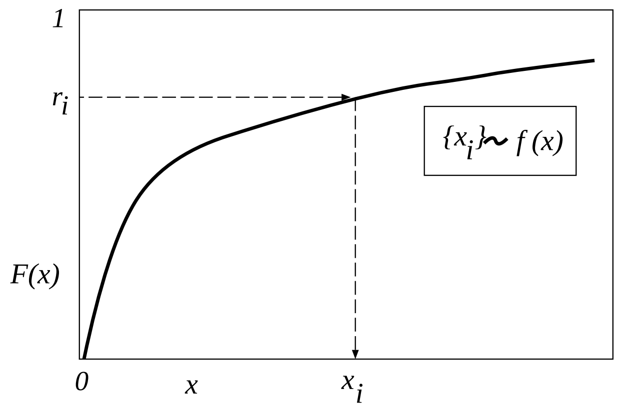
Given a sequence of random numbers r_i distributed over the same range $(0, 1)$

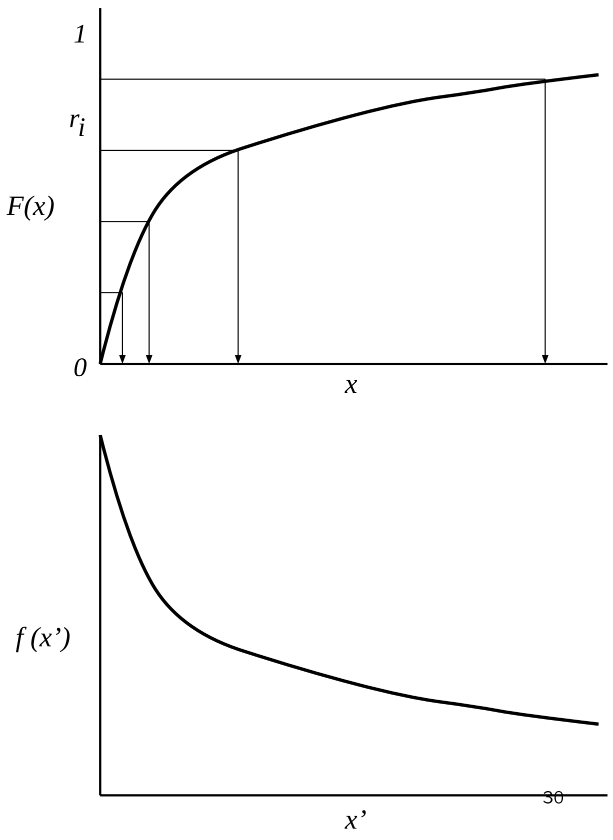
Let each value of r_i be a value of the function $F_x(x)$

Then the corresponding value x_i is uniquely determined

The sequence x_i is randomly distributed and has the probability density function $f(x)$

Random Variate Generation





Uniform Distribution

Consider the **Uniform** random variable on the interval (a, b) where

- a, b are arbitrary real numbers
- $a < b$

The distribution function is

$$F_x(x_i) = (x_i - a)/(b - a)$$

for x_i in the interval (a, b)

Given r_i on interval $(0, 1)$ the problem is that of obtaining x_i

Starting with $r_i = (x_i - a)/(b - a)$ it is obvious that $x_i = (b - a)r_i + a$.

Exponential Distribution

For the exponential distribution

$$F_x(x_i) = 1 - \mathrm{e}^{-\lambda x_i}$$

for positive x_i

Thus

$$\begin{aligned} r_i &= 1 - \mathrm{e}^{-\lambda x_i} \\ 1 - r_i &= \mathrm{e}^{-\lambda x_i} \\ \ln(1 - r_i) &= -\lambda x_i \\ x_i &= -\frac{1}{\lambda}\ln(1 - r_i) \end{aligned}$$

Note that r_i has the same distribution as $1 - r_i$ so we would in reality use

$$x_i = -\frac{1}{\lambda}\ln(r_i)$$

Other random variates can be derived in a similar fashion.

Simulation Performance Measures

As the simulation itself is stochastic, so too will be the output of the observations of a stochastic process

It is critical to realize that a simulation can only yield **estimates** for performance measures

There will **always** be some error in the estimates

The error can be made small by

- running the simulation for a very long time until sufficient samples have been taken
- running the same simulation with a different random number generator, and collecting the results from multiple runs

Utilization

The utilization is the fraction of time that a server is busy

An estimate can therefore be obtained by taking the sum of the busy times of the server and dividing by T

In the case of an k-server, the busy times can be estimated together and divided by kT

There are 2 obvious ways to aggregate the busy times:

- Add the service times at some appropriate point
- When the server becomes busy the time is saved, and when it becomes idle again the difference in the two times is the total busy time.

Throughput, Queue Length

A simple count is kept of the number of customers receiving service. The **throughput** is this value divided by the total time.

There are at least two ways to estimate **queue length**:

- Estimate the queue length distribution, then use to obtain the mean
- View the queue length as a function of time: the mean queue length is integral of this function.

Queue Length

- Each time the queue length changes the time t_{i+1} is stored
- subtract previous recorded time t_i from current time
- multiply by previous queue length n_i

Sum these areas to give mean queue length (for M observations) by

$$\bar{N} = \frac{1}{T} \sum_{i=1}^{M} n_i(t_{i+1} - t_i) \tag{1}$$

Queueing Time

Two obvious ways: to obtain **queueing time**

- observe the queueing times and simply take their average
- simply use Little's law

Independent Replications

Generally assume that

- The modelled system reaches an equilibrium condition,
- The modeller wishes to measure performance of the system in equilibrium.

Crucial to realize that the simulation represents a **random** experiment

The results of a single experiment **may not** be close to the actual performance

Obvious step is to repeat the experiment, and use the average

Independent Replications

Generate several sample paths for the model which are statistically independent and identically distributed.

Reset the model performance measures at the beginning of each replication,

And use a different random number seed for each independent replication

Distributions of the performance measures can then be assumed to have finite mean and variance

With sufficient replications the average over the replications can be assumed to have a Normal distribution

Statistical Analysis of Results

Given that each independent replication of a simulation experiment will yield a different outcome...

To make a statement the about accuracy we have to estimate the distribution of the estimator

Need to determine that the distribution becomes asymptotically centered around the true value

Cannot be established with certainty in the case of a finite simulation

The usual method used to estimate variability is to produce "confidence interval" estimates

Confidence Intervals

Given some point estimate p a we produce a confidence interval $(p - \delta, p + \delta)$

The "true" value is estimated to be contained within the interval with some chosen probability, e.g. 0.9

The value δ depends on the confidence level – the greater the confidence, the larger the value of δ

Let $x_1, x_2, \ldots, x_n$ be the values of a random sample from a population determined by the random variable X

Let the mean of X be $\mu = \mathbb{E}(X)$ and variance σ^2

Assume: either X is normally distributed or n is large

Then: by the law of large numbers, $\bar{X} \approx$ normally distributed

Then, given σ the $100(1-\alpha)\%$ confidence interval is given by

$$\bar{x} \pm \delta$$

where

$$\delta = \frac{z_{\alpha/2}\sigma}{\sqrt{n}} \qquad (2)$$

z_α is defined to be the largest value of z such that $\mathbb{P}(Z > z) = \alpha$ and Z is the standard normal random variable

Can be taken from tables of the normal distribution

for example, for a 95% confidence interval $\alpha = 0.05$ and $z_{\alpha/2} = z_{0.025} = 1.96$

Using Student's T

When we know neither μ nor σ we can use the observed sample mean $\bar{x}$ and sample standard deviation s

If n is large then we simply use s for σ in Equation (2).

If n is small and X is normally distributed then we may use

$$\delta = \frac{t_{\alpha/2} s}{\sqrt{n}} \qquad (3)$$

$t_{\alpha/2}$ is defined by $\mathbb{P}(T > t_{\alpha/2}) = \alpha/2$

T has a Student-t distribution with $n-1$ degrees of freedom

This is the more frequently used formula in simulation models

Issues

- The confidence intervals can only be computed once the replications have already been carried out
- An alternative is to use many shorter replications
- Can divide a single long simulation run into several time periods which can be viewed as independent experiments
- Parallel/distributed simulation is possible

Stopping Rules

If the system is ergodic, we know that "time average" $\approx$ "ensemble average"

Unfortunately in practice we seldom see a run of sufficient length to be sure of this

The method of independent replications permits us to obtain many samples to calculate the ensemble average

Given a performance measure L we can approach the ensemble process by

$$L = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{L_i}{n}$$

Having repeated the experiment n times, we can construct a confidence interval on our measure L

We also need to be aware of transient conditions at the start of the simulation

Whereas large numbers of replications reduce the variance in the sample, each replication requires re-stabilizing the simulation

Can we avoid this?

If we use only a single, long run, and break up our samples into n blocks, each of these can form a sample L_i

What could go wrong with this?

Correlation between successive blocks could mean that we have biased samples.

If the block size is large then the correlation should be small.

Explicit techniques exist to estimate the correlation and obtain the block size

The simulation can be stopped once the estimate of L becomes stable, or once the confidence interval around L becomes sufficiently narrow