Computer Systems Modelling
Discrete Event Simulation

Simon Crosby

Michaelmas Term 1996

1 Introduction

In computer science, as well as in other disciplines, one is often interested in the behaviour of an existing system or a system under design or analysis, or some aspect of that behaviour. Analytical models must frequently make simplifying assumptions and are often unable to capture the dynamic behaviour of systems. Simulation is a technique used to capture the dynamics of complex systems by imitation. A simulator is usually a computer program that performs a simulation. Examples of areas where simulation is used include communication network design, where simulation can be used to find network bottlenecks, traffic patterns, dimension trunk capacities, etc. More generally, simulation is widely used as a design aid for both small and large systems. It is a major application of digital computers. In this brief introduction we will discuss the limitations of analytical models, the simulation paradigm, and important statistical aspects of discrete event simulation. Simulation is a whole field of study in itself, and there are many open research issues which remain to be addressed.

1.1 Limitations of analytical models

Although efficient models can be developed for the computation of exact and approximate solutions to many queueing problems, often the system under study cannot be modelled sufficiently accurately using analytical techniques. Drawbacks of analytical models include the computational and storage requirements of some exact solution techniques, and the simplifying assumptions which must sometimes be made about the underlying statistical properties the system under study. For example, it is unacceptable to assume that the arrival process of data traffic in a computer network will follow a Poisson distribution, because we have measured such systems and know this to be untrue.

Other limitations of analytical techniques include:

- Multiple resource holding: when a transaction is holding more than 1 resource, other transactions must wait before they can secure the resources they require. Analytical models of simultaneous resource possession pose considerable difficulty.
- Blocking: the product form solution which can be achieved for simple queueing networks is often violated when limits are placed on the resources available, for example a limited buffer space in a computer network can cause loss. Analysis of such systems is difficult.
- Parallelism: frequently processes such as daemons in unix, spawn sub-processes which execute in parallel. Analytical models cannot deal with the splitting of "customers".
- Frequently to attain product form solutions it is necessary to assume exponential distributions for service times and inter-arrival times. When this does not hold solution of the analytical model can be extremely difficult.
- Service disciplines frequently found in computer and communication systems often violate product form, making analytical models difficult to solve. Earliest Due Date is an example of a discipline which is analytically intractable.
- Adaptive routing in computer networks where the routing of a packet depends the current load of the links, is analytically intractable.
- Concurrency constraints, such as locks, monitors, semaphores etc make the analytical models violate product form.
- Response time distributions cannot be computed using analytical models of queueing networks.

Several of these problems can be addressed by approximate methods. These are approximate in the sense that their accuracy cannot be defended formally. Frequently the accuracy of approximate methods is within 5 to 10% of the exact results, yet our reservations about the fundamental assumptions of analytical models are often well founded, and we seek an alternative paradigm which can represent arbitrarily complex systems. Simulation is such a paradigm.

1.2 The Need to Simulate

From an implementation viewpoint, a simulation is usually classified as being either discrete event (sequential) or continuous. To permit the use of discrete event simulation it must be possible to view all significant changes to the state of the system being modelled as distinct events that occur at specific points in time; the simulation then achieves the desired behaviour by modelling a sequence of such events, treating each individually. By contrast, a continuous simulation views changes as occurring gradually over a period of time and tracks the progress of these gradual changes. In computer science, discrete event simulation has been applied with particular success and has therefore been studied extensively.

Simulation, as opposed to analytical solutions, consists of a series of state space changes that closely follow the chronological order of events in the system being modelled. A simulation program is one which behaves like the system under study, evolving over time under the influence of external events and internal state modifications. A simulation model can be viewed as consisting of a set of equations describing the state of the system, which are solved by following the evolution of their values over a period of time. In simulation the concept of time is explicit. The simulation always describes the dynamic behavior of a system, even when the system analyst may ultimately be interested in only the mean value of some performance measure in the steady state.

Simulation is of particular use in modelling situations which do not reach steady state – it permits the study of boundary conditions in which extremes of state can be reached.

Computer Simulation

Computer systems are usually seen as discrete systems, which evolve by discontinuous state transitions called events. Their simulation models therefore consist of equations expressing the logical conditions across these transitions occur. Solving the equations by a simulation technique involves determining the chronological sequence the events of interest as they occur in the system and the corresponding sequence of system states. Once a model has been formulated, a simulation program is written which keeps track of the evolution in time of the model as determined by the occurrence of events at discrete time instants.

The principal advantage of simulation is its extreme generality. For example, simulation can be used to model extremely bursty video sources in high speed computer networks, the flow of fluids through pipes, and even human behavior. The principal drawback of simulation is the complexity of the model itself, and the time which it takes to "solve" the model. In a model of an ATM network, for instance, where we expect to observe the loss of a cell (packet) of data with a probability of 10^{-9}, we must simulate the arrival of many times more than 10^9 cells before we have any confidence in our observation of the model under study.

There are 4 main problems with simulation:

- The design, coding and debugging of a simulation program is often time consuming and difficult to understand
- Simulation can be applied to very complex problems, yet the inclusion of complexity can obscure the understanding of the model.
- Execution of the simulation program can be computationally expensive.
- The statistical analysis of the output of a simulation program is non-trivial. Two of the traditional statistical problems are estimation of the accuracy of the results and determination of the run length of the simulation.

We will discuss several techniques which help to overcome the problems with simulation, and hopefully provide a general understanding of the applicability of simulation to modelling problems.

2 Principles of Discrete Event Simulation

Discrete event simulation concerns the modelling of a system as it evolves time by a representation in which state variables change only at a countable number of points in time. These points indicate event occurrences, where an event is defined as an occurrence which may alter the system state. This is in contrast to continuous simulation which deals the modelling over time of a system by a representation in which the state variables change continuously with respect to time. Frequently in a continuous simulation the system is described using differential equations which relate the state variables.

A principal difference between simulation programs and normal programs is that the simulation program must provide the timing mechanism for the simulated system and take simulated time into consideration in its actions. Much of processing in a simulation program is spent updating the internal data structures which reflect the current state of all entities in the model of the system under study, adjusting this state in response to events, ordering the list of events which need to be stimulated and sampling the statistical properties of the system under study.

The result of a simulation run is a set of observed statistical results describing the behavior of the simulated system during the simulation.

3 The Discrete Event Simulation Algorithm

In discrete event simulation we identify beforehand the set of events which cause state transitions of interest in the simulated system. We track the evolution of state only at the occurrence of these events.

The fundamental requirement of a simulation program is a clock which keeps track of the simulated time. At the start of the simulation, the simulation clock is initialised to 0 and the times of the intial set of future events are determined. The simulation clock is then advanced to the time occurrence of the most imminent of these events. The state changes corresponding to the event are then simulated by the simulation program, which can cause the addition of more events to the list of future events which will occur. Once the simulation of an event is complete, the clock is advanced to the time of the next event, where the process is repeated.

In this way, idle time in the system under study, or time which is spent in the real system on events which are not of interest, is skipped, and only events of interest are tracked.

The simulation algorithm then, requires two basic variables: the clock and a future events list which is ordered by future event time. Events on the future events list are sorted according to their scheduled occurrence time – events which should occur sooner are closer to the head of the list.

With the simulation of each event, more events are added to the future events list. The simulation ends when the future events list is empty or some predicate on the system state holds.

3.1 Physical systems

Simulation is concerned with the problem of representing so-called "physical systems", that is, we restrict our attention to a modelling paradigm which is explained as follows: A physical system consists of one or more physical processes. A Physical process denotes some component of the real system to be simulated. Physical processes are distinguished from one another by virtue of their mutual independence other than for the specific purpose of interaction, which is done solely via messages (or events, denoting the changes in important system state) that are exchanged between them.

In order to be able to define the meaning of simulation for physical systems, we need to consider properties which affect the notion of correctness that we intuitively associate with real world systems. For example, it is intuitively clear that no event in the future may affect an event that happened in the past (causality). In identifying these properties, the constraints imposed on any simulator become visible.

3.1.1 Realisability and predictability

The conditions that are assumed to be satisfied for all physical systems considered, are *realisability* and *predictability*:

Realisability states that 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. It essentially means that a physical process does not know ahead of time the contents of any message that it will receive in the future, which implies that the future cannot affect the past. This property is central to correctness, since it is an intuitive requirement satisfied by any real system.

Predictability requires that, given the initial state of a physical system, the output of every sub-process represented as an independent (concurrent) entity in the simulation, up to any time t, can be computed. It guarantees that the system is well defined in the sense that the behaviour of every sub-process is determined at any given time.

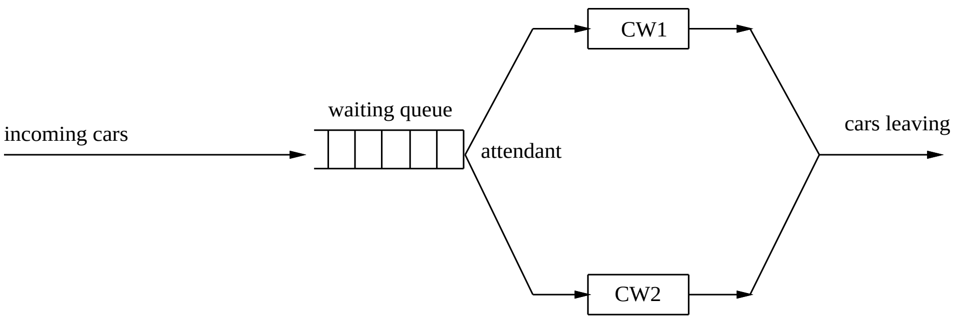

Figure 1: Model of a carwash

3.1.2 An Example

Consider the case of a garage owner who has installed an automatic car wash which is able to service one car at a time. Suppose that the popularity of the service increases such that she has to face the following problem: As more and more customers use this car wash, their average waiting time will tend to increase and the service becomes less attractive. The garage owner could simply do nothing about the problem and let the customer flow stabilise itself at a lower level. On the other hand she could install more automatic car washes which will keep the present and future customers happy. But the latter is not an easy decision because automatic car washes are expensive.

This is the point where simulation can help. Simulation is a technique used for representing a dynamic system by a model, and performing experiments with the model in order to gain more information about the system. A simulation approach is useful if the system is too complex for the use of analytical techniques and direct experimentation with the system is not practicable (for example installing additional car washes and observe what happens).

We now formulate a model of the car wash system with two car washes and will show how to view it within the scope of physical processes and discrete event simulation (also called message passing).

A model of a car wash system consists of an attendant and two car washes, CW_1 and CW_2. The inter-arrival time of cars that need to be washed is randomly distributed. The attendant sends an arriving car to a non-busy car wash. If both car washes are busy, any arriving car joins the queue at the attendant. When a car wash becomes non-busy then the car at the head of the queue is sent to it. We are interested in the behaviour of this system over time, for example, the average length of the queue that builds up at the attendant. In this model an event is either 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 shown in table 1. However, the car wash model given does not yet match the physical process paradigm which requires the model to be divided into autonomous

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

Table 1: Possible sequence of events

components that interact solely via events. Furthermore, an event has always a sender, a receiver, a time value and a event content. To fulfill the requirements of the physical system paradigm, the car washes must send events to the attendant stating when they become non-busy. The flow of events is depicted in Figure 2 A resulting sequence of messages corresponding to the possible sequence of events in Table 1 is given in Table 2.

Let us now verify realisability and predictability in the car wash model. The output at time t of a physical process only depends on the messages it has received up to t (realisability). The attendant, for example, given a "non-busy" message from a car wash at time t and given the queue is non-empty, sends a message to that car wash at time t. Predictability holds because any cycle in the message flow contains one of the car washes cw_1 or cw_2 and, given we know their output up to time t, we are able to predict the output up to time $t+8$.

4 Modelling Stochastic Systems

To represent the stochastic nature of the systems being modelled, simulation requires the use of random variables. Given that the input to (part of) a model is a stochastic variable following a particular distribution (for example the arrival process at a queue), the simulation proceeds by generating or sampling values from that distribution to represent the input process. We thus require the ability to generate random variates from different

message	event	time	sender	receiver	content
1	-	0	cw1	attendant	non-busy
2	-	0	cw2	attendant	non-busy
3	1	6	source	attendant	car 1
4	2	6	attendant	cw1	car 1
5	3	11	source	attendant	car 2
6	4	11	attendant	cw2	car 2
7	5	12	source	attendant	car 3
8	6	14	cw1	sink	car 1
9	-	14	cw1	attendant	non-busy
10	7	14	attendant	cw1	car 3
11	8	17	source	attendant	car 4
12	9	19	source	attendant	car 5
13	10	21	cw2	sink	car 2
14	-	21	cw2	attendant	non-busy
15	11	21	attendant	cw2	car 4
16	12	22	cw1	sink	car 3
17	-	22	cw1	attendant	non-busy
18	13	22	attendant	cw1	car 5
19	14	25	source	attendant	car 6
20	15	30	cw1	sink	car 5
21	-	30	cw1	attendant	non-busy
22	16	30	attendant	cw1	car 6
23	17	31	cw2	sink	car 4
24	-	31	cw2	attendant	non-busy
25	18	37	cw1	sink	car 6
26	-	37	cw1	attendant	non-busy

Table 2: Possible sequence of messages

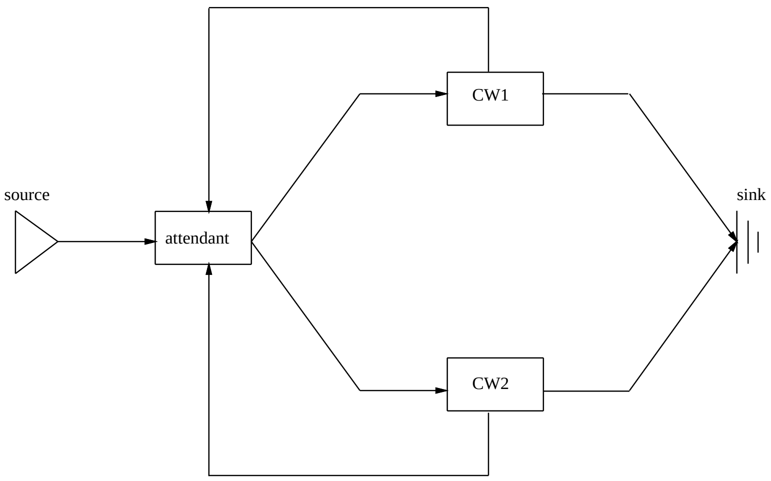

Figure 2: Message flow in car wash model

distributions. It is a remarkable fact of probability theory that variates can be generated for a wide range of theoretical and empirical distributions provided only that a sequence of independent random variates each with a uniform distribution on the interval $(0, 1)$ can be generated. The following subsections discuss the generation first of random numbers, and then random variates from different distributions.

4.1 Random Number Generation

Many random number generators in use today are not particularly good. It is important not to attempt to generate random numbers with a method chosen at random. Important practical considerations of a pseudo-random number generator are:

- 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 a variety of reasons most random number generators utilize 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.

4.2 Midsquare Method

Suggested by von Neumann and Metropolis (1946): the idea was to take the square of the previous random number and extract the middle digits, for example:

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

How can such a sequence be random, since each element is completely determined by its predecessor? The answer is that the sequence is not random, but appears random as far as can be determined from statistical tests. Sequences which are generated in a deterministic way as this one is, are called pseudo-random sequences.

This method has proved to be a comparatively poor source of random numbers. The danger is that the sequence tends to "get into a rut", a short cycle of repeating elements. For example if 0 ever gets into the middle number of the sequence, it will continually perpetuate itself.

4.3 Linear Congruential Method

Many of the most successful random number generators used today 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

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

For example with a choice of $X_0 = a = c = 7$, $m = 10$ the sequence $7, 6, 9, 0, 7, 6, 9, 0 \ldots$ is obtained.

This clearly demonstrates that the choice of the values is critical to the performance of the algorithm. It also demonstrates that these methods always "get into a loop" and repeat themselves after some period. We need to ensure that the choice of values gives a large period.

4.4 Multiplicative Congruential Method

The special case where $c = 0$ deserves explicit mention. When $c = 0$ the period of the sequence is reduced, but the generation is a little faster. This is called the multiplicative method as opposed to the linear congruential method. How do we choose the values? Well, $a = 1$ can be rejected immediately – this would mean that $X_n = (X_0 + nc)$ modulo m – this would definitely not be random. The case $a = 0$ is even worse. So $a > 1$ is necessary.

Since the period can never be greater than m this should be chosen to be as large as possible. A factor influencing the choice of m is the speed of generation. The following values were chosen for efficient computation of random numbers on the IBM/370, and seem to have been perpetuated until now:

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

Knuth has observed that the value a should end in the digits $x21$ where x is even because this avoids certain particularly troublesome cases which have been discovered using analytical techniques. The problem of overflow must be taken into account when generating random numbers. Repeatedly multiplying large numbers can lead to overflow because of the limited word size on all computers. Overflow can cause unpredictable results, but can be avoided by clever coding – see Sedgwick's Algorithms book for details.

4.5 Tests for Randomness

Various statistical tests exist to test the fidelity of a pseudo-random sequence. A basic test that should always be performed to validate a generator is the test of uniformity. The Kolmogorov Smirnov and Chi-Square tests measure the degree of agreement between the distribution of a sample of generated random numbers and the theoretical uniform distribution. The null hypothesis in the tests is that the stream of data is uniformly distributed on the interval [0, 1]. Runs tests 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" (non-decreasing sequences), the number of "down runs" (non increasing sequences), and the runs above and below the mean. Autocorrelation tests checks the correlation between every m numbers starting with the i^{th} number.

4.6 Random Variate Generation

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 $(0, 1)$ each value of r_i is considered to be a value of the function $F_x(x)$, and the corresponding value x_i is determined. The sequence x_i is randomly distributed and has the probability density function $f(x)$.

4.7 Uniform Distribution

Consider the Uniform random variable on the interval (a, b) where a, b are arbitrary real numbers, with $a < b$. 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 $x_i = (b - a)r_i + a$.

4.8 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.

5 Simulation Performance Measures

In many simulation studies a large amount of time and money is spent on model development and programming, yet little effort is made to analyze the simulation output in an appropriate manner. As the simulation itself is stochastic, so too will be the output of the observations of a stochastic process, and the appropriate tools for statistical interpretation are required.

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 either running the simulation for a very long time until sufficient samples have been taken, or by running the same simulation with a different random number generator, and collecting the results from multiple runs. The following subsections describe how several common performance measures can be made from a simulation whose elapsed time is T.

5.1 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 the utilization of each determined by dividing the aggregate busy time by kT. There are 2 obvious ways to aggregate the busy times:

- Add the service times at some appropriate point, for example when a job begins or ends service.
- 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.

5.2 Throughput

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

5.3 Mean Queue Length

There are at least two ways to estimate this:

- Estimate the queue length distribution, then use this to obtain the mean. The distribution is estimated by keeping a histogram with a bucket for each potential value of the queue length. Each time that value is observed, the bucket value is incremented.
- View the queue length as a function of time: the mean queue length is integral of this function. Each time the queue length changes the time t_{i+1} is stored. The previous recorded time t_i is subtracted from the current time. This difference is multiplied by the previous queue length n_i. The product is then added to the sum of the area. The mean queue length (for M observations) is then given by

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

5.4 Mean Queueing Time

Two obvious ways exist to compute the mean queueing time: observe the queueing times and simply take their average as an estimate of the mean or simply use Little's law.

6 Statistical Analysis of Simulation Results

6.1 Confidence Intervals

Each independent replication of a simulation experiment will, in general, yield a different outcome which can be used to estimate the true value of a performance measure. To make a statement the about the accuracy of simulation results one has to estimate the distribution of the estimator. One also needs to determine that the distribution becomes asymptotically centered around the true value. Neither of these can be established with certainty in the case of a finite simulation.

The usual method used to estimate the variability of simulation results is to produce "confidence interval" estimates. Given some point estimate p a confidence interval $(p - \delta, p + \delta)$ is produced. The "true" value is estimated to be contained within the interval with some chosen probability, for example 0.9. This probability is known as the confidence level. 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 which has a finite mean $\mu = \mathbb{E}(X)$ and variance σ^2. Suppose that either X is normally distributed or n is large enough that by the law of large numbers, $\bar{X}$ can be considered to be normally distributed. Then, given σ the $100(1-\alpha)\%$ confidence interval is given by

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

where

$$\delta = \frac{z_{\alpha/2}\sigma}{\sqrt{n}} \tag{2}$$

and 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. This can be taken from tables of the normal distribution. Thus, for example, for a 95% confidence interval $\alpha = 0.05$ and $z_{\alpha/2} = z_{0.025} = 1.96$ since this is the value of z corresponding to the standard normal variable at the point 1-0.025 = 0.975.

When we know neither the mean nor the standard deviation σ we can use the observed sample mean $\bar{x}$ and sample standard deviation s in place of μ and σ respectively. If n is large then we simply use $s, \bar{x}$ for σ, μ in Equation (2). However, if n is small and X is normally distributed then we may use

$$\delta = \frac{t_{\alpha/2}s}{\sqrt{n}} \tag{3}$$

Where $t_{\alpha/2}$ is defined by $\mathbb{P}(T > t_{\alpha/2}) = \alpha/2$ and T has a Student-t distribution with $n-1$ degrees of freedom. This is the more frequently used formula in simulation models.

6.2 Independent Replications

In most modelling exercises the following two assumptions are made

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

Occasionally the modeller may be interested in transient behaviour of the system, in which case the equilibrium conditions do not apply. In both cases, however the modeller must realize that the simulation represents a random experiment and that the results of a single experiment may or may not be close to the actual performance of the real system. The obvious step is to repeat the experiment several times, and use the average of the experimental results as the final estimate.

Independent replications of the same experiment are the easiest way of gaining more confidence in the results. Essentially the method involves generating several sample paths for the model which are statistically independent and identically distributed. This is achieved by resetting the model performance measures at the beginning of each replication, but using a different random number seed for each independent replication.

If identical replications of the same experiment are made, distributions of the performance measures can be assumed to have finite mean and finite variance. The former is almost certain for systems which do reach equilibrium, and is always a safe assumption, whilst the assumption of finite variance is somewhat harder to justify but will generally apply to computer systems modelling problems. If the number of replications is sufficiently large then the average over the replications can be assumed to have a Normal distribution, which enables the computation of confidence intervals.

Several problems arise:

- The confidence intervals can only be computed once the replications have already been carried out. Typically these will be simulation runs of some limited duration, after which the results are collected. If the confidence intervals derived from the multiple replications are unsatisfactory, then the simulations must be continued. This is frequently impossible.
- An alternative is to use many shorter replications, to gain a larger sample space for the confidence intervals. This has the drawback that shorter simulations are more likely to have higher variance and to be non-stationary (i.e. to not reach equilibrium). Several techniques exist to determine when a simulation program has achieved statistical equilibrium and sampling may begin.

Yet another alternative is to divide a single long simulation run into several time periods which can be viewed as independent experiments – to do this successfully it is important that the simulation state is the same at each boundary. Using the Markov assumption the future behaviour can then be assumed to be independent of the past. Of course we then have the problem that the simulation must run on a single processor for a long period of time. Parallel discrete event simulation is a separate and active topic of research.