

Simulation

An introduction to the important ideas of simulation, an important technique in call-centre planning.

Michael Tanner

Copyright Mitan Ltd. 2004

Introduction

By simulation we mean working out the series of events that occurs when a system deals with a series of customers. This requires us to write a computer program that mimics the behaviour of the systems, so obviously a good knowledge of the workings of the system is essential. For many analysts a great advantage of simulation is that it requires less (but not zero!) mathematical and statistical competency than queueing theory. Simulation has some more important advantages. Complex arrival patterns and scheduling rules can be represented, and system-wide effects for systems with multiple service centres can be handled relatively easily. On the other hand, a queueing theory analysis can provide much more insight into the performance characteristics of a system than simulation.

The actual programming and testing of a simulation is also a non-trivial task, since the structure of a simulation program is akin to a real-time operating system, and not the straightforward structure that is typical of application programs. For this reason, specialised simulation languages have been developed. On the other hand, with a suitable subroutine library, it is perfectly possible to write significant simulations using general-purpose languages such as Pascal. It used to be said that simulation was expensive in terms of computer time, since a simulation must be repeated a number of times for each set of parameters, and each run is a "number crunching" sort of job. But with the cost of modern computing power this is not such an important consideration.

Basic Ideas of Simulation



To see what simulation involves we will simulate a single server queue. First of all, when will each customer arrive? For our simulation we shall generate random inter-arrival times by throwing a single die. Each throw of the die is the time before the next customer arrives. The results are shown in Table 1, where the first column shows the inter-arrival times, and the second column shows the arrival times. The second column is just the cumulative sum of the first column.

Next we need to assign a specific service time to each customer. We shall use the die again to do this. In Table 1 the third column contains the service times. (Astute readers will notice that the average inter-arrival time is 3.5, which is also the average service time. This system is not stable, since the server utilisation will be 100%, and for stability we need utilisation to be strictly less than 100%. We shall ignore the implications of this in return for the simplicity of using a die to generate inter-arrival and service times.)

k Customer	I[k] Inter-arrival time	A[k] Arrival time	S[k] Service time
1	2	2	5
2	3	5	3
3	1	6	1
4	6	12	4
5	3	15	2
6	4	19	3
7	1	20	6
etc.			

Table 1. Simulated arrival and service times.

Now we have all the details about the customers that we need, and we can proceed to work out the timing of events in the system. For our simple example this is easy, but in a real simulation of a complex system the analysis of how the system works will be a substantial and challenging piece of work. In Table 2 we have added three more columns. Two of these columns are used to hold the times at which each customer begins and ends service. A customer cannot begin service until that customer has arrived, and until the previous customer has ended service. So for customer k we have the simple relationships

$$B[k] = \max(A[k], E[k-1]) \quad \text{where } E[0] = 0$$

$$E[k] = B[k] + S[k]$$

The final column that has been added in Table T27.02 holds the waiting time for each customer. This is calculated in the obvious way

$$W[k] = B[k] - A[k]$$

Using these simple relationships we can calculate the times of all relevant events in the system, and observe the values of waiting times, queue lengths, busy-period lengths, or whatever characteristic interests us.

k Customer	I[k] Inter-arrival time	A[k] Arrival time	S[k] Service time	B[k] Begin service	E[k] End service	W[k] Waiting time
1	2	2	5	2	7	0
2	3	5	3	7	10	2
3	1	6	1	10	11	4
4	6	12	4	12	16	0
5	3	15	2	16	18	1
6	4	19	3	19	21	0
7	1	20	6	21	27	1
etc.						

Table 2. Calculating start, finish, and waiting times

Random Numbers

In the example above we used a die to generate random inter-arrival and service times. Within a simulation program how do we get random numbers? This is a subject, who has received much attention by researchers, and simple techniques have been devised to generate sequences of "pseudo-random" numbers. Most algorithms for generating a sequence of random numbers are of the form shown below.

$$X[n] = (bX[n-1] + c) \text{ modulus } m$$

Where $X[n]$ is the n 'th number in the sequence while b , c and m are appropriately chosen constants. For a discussion of suitable choices of these constants see [LAVEN]¹ or specialised books on simulation. The relationship above will produce numbers that are non-negative integers in the range 0 to m , so by using $X[n]/m$ we can get values in the range 0 to 1.

Pseudo-random number streams can be subjected to a number of tests to see if they have the characteristics of a genuinely random sequence. The obvious test is that the numbers should be evenly distributed over the range of values, usually 0 to 1, required. Another test is that there should be no correlation between values separated in the sequence by a fixed lag i.e. there should be no serial correlation. This test includes the case of trends and cycles. Other, more esoteric, tests can be applied to exclude various patterns in the sequence.

Simulation requires a random number generator that has been properly designed and tested. The actual programming of a generator may be quite trivial, but the choice of method and parameters is a specialised job. Special-purpose simulation languages can be expected to have random number generators that can be trusted. However, many general-purpose language compilers include a random number generator. Some of these may be well designed, on the other hand some may be intended just for use in computer games. For a computer game, it doesn't matter too much if the sequence of numbers would fail proper statistical tests.

Converting Random Numbers to Particular Distributions

Assuming we have a reputable random number generator that supplies us with values in the range 0 to 1, we then have to somehow convert these into, say, exponentially distributed values, or some other distribution that we want to simulate. Several techniques have been developed for doing this. The basic method is to mathematically invert the cumulative probability distribution, but less obvious methods turn out to be better in some ways for some distributions. Again see [LAVEN] for a detailed description and algorithms for specific distributions.

Analysis of Simulation Results

Analysts with limited statistical training often undertake simulation. Because of this it is sometimes forgotten that the result of a simulation run is just one experimental observation, and an observation of an abstract model rather than the real system. Like all experiments, a simulation needs to be properly designed to answer the question of concern, and to be repeated a number of times (with different sequences of random numbers of course!) to provide a sample of results for analysis.

¹ [LAVEN] Lavenberg S. S. (ed.). *Computer Performance Modelling Handbook*, Academic Press, London, 1983

Simulation of a Simple System

In order to illustrate some ideas about simulation, an simple queueing system was simulated with a single server, random arrivals, and “exponentially distributed” service times. We don't need to simulate this system (known as an M/M/1 system), since simple formulae are available for all the characteristics of interest, but this does mean that we can compare the simulation results with the exact theoretical calculations.

Figure 1 shows a graph constructed from the trace of the simulation program. The horizontal axis is time, and the vertical axis shows the queue size (number waiting plus number being served) when a new customer arrives. This number does not include the arriving customer. The curve looks quite erratic, and it is worth reminding ourselves that a stable queueing system will still exhibit this kind of erratic behaviour.

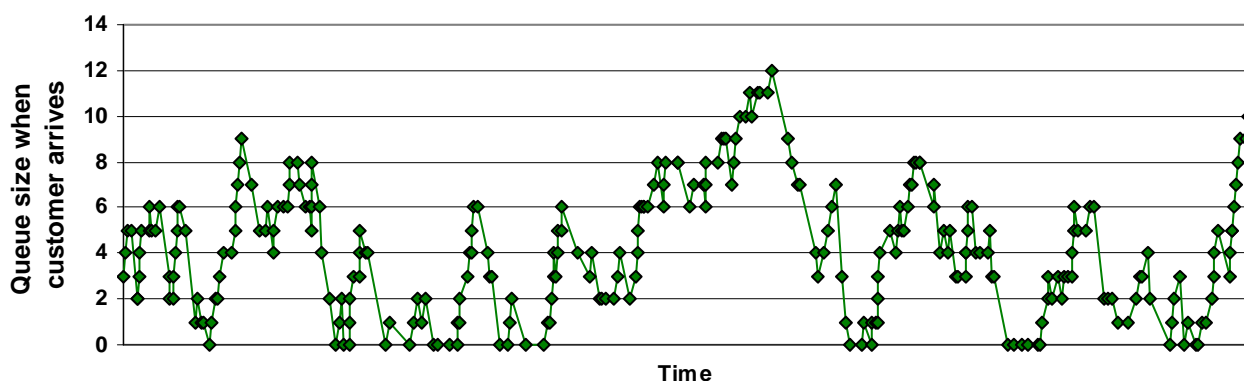


Figure 1 Extract from a simulation run

Next observe that the extract selected starts at time 300, not 0. When we start up a simulation program the simulated system will be empty, and it will require quite a lot of customers to be processed before the system achieves a "steady-state". So we need to run the simulated system for a "warm-up" period before we can start collecting statistics such as waiting time. How long should the warm-up time be? The answer of course is long enough for a steady state to be reached. There are formal statistical tests that could be used to decide when a steady state is reached, but in practice knowledge of the real system being simulated will often suggest a suitable warm-up time. There are also techniques to speed up the reaching of a steady state. We could set up the first few customers to arrive at time zero, so there is a queue of customers at the start. Or we could have a faster arrival rate for, say, half the warm-up period. These are refinements, the analyst still has to check somehow that the system reached steady state before statistics gathering starts.

On the other hand, maybe we are not actually interested in the steady-state behaviour. We may want to understand how the system performs when starting "cold", or dealing with short-term high arrival rates. Some real systems may never achieve a steady state. For example, a shop or store may not have enough customers during the day to make the concept of an average arrival rate meaningful. Most of the queueing theory in this book is about steady-state behaviour, which cannot be applied to such situations. Simulation, on the other hand, can cope with overloaded and unstable queueing systems, with changing arrival rates. Analysis of the results of simulating unstable systems needs great care, but at least is possible.

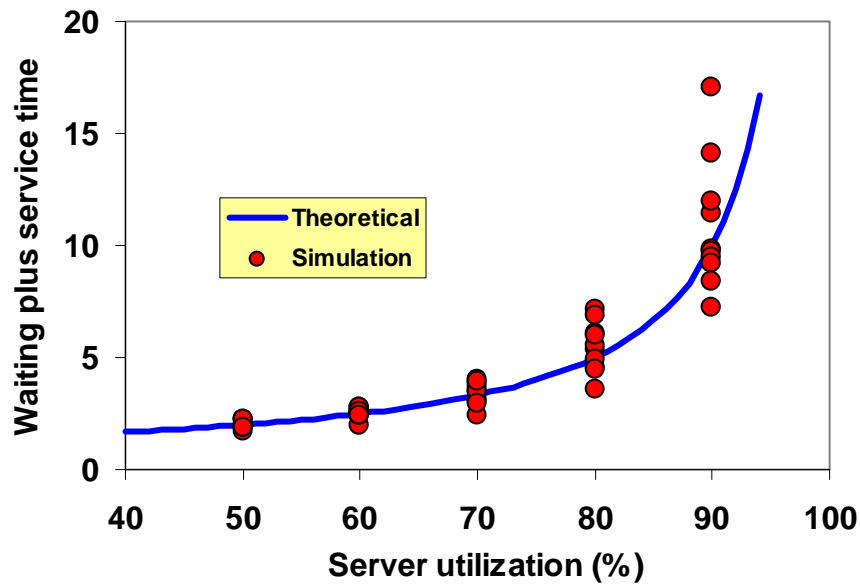


Figure 2. Results of multiple simulation runs.

Now look at Figure 2. This shows the average time in system as measured for a series of separate simulation runs. For each level of utilisation 10 separate simulation runs were done, each run for a given utilisation using a different sequence of random numbers. Also shown is the exact theoretical result. The important point is that a single run produces one point that is not in itself reliable. It is necessary to do several runs and take the average of the results. Figure 3 shows the averages for the 10 runs at each utilisation value. The match between simulation and exact theory is clearly much better. Note also that the variance or spread of results is greater for higher utilisations. This is to be expected, since queueing theory tells us that the variance of time in system increases with server utilisation. The need to take averages over a number of runs is an elementary point, but it is not uncommon for analysts to attach too much significance to a single simulation run.

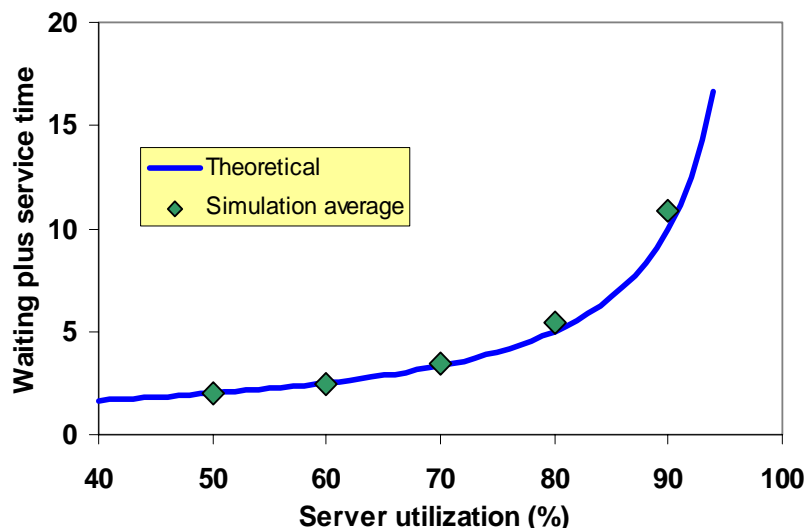


Figure 3. Averaging 10 simulation runs

One question we have not addressed so far is how long should each simulation run be? We know we need a warm-up time to get to a steady state, but we have seen in Figure 1 how the queue size varies even when statistically a steady state exists. The simple, but not necessarily helpful, answer is that enough customers need to be processed so that the variance of \bar{W} , for example, the average waiting time for one simulation run is not too high. Since we are going to take the average of the averages for several runs, there is a trade-off between doing fewer but longer runs and more but shorter runs. Doing more, but shorter, runs has the advantage that, if a distinct random number stream is used for each run, statistically more reliable results are obtained. However, fewer longer runs tends to be more convenient in practice.

Exercise

Construct a table like Table 2, and perform a simulation using a single die to generate inter-arrival and service times. Add a column to represent the waiting-line size when the customer arrives.