Fast Simulation of Open Queueing Systems

Cortney S. Hunt
University of Oklahoma
School of Industrial Engineering
Norman, OK 73019
(405) 325-3721 (W)
e-mail cshunt@essex.ecn.uoknor.edu

Bobbie L. Foote
University of Oklahoma
School of Industrial Engineering
Norman, OK 73019
(405) 325-3721 (W)

We describe a technique developed by the authors for fast simulation of open queueing networks. The technique takes advantage of the recursive nature of departure times of customers in various parts of the system. The event calendar is circumvented using these recursive relationships whenever possible. A framework for identifying these recursive aspects of a network is presented. The technique involves identifying the servers in the system at which flows merge or diverge. Knowledge about the merge and diverge points defines the dependency relationships a specific customer has with other customers at each point in the system. The concept of a system level is based on these dependency relationships.

The technique is implemented in the object oriented language SmallTalk 80. The implementation utilizes Windows interfaces and allows graphical analysis of the simulation results. The implementation is shown to achieve significant reductions in execution time compared with the traditional discrete-event approach to simulation of such networks in most cases. Several different types of systems are explored in an attempt to characterize those systems on which the technique functions well.

Keywords: Fast simulation, queueing systems, object-oriented modeling

Introduction

Perhaps one of the biggest hindrances to simulation users is the intensive computer time simulation requires (Law, 1991). In an attempt to alleviate the problem, parallel processing algorithms have been developed (Kamath and Bhuskute, 1991), various schemes for handling event lists have been implemented, and other approaches have been proposed for making simulation faster. Extensive work has been done on queueing models in an attempt to avoid simulation entirely by deriving mathematical approximations of various performance measures for certain classes of systems (Whit, 1983) (Walrand, 1988).

The initial work on using recursion relationships in simulation was done by Chen (1990 & 1993). Chen presented a Fast Simulator for simple tandem queueing systems. Chen's Fast Simulator completely avoided use of an event calendar by considering only simple tandem lines. We present an approach which uses an event calendar but which reduces the number of events which it must process.

The reduction in the number of events required is achieved by using the same recursive relationships developed by Chen. However, we show that these recursive relationships hold not just for simple tandem lines, but for all servers within what we define as a system level. The process of recognizing the levels in the system to be simulated does require some pre-simulation processing time. This additional analysis time is relatively small compared with simulation execution time.

The simulation approach presented here bridges the gap between the extremely fast but limited simulation of simple tandem lines proposed by Chen and the slow but general, discrete event approach.

Pre-Simulation Analysis

In order to take advantage of the recursive relationships within a system we must isolate those parts of the system where the recursion exists. We define the level boundaries in each customer routing. This concept is explained in the next section. We group the servers associated in levels and create appropriate software objects to represent these groupings.

System Levels

The boundaries of a *system level* are defined by merge points in the system. A *merge point* is simply any server which has more than one predecessor server. A customer flowing through the system is said to be in the same level from the time it enters a merge point queue until the time it encounters the next merge point.

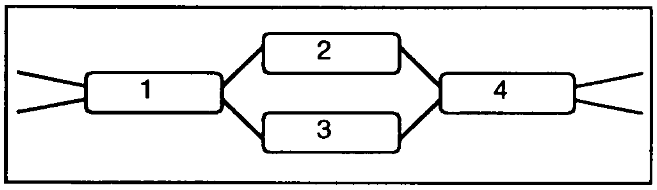


Figure 1. System Levels and Merge Points

In Figure 1 each of the four nodes represent any number of servers in a simple tandem line. Nodes 1 - 3 make up system level one for this system. Node 4 is considered a level two node. Notice that node 4 is *directly* dependent on nodes 2 and 3, and is *indirectly* dependent on node 1.

Definition 1: The level (l_k) of a node k is defined as one greater than the maximum level of any node in the set of preceding nodes P on which it is directly dependent if the size of set P is greater than 1.

$$l_k = \underset{i \in P}{\mathrm{Max}}(l_i) + 1 \qquad (1)$$

Node 1 in Figure 1 is by definition a level one node. Nodes 2 and 3 are also level one nodes because they have only one predecessor node (size of the predecessor set P is 1). Node 4 is a level two node which is calculated as follows.

$$l_4 = \mathrm{Max}(1,1) + 1 = 2$$

A system level comprises all nodes with identical level designations. We use the system level concept in Definition 2 to express the relationship between customers flowing through first-come-first-served queues. Equation 2 is an extension of that presented by Chen (1990), in that for a simple tandem line he defined $d_{i,j}$ as the departure time of customer j from server i where there was only one customer routing. In our definition the ith server in a routing may represent a different server for each routing.

Definition 2: Departure times of customers within a system level may be expressed in terms of the following recursive relationship.

$$d_{i,j} = \mathrm{Max}(d_{i-1,j}, d_{i,j-1}) + s_{i,j} \qquad (2)$$

where
$d_{i,j}$: departure time of customer j from the ith server in its routing in the present level
$s_{i,j}$ = service time of customer j on the ith server in its routing in the present level

Using these recursive relationships we can completely avoid processing events within a level. In the next section we show how events need only be processed at the level boundaries of a system.

Kamath (1991) was the first to develop an algorithm for finding recursive departure times of customers in simple tandem lines with parallel server nodes. The technique involves tracking the next departure time of all customers at parallel servers. When at least one server is idle, the departure time of customer j from server i is simply its departure from the previous server plus its service time. When all servers are busy the algorithm finds the next customer to depart and calculates the departure time based on that customer. The algorithm below is adapted from Kamath (1991).

Parallel Server Algorithm
Variable Definitions:
- N - Total number of customers
- M - Total number of nodes
- NS_i - Number of servers at node i
- $s_{i,j}$ - Processing time for customer j at node i
- $a_{i,j}$ - Arrival time for customer j at node i = $d_{i-1,j}$
- P_j - Previous customer served at the same server as customer j
- BS_j - Number of busy servers at a node when customer j arrives
- $b_{i,j}$ - Beginning time for customer j at node i
- $d_{i,j}$ - Departure time for customer j from node i

$SEQ_{i,j}$ - New sequence of j customers after processing at node i
($i = 1, 2, ..., M$ and $j = 1, 2, ..., N$)

Algorithm: At a parallel server node calculate the departure times from the nodes as follows.

```
Initialize: $Temp_j = 0$ and $SEQ_{i,j} = 0$ for $j = 1, 2, ..., N$
For all j
   1. Find $BS_j$ and $P_j$ using Procedure 1
   2. If $BS_j < NS_j$ then
         $b_{i,j} = d_{i-1,j}$ and $d_{i,j} = d_{i-1,j} + s_{i,j}$
      else
         $b_{i,j} = \max(d_{i-1,j}, d_{i,Pj})$ and $d_{i,j} = \max(d_{i-1,j}, d_{i,Pj}) + s_{i,j}$
   3. Update $Temp_j$ and $SEQ_{i,j}$ using Procedure 2
   4. For $j = 1, 2, ..., N$; $d_{i,j} = Temp_j$
```

Procedure 1: To find P_j and BS_j, the previous customer served and the number of busy servers.

```
1. Initialize BS = 0; Min = $d_{i-1,j}$ ; $P_j = 0$
2. For k = 1, 2, ..., j-1
   {If ($d_{i,k} > d_{i-1,j}$) then
         {BS = BS + 1;
         If ( Min > $d_{i,k}$ ) then Min = $d_{i,k}$ ; $P_j$ = k;}}
3. $BS_j$ = BS
```

Procedure 2: To update $Temp_j$ and $SEQ_{i,j}$

```
1. Set k = j
2. While (k <> 1) {
   If ($d_{i-1,j} > Temp_{k-1}$) then k = k - 1
   else k = 1}
3. If (k <> j) then
   For h = j, j-1, ..., k do {
         $Temp_h = Temp_h - 1$
         $SEQ_{i,k}$ = j }
```

Table 1 gives characteristics of systems which are simulatable by our approach.

Table 1. General Fast Simulation Assumptions

Category	Characteristic
System	Open job shop like queueing network
Servers at a Station	Single or Multiple
Server Failures	None allowed
Queue discipline	Any (additional time associated)
Buffer size	Infinite
Buffer size consistency	All infinite
Customer types allowed	No limit

The simulator does not model assembly nodes, material handlers, or probabilistic branching. Customer routings must be predetermined.

Control and Event Sets

The traditional discrete event calendar approach to problems of the nature described here is to place an event on the calendar for every arrival of a customer to the system and every departure of a customer from a server. This is an example of total control being exercised by the simulation controller, which in this case is the event calendar. Our approach is to reduce the amount of control exercised to only that which is necessary.

Simulation Control Issues

The total control used by the traditional simulation approach is necessary because no analysis of the system structure is done prior to simulation execution. However, when we decompose the system into nodes and levels we simultaneously define the dependency relationships of customers in the system. For acyclic systems (a customer visits the same server no more than once) a linear sequential dependency relationship exists between system levels. In this special case, we can completely simulate the flow of all customers in level 1, and then in level 2, and so on. This is essentially equivalent to q unique simulations tied together, where q is the number of levels in the system. For general systems, where cycling does occur, one event calendar is used to process events at all level boundaries. In both cases only events at the boundaries need be considered. In Figure 2 we show the points in an example system where control is exercised by General Fast in contrast with traditional simulation. We quantify the results of this savings in events in the next section.

Event Sets and Ratios

Since much of the time used by the traditional simulation approach is consumed in event processing we benefit from any reduction in the number of events placed on the calendar. We define the set of events required by a simulation approach to do a specific simulation as the *event set* for that simulation. For the restricted types of systems which we can simulate, we calculate the total number of events required by a traditional simulation approach as

$$m_{ts} = \sum_{j=1}^{J} \left(\frac{n_j}{b_j} + n_j M_j\right) \qquad (3)$$

where

J : total number of customer types

n_j : number of customers of type j simulated

b_j : lot size of customer arrivals of type j

M_j : total number of servers in the routing of customer type j

The expression for m_{ts} is divided into two parts. The first part is the events related to the arrival of batches. We generate one event for each batch arrival. Since n_j is the number of customers of type j that will be simulated and since we assume that the batches will be of equal size, the number of events associated with arrivals is

$$n_j / b_j.$$

The second part of the expression relates to departures from servers. Traditional simulation utilizes an event for each departure from a server. Therefore $n_j M_j$ gives us the number of departure events associated with customers of type j. We can see that the size of the event set is affected by the relative number of each customer type, the batch size of each customer type, and the length of their routing.

Similarly, the total number of events required by the General Fast Simulation approach may be expressed as

$$m_{fs} = \sum_{j=1}^{J} \left(\frac{n_j}{b_j} + n_j L_j\right) \qquad (4)$$

where

L_j : total number of system levels visited by customer type j

The expressions for event set size for General Fast and traditional simulation are very similar. The difference is that General Fast generates one event per level while traditional simulation generates one departure event per server. So $n_j L_j$ gives us the total number of departure events associated with customers of type j from each of the levels in their routing.

The ratio of the size of these event sets is the *event ratio*. The event ratio gives us some idea about the efficiency of General Fast relative to traditional simulation. The amount of computational effort involved with processing events in General Fast is proportional to the number of servers a customer visits in the corresponding level.

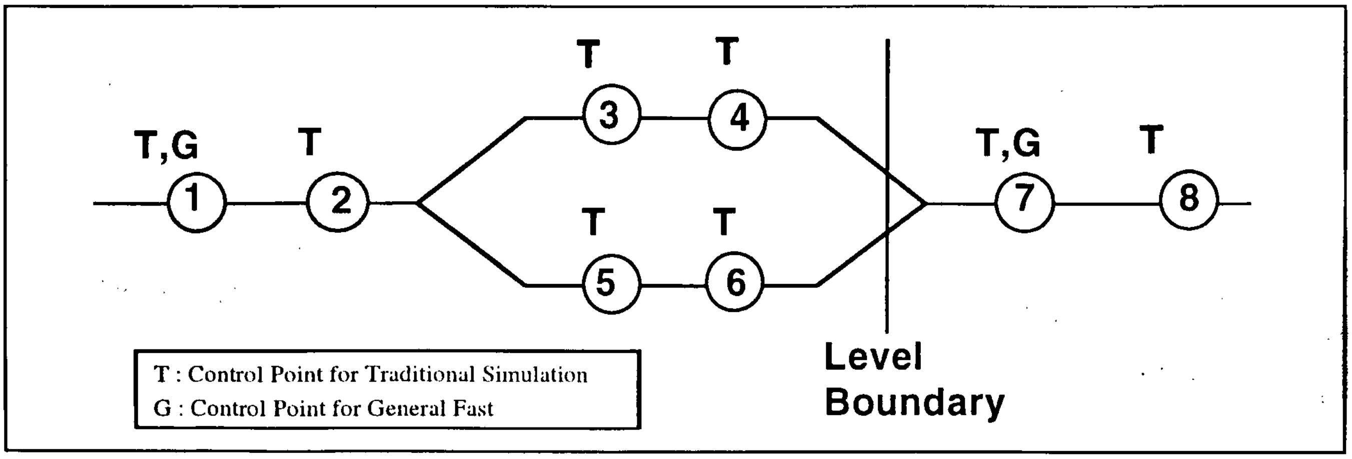

T,G
T
T
T
T
T
T,G
T
1
2
3
4
5
6
7
8
Level Boundary
T : Control Point for Traditional Simulation
G : Control Point for General Fast

Figure 2. Control Points in General Fast and Traditional Simulation

For this reason, the event ratio

$$\frac{m_{fs}}{m_{ts}} = \frac{\sum_{j=1}^{J}(\frac{n_j}{b_j}+n_jL_j)}{\sum_{j=1}^{J}(\frac{n_j}{b_j}+n_jM_j)} \quad (5)$$

is not an exact indicator of the actual execution speed of General Fast relative to traditional simulation, but it is proportional.

When we simplify the event ratio we notice an interesting result. We let

$$p_j = \frac{n_j}{n} \text{ where } n = \sum_{j=1}^{J} n_j .$$

That is, we represent n_j as some percentage p_j of the total customers simulated.

By doing so we show that the event ratio (or relative efficiency) is independent of the number of customers simulated. The efficiency of General Fast Simulation with respect to traditional simulation will be governed by the number of servers visited, the number of levels visited, and the relative percentages of the different customer types. (eq. 6)

Alternate Queue Disciplines

We have assumed prior to now that all queue disciplines are first-come-first-served. The first-come-first-served queue discipline is an exceptionally efficient case for our simulation methodology since the event calendar presents arrivals to a queue in first-come-first-served order. When we desire an alternate queue discipline we must not only sort by time (on the event calendar) but also by some other priority. The additional overhead associated with choosing an alternate queue discipline comes in the form of an extra sort, and an extra event which we must place on the calendar for each customer arrival to each level. Our previously defined event ratio becomes

$$\frac{m_{fs}}{m_{ts}} = \frac{1+\sum_{j=1}^{J}p_jL_j+\sum_{j=1}^{J}2p_jL_j^{AQD}}{1+\sum_{j=1}^{J}p_jM_j} \quad (7)$$

where L_j^{AQD} is the number of levels which customer j must visit with alternate queue disciplines. L_j is the number of remaining levels which customer j must visit (assumed to be FCFS). We can see that the efficiency of the simulator will decline with each additional queue that we choose to model alternately.

Implementation Results

The implementation of these concepts was tested using systems with a special structure. Figure 3 shows an example of this structure.

In all systems seven different customer types flow through the servers in three parallel lines. Three of the customer types flow directly down the three tandem lines. The four remaining customer types may follow the dashed routings indicated, or simply travel down the line on which they start. For any given system the three parallel lines always have identical length. The length of the lines and the point at which mixing stops were the variables of interest.

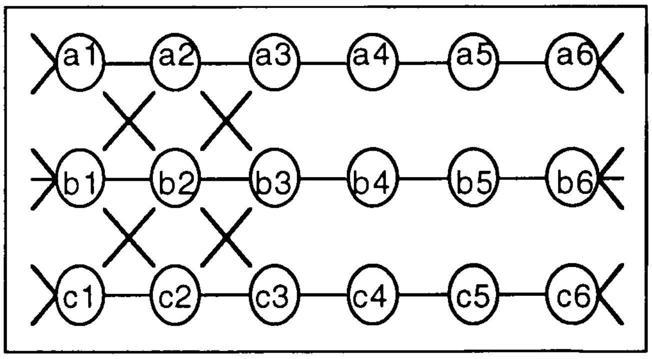


Figure 3. Sample Test System Representation

$$\frac{m_{fs}}{m_{ts}} = \frac{\sum_{j=1}^{J}(\frac{n_j}{b_j}+n_jL_j)}{\sum_{j=1}^{J}(\frac{n_j}{b_j}+n_jM_j)} = \frac{\sum_{j=1}^{J}(\frac{p_jn}{b_j}+p_jnL_j)}{\sum_{j=1}^{J}(\frac{p_jn}{b_j}+p_jnM_j)} = \frac{\sum_{j=1}^{J}(\frac{p_j}{b_j}+p_jL_j)}{\sum_{j=1}^{J}(\frac{p_j}{b_j}+p_jM_j)} \quad (6)$$

Systems tested had the following properties:

1. Equal numbers of each customer type were simulated $p_j = \frac{1}{7} \forall j$, J=1,7
2. Batch size was the same for every customer type (b_j=1).
3. In every system all customer types visit the same number of servers and levels (M and L respectively).

Dependency Index

When the event ratio is simplified we refer to it as the dependency index (DI).

$$DI = \frac{m_{fs}}{m_{ts}} = \frac{\sum_{j=1}^{7}\left(\frac{p_j}{b_j} + p_j L_j\right)}{\sum_{j=1}^{7}\left(\frac{p_j}{b_j} + p_j M_j\right)} = \frac{\frac{1}{7}\sum_{j=1}^{7}(1+L)}{\frac{1}{7}\sum_{j=1}^{7}(1+M)} = \frac{1+L}{1+M} \quad (8)$$

We can see that the dependency index is simply a ratio of the number of levels to the number of servers for our test case. We would expect that as the dependency index increases, the efficiency of General Fast simulation with respect to traditional simulation will decline. Figure 4 shows the results of our exploration.

As we increase the number of servers the execution time of every simulation increases. General Fast gives the fastest execution times when the dependency index is zero (the ideal case), as we would expect. Increasing the dependency index will cause slower execution times, but these times will still represent an improvement over traditional simulation up to a point. When we increase the dependency index to 1.0 the General Fast approach is slower than traditional simulation. In this situation every server is represented by a special tandem node which has more overhead than a simple traditional simulation server node. A fairly easy change to the implementation would be to represent tandem nodes of length 1 with a simple server instead of a tandem server node. Figure 5 shows a more detailed exploration of the effects of the dependency index for systems with a line length of 60.

With the present implementation General Fast simulation performs better than traditional simulation for systems with a dependency index less than approximately 0.7.

Software Interfaces

Implementation work was done in SmallTalk 80. Interfaces are provided for model specification, simulation parameter control and model execution, and graphical output analysis capabilities. In Figure 6 we show an example of the model specification interface.

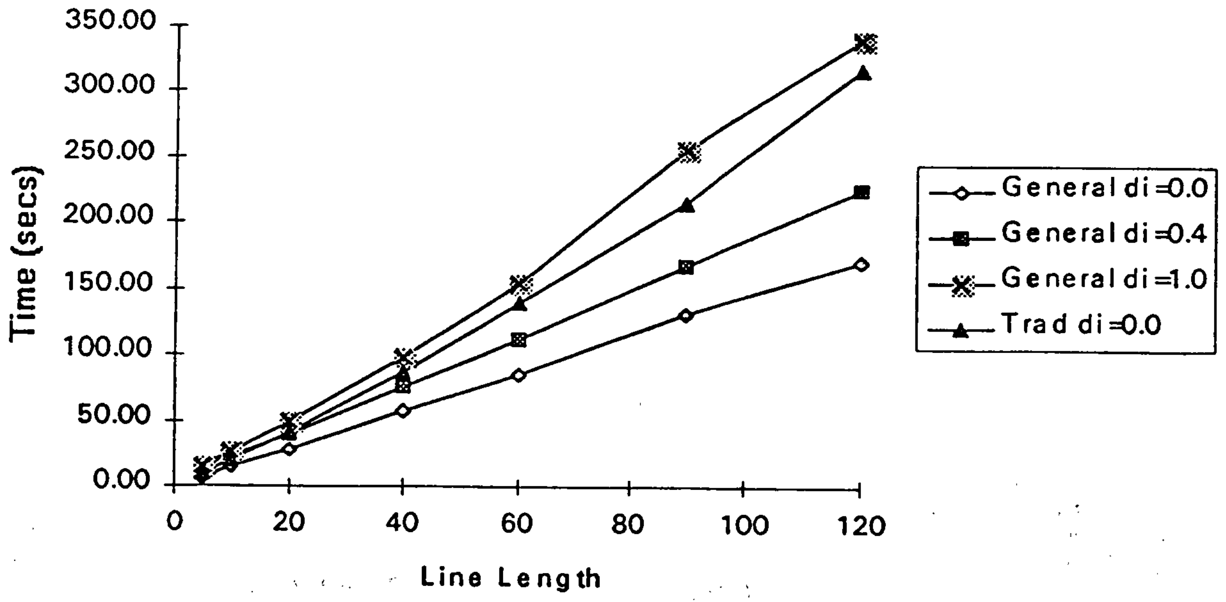


Figure 4. Execution Time Results

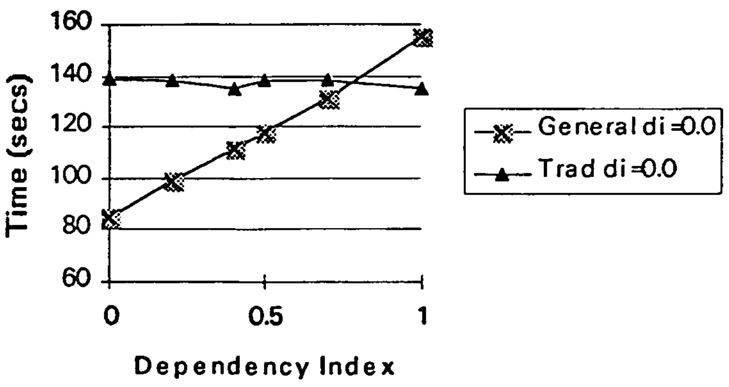
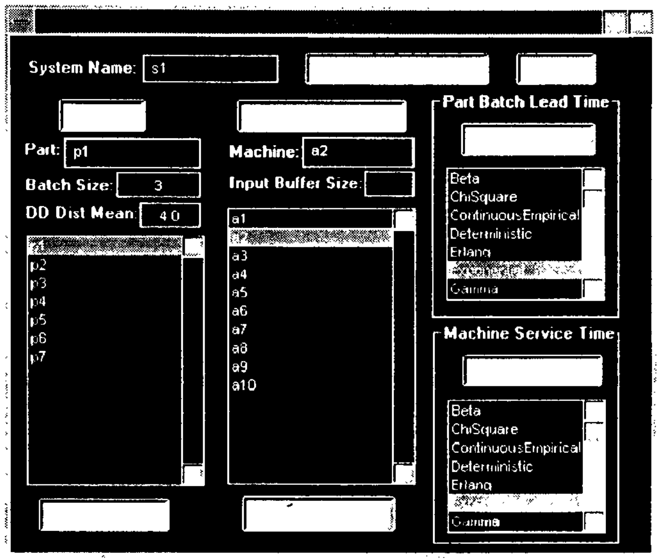


Figure 5. System with Line Length 60



System Name: s1

Part: p1

Batch Size: 3

DD Dist Mean: 40

Machine: a2

Input Buffer Size:

p2
p3
p4
p5
p6
p7

a1
a3
a4
a5
a6
a7
a8
a9
a10

Beta
ChiSquare
ContinuousEmpirical
Deterministic
Erlang
Gamma

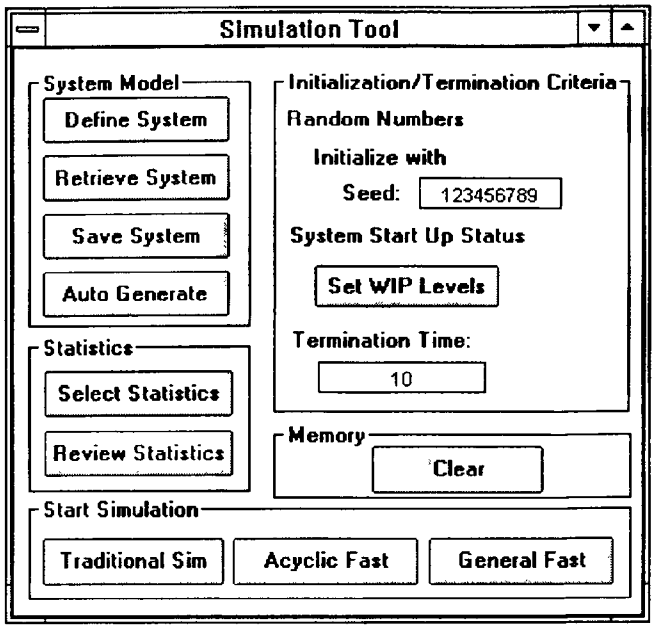
Figure 6. Model Specification Interface

The simulation is executed from the interface shown in Figure 7. In this interface the user may specify the seed and run length for the simulation, as well as selecting which servers to collect statistics for.

Figure 8 presents a sample output analysis interface that appears when a simulation execution is completed.

When the user clicks the Display Stats button in the interface in Figure 8 the window in Figure 9 appears.

The point estimate of the average number in system is shown as the horizontal line across the plot. If Statistics were selected in this window we would see that the value of the point estimate is 0.88 customers. A more detailed explanation of the output analysis interfaces is provided in Tretheway (1992). Similarly, Oltmanns (1992) describes the development of the object-oriented probability model and random number generation classes used in this implementation.

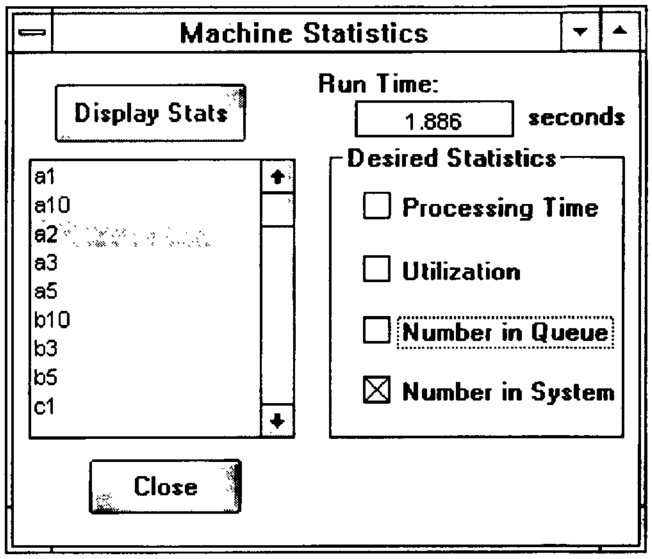


Initialize with Seed: 123456789

System Start Up Status

Termination Time: 10

Figure 7. Simulation Control Interface



Run Time: 1.886 seconds

a1
a10
a2
a3
a5
b10
b3
b5
c1

Desired Statistics

☐ Processing Time

☐ Utilization

☐ Number in Queue

☒ Number in System

Figure 8. Output Analysis Interface

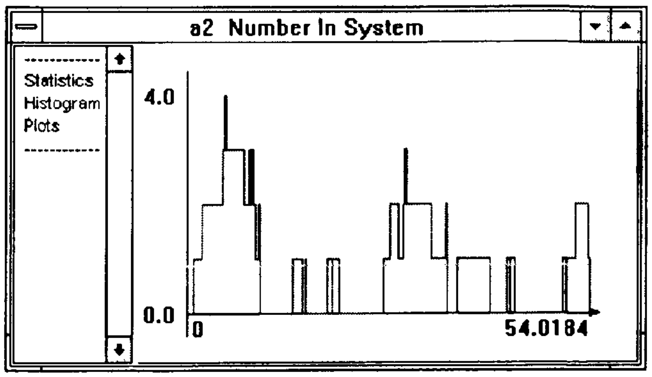


Figure 9. Number in System plot for Server a2

Conclusions

A methodology for greatly reducing the execution time for simulations of certain job shop-like queueing systems has been presented. The implementation, General Fast, demonstrates this reduction in execution times when compared with traditional simulations of the same systems. The concept of the system level and the use of recursive relationships between customer departure times in parts of the system allow us to reduce the time spent in event processing by reducing the number of events processed in a simulation. We have characterized the types of systems for which the methodology will yield the most improvement in execution time. We have demonstrated that though this approach requires analysis of the system prior to simulation, the technique is still much faster than traditional simulation and has a wide variety of applications.

Acknowledgments

This work is part of ongoing research at the University of Oklahoma both in Advanced Modeling Methodologies and in Integrated Production Management, and is funded in part by the National Science Foundation and the Oklahoma Center for Integrated Design and Manufacturing.

References

Chen, L., and Chen, C., (1990). A Fast Simulation Approach for Tandem Queueing Systems. *Proceedings of the 1990 Winter Simulation Conference*, Balci, O., Sadowski, R., Nance, R., Eds., IEEE, 890-905.

Chen, L., and Chen, C., (1993). A Fast Simulator for Tandem Queueing Systems. *Computers and Industrial Engineering* 24, 2, 267-280.

Kamath, M., and Bhuskute, H. (1991). Fast Simulation Techniques for Queueing Networks. *Working Paper CIM-WPS-91-MK1*, Center for Computer Integrated Design & Manufacturing, Oklahoma State University, Stillwater, Oklahoma.

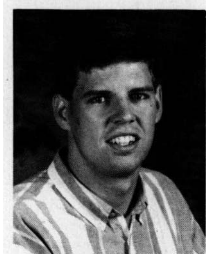
Law, A. and Kelton, D. (1991). *Simulation Modeling and Analysis*. Second Edition, McGraw-Hill.

Oltmanns, M., Tretheway, S., and Leemis, L. (1992). The Probability Model: A Construct for Simulation Input Modeling. *OU IE Internal Working Paper.*

Tretheway, S., Oltmanns, M., Leemis, L. and Hunt, C. (1992). A New Framework for Statistics Collection During Output Analysis of Computer Simulations. *OU IE Internal Working Paper.*

Walrand, J., (1988). *An Introduction to Queueing Networks*. First Edition, Prentice-Hall Inc.

Whit, W. (1983). The Queueing Network Analyzer. *The Bell System Technical Journal* 62, 9, November, 2779-2815.



CORTNEY S. HUNT received his BS and MS degrees in Industrial Engineering from the University of Oklahoma, Norman in 1992 and 1994 respectively. He is a member of the ORSA/TIMS, IIE, APM, and SME chapters. His research interests include advanced modeling methodologies, production management, and facility layout.

BOBBIE L. FOOTE, P. E., is a professor at the School of Industrial Engineering at the University of Oklahoma, Norman. He teaches production planning and control, quality control, statistics, engineering economy, and sequencing and scheduling. He is a member of IIE, ORSA, TIMS and OSPE. Professor Foote received honorable mention from the 1988 Franz Edelman Award for Management Science Committee. His current research interests include automated process planning, production planning, quality control, and facility design. Dr. Foote was named a fellow of IIE in 1991. He is the Associate Director of Oklahoma Center for Integrated Design and Manufacturing located at Oklahoma State University in Stillwater.