FAST SIMULATION OF GENERAL MANUFACTURING NETWORKS

By

RAM SREENIVASAN

Bachelor of Engineering

Annamalai University

Tamil Nadu, India

1993

Submitted to the Faculty of the
Graduate College of the
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
May, 1996

FAST SIMULATION OF GENERAL MANUFACTURING NETWORKS

Thesis Approved:

Manjunath Kamath

Thesis Advisor

David B. Pratt

T. J. Theel

Thomas C. Collins

Dean of Graduate College

ACKNOWLEDGMENT

I wish to express my heartfelt appreciation and gratitude to my advisor, Dr. Manjunath Kamath, without whose supervision, constructive guidance, inspiration, tolerance, and a certain facility to use the carrot instead of the stick this work would not have been possible. While I am essentially groping for words, and any superlatives to describe the interest he has shown in my well-being and progress (academically and otherwise) would still be an understatement, I can definitely say he has been a *teacher* in the truest sense of the term.

My sincere appreciation extends to my other committee members Dr. David Pratt, who taught simulation and the spirit of simulation to me, and Dr. Timothy Greene, who has always been effusively encouraging. I would like to thank the School of Industrial Engineering & Management for providing me with precious research and teaching opportunities and generous financial support. I would like to thank the rest of the CIM Center team for their support. I would like to, especially, acknowledge my friends Ralph and Shankar for their extensive guidance in programming and Baskar for his help in SLAM system.

I would like to thank my brother-in-law, Aththim, who pushed me into all *this*. Thanks also go to my mother and sisters for their support and encouragement. Special thanks are in order for my friend Amith and his family, comrade Thaths, and Pals who were always there when I so often got myself into trouble.

I wish to remember my father who would have loved to see me move in the right direction and my niece, Lajo, and nephew, Ashwath whose impishness has always been a source of joy and has vested in me a certain amount of responsibility.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION....1

Simulation of Systems....1
Motivation Behind this Research....4
Overview of Research....7

II. BACKGROUND OF THE STUDY....9

Review of Previous Work....9
Statistics Collection and Execution Efficiency....15
Fast Simulation - Unanswered Questions....16

III. STATEMENT OF THE RESEARCH....18

Research Goal....18
Research Objectives....19
Research Scope and Limitations....19
Research Contributions....20

IV. PRELIMINARIES: PERFORMANCE MEASURES, IMPLEMENTATION LANGUAGE, AND PHASES OF RESEARCH....21

Performance Measures....21
Selection of Modeling and Simulation Environment....22
Research Phases....23

V. DEVELOPMENT OF THE NEW METHODOLOGY....25

The New Methodology....26
Description of the Experimental Prototype....31

VI. FAST SIMULATION - IMPLEMENTATION....33

Design of Reusable Fast Simulation Building Blocks....33
Executional Details of the Fast Simulation Methodology Proposed....37
Memory Management of the Fast Simulator....40
Verification of Fast Simulation Logic....44

A Discussion on Linearity of Execution Time of Fast Simulation.....46

VII. RESEARCH SUMMARY, CONTRIBUTIONS, AND FUTURE RESEARCH.....51

Research Summary.....51
Research Contributions.....53
Future Research.....55

BIBLIOGRAPHY.....56

APPENDICES

APPENDIX I--ALGORITHMS FOR SIMULATING INDIVIDUAL TOPOLOGIES.....59

APPENDIX II--SOURCE CODE FOR SIMULATING EXPERIMENTAL PROTOTYPE.....80

LIST OF TABLES

Table Page

1. System Parameters for the Experimental Prototype.....45

2. Verification of Simulation End time for Prototype System.....45

LIST OF FIGURES

Figure Page

1. The relationship between departure times in a system with finite buffer.....13

2.1. A Simple Prototype.....27

2.2. An Alternate Prototype.....27

3. Fast Simulation Algorithm to Simulate Prototype System in Figure 2.1.....30

4. The Experimental Prototype.....32

5.1. Psuedocode showing modules that simulate system in Figure 2.1.....36

5.2. Psuedocode showing modules that simulate system in Figure 2.2.....36

6.1. Increase in Execution Time with Increase in Size of a Tandem Line.....50

6.2. Increase in Execution Time with Increase in Size of Assembly.....50

6.3. Increase in Execution Time with Increase in # of Merging Lines.....50

6.4. Increase in Execution Time with Increase in # of Parallel Servers.....50

CHAPTER I

INTRODUCTION

Simulation of Systems

Performance evaluation of any system is vital for its design and operation. Simulation, queueing networks, Markov chains and Petri nets are the most common techniques for performance evaluation of manufacturing systems (Viswanadham and Narahari 1992). However, for the detailed modeling and analysis of complex systems, simulation remains the most commonly used tool. As Kelton (1994) puts it, "As a general approach to addressing analytically intractable problems, simulation has always had an attractive directness and simplicity about it." He further adds, "While the general idea of simulation is popular and appealing, it has had its drawbacks, and thus its detractors. Perhaps the most obvious limitation is the need to keep track of and manipulate a lot of numbers as a simulation progresses."

Among the three fundamental simulation worldviews, namely, discrete event scheduling, process interaction, and activity scanning, the discrete event scheduling (DES) approach seems to be the most widely used, due to its execution efficiency and applicability to systems in general (Nance 1971). Another equivalent approach would be the 3-phase approach which takes a global view of the simulation model (Paul 1991); the time is advanced until there is a state change in the system or until something happens. At this point the system is examined to find out all the events that take place at this time, i.e. all the activity completions that take place at this time. Only when all resources due to be released at this time have been released, is the reallocation of these resources into

new activities started in the third phase of the simulation. The first phase is time advance. The second phase is to release those resources scheduled to end their activities at this time. The third phase is to start activities given the global picture about resource availability. The attraction of this method is that it gives maximum control of the model, the experimental tool for simulation, to the analyst. Decisions as to priority over resource allocation are more readily made within this structure. Since the third phase, the allocation of resources to new activities, is distinct from the rest of the modeling structure, some very esoteric allocation rules can be encapsulated in such a structure. The disadvantage of this method is that it can be computationally inefficient to run (Paul 1991).

In the DES approach, the execution time is primarily made up of the time consumed in manipulation of a list of scheduled future events. The implication of the data structure used for storing/removing events from the calendar on the execution efficiency of simulation has been recognized by several researchers. A summary of comparative performance of various data structures and algorithms has been reported in Adam and Dogramaci (1979). Reeves (1984) has studied the performance of various algorithms under certain conditions and has found *ternary heaps* to be more attractive for event manipulation.

When using simulation for systems design and analysis, DES does remain a powerful tool. However, for quite sometime now, researchers have been exploring real time control of systems using simulation (Harmonosky 1990). Even when real time control is not an objective, there is some need for continuously trying to speed up simulation, for, as Kelton (1994) says-

"Despite all this impressive technology advancement, though, simulationists continue to push the envelope by continuing to ask more of their simulations- run them longer, replicate them more, look at more scenarios, allow for more experimental factors, and search for input-parameter combinations that optimize a performance measure."

He further continues,

"Be that as it may, there remain at least two additional barriers, of a more fundamental nature, to the continuing advance of simulation's utility, neither of which can ever really be "solved" by faster, bigger, cheaper hardware.

- General methodological problems concerning how to model, how to plan a course of simulation experimentation, and how to interpret the results. Research in these areas is quite active.
- As such methodological advances occur, they must be effectively embedded in simulation *software* to make them available to the wider world of applications in a form that will gain them acceptance and routine use. This will involve closer collaboration between methodological research and simulation-software developers than has been the case so far."

So saying, he "argues for the vitality of simulation, thus justifying investment in methodological research and its implementation in software."

In the same vein, we can see, any methodological course to improving the execution speed of simulation should be welcome. Proceeding in this train of thought, we can theorize that if the execution time of DES can be reduced by reducing the manipulation of the event list, or by total elimination of the event list itself, then we would have brought the dream of using simulation for real time control one step closer to reality. This is exactly what fast simulation attempts to achieve. Chen and Chen (1993) remark, "We observe that, when the event-scheduling technique is employed, complex data structures (e.g. pointers and linked lists) are generally used, and much simulation run time is devoted to the management of complex procedures (e.g. search, sort, link, and unlink). In order to save simulation run time, if possible, we want to avoid using these structures and procedures." Furthermore, the simpler fast simulation models may

provide us some insight into how some systems may be simulated with more ease than when DES is employed.

The field of fast simulation traces its origin to Chen and Chen's (1990) seminal paper, in which they provide a methodology for simulating a simple tandem line using recursive relationships that are established between departure times of customers (which is what this type of fast simulation is all about) thereby precluding the overhead of event manipulation. They followed that publication with a paper which had a better implementation of the same tandem line simulation (Chen and Chen 1993). Since then Duse (1994) has come up with recursive relationships to fast simulate other manufacturing topologies like merge, split, assembly, parallel server workstation and unreliable server.

Motivation Behind this Research

Continuing the discussion of fast simulation, one very soon comes to think of how to model the dynamics of systems to enhance the generality (like ability to handle larger manufacturing networks and non-FCFS queue disciplines) of the approach. The current approaches deal with the modeling issues by imposing restrictions on the system features; or by using a hybrid approach. This brings us to the question of what the available fast simulation approaches are, and what the other related approaches are.

Methodologies that attempt to reduce execution time of simulation of general systems through methodological changes in simulation mechanism can be classified as follows.

1. Acyclic Fast (Hunt 1994, Hunt and Foote 1995)

Here the event calendar is completely avoided. Hence this methodology may be classified as 'pure' fast simulation. Customers being simulated visit any node in the system at most once (hence acyclic). The system is decomposed into nodes and levels. 'Nodes' are typically tandem line structures which can be fast simulated by Chen and Chen's procedure (1990) of recursion. A 'level' is a division in a system within which dependency relationships of customer departure times are obvious; like in the case of a single tandem line. We can then exploit the linear sequential dependency relationship that exists between levels in a system. This implies that we can completely simulate the flow of all customers in level 1, and then in level 2, and so on. Since all customers have to be simulated through one level before they can enter another level, this methodology entails a number of shortcomings, viz.,

- The departure times of all the customers from a system level taken together have to be stored in memory at some point of time. This places a lot of demand on the memory and puts a limit on the number of customers that can be simulated;
- As the name indicates, only acyclic situations can be modeled;
- Since there is no way to conglomerate departure times of customers, assembly servers could not be included;
- Buffers had to be infinite;
- Parallel servers were not modeled; and
- Only systems which can be modeled as combination of tandem lines can be simulated.

2. General Fast (Hunt 1994)

This does not completely avoid the event list. Here, control points (typically, points where tandem lines merge or split into other tandem lines) are identified for a network. Only events happening at these points are entered in the event list. In effect, we breakdown the event list necessary to maintain the logical control of the simulation into a smaller list containing events happening at points identified as control points. What this does is, make the control points see smaller lists for manipulation (search and insert). This results in lesser time being spent in event list searching, and hence in simulation execution. Though this has been called fast simulation methodology by Hunt (1994), we have to understand the basic difference between this approach and 'pure' fast simulation where the event list is avoided completely. Here, the reduction in execution time is achieved by fast simulating any system component (like tandem line) which can be fast simulated. This way the number of events that we have to take care of is reduced. In this sense, general fast is more like hybrid simulation which is discussed next. However, general fast has a few differences from hybrid simulation in the way it has been applied so far. It, like acyclic fast, has been applied only to systems which could be treated as combination of tandem lines, and infinite buffer cases. It showed some promise in dealing with non-FCFS queue disciplines.

3. Hybrid Simulation (Duse 1994)

One other technique that one should be sensitive to while talking of fast simulation is one called hybrid simulation. It was theorized that for a fairly complex manufacturing system, fast simulation may not really be an efficient mechanism for simulation. In some cases, pure fast simulation may not even be possible. Hence

2. General Fast (Hunt 1994)

This does not completely avoid the event list. Here, control points (typically, points where tandem lines merge or split into other tandem lines) are identified for a network. Only events happening at these points are entered in the event list. In effect, we breakdown the event list necessary to maintain the logical control of the simulation into a smaller list containing events happening at points identified as control points. What this does is, make the control points see smaller lists for manipulation (search and insert). This results in lesser time being spent in event list searching, and hence in simulation execution. Though this has been called fast simulation methodology by Hunt (1994), we have to understand the basic difference between this approach and 'pure' fast simulation where the event list is avoided completely. Here, the reduction in execution time is achieved by fast simulating any system component (like tandem line) which can be fast simulated. This way the number of events that we have to take care of is reduced. In this sense, general fast is more like hybrid simulation which is discussed next. However, general fast has a few differences from hybrid simulation in the way it has been applied so far. It, like acyclic fast, has been applied only to systems which could be treated as combination of tandem lines, and infinite buffer cases. It showed some promise in dealing with non-FCFS queue disciplines.

3. Hybrid Simulation (Duse 1994)

One other technique that one should be sensitive to while talking of fast simulation is one called hybrid simulation. It was theorized that for a fairly complex manufacturing system, fast simulation may not really be an efficient mechanism for simulation. In some cases, pure fast simulation may not even be possible. Hence

research was done to study issues involved in effectively combining fast simulation and DES, and this technique came to be called hybrid simulation or multi-mode simulation (Duse 1994). While the basic idea of general fast and hybrid simulation is the same, certain differences in explaining away the scope of each have been due to their contemporaneous development.

Reiterating what we have gone through so far, all previous research has focused on finding individual manufacturing network topologies (e.g. tandem queueing system, single server assembly station, and merge node topology) that can be fast simulated. Simultaneously dealing with the various topologies in a fairly complex system and fast simulating the same was beyond the scope of previous research efforts. One previous research effort did concentrate on the combination of tandem lines (to form a prototype job shop) (Hunt 1994). However, its scope was limited as discussed earlier.

The motivation behind this research is this recognition of an opportunity to develop a robust approach to model a general manufacturing system that is a combination of various individual network topologies, for purposes of simulating the same using pure (has no event list whatsoever) fast simulation.

Overview of the Research

The rest of this thesis is laid out in the following chapters. The literature relevant to this research is reviewed in Chapter II. This chapter also contains the research questions that remain unanswered in this area. In Chapter III, the scope and limitations of this study are defined by presenting a concise statement of the goal for this research. The contributions this effort would provide to the body of knowledge are also outlined in

this chapter. Chapter IV discusses the performance measures to be used and various phases of this research. Chapter V discusses the proposed approach (in the context of an experimental prototype) to solve the stated research problem. Chapter VI discusses the implementational details of the methodology. Chapter VII gives a summary of the research followed by contributions to the body of knowledge and future research possibilities.

Appendix I gives detailed algorithms for some typical topologies that have been previously dealt with by researchers. It has been provided to give the reader a flavor of how implementations of fast simulation models of typical network topologies may be made. Appendix II gives the source code for fast simulation of an experimental prototype.

CHAPTER II

BACKGROUND OF THE STUDY

Review of Previous Work

In their seminal work in fast simulation, Chen and Chen (1990) identified how single server-infinite/finite buffer stations in tandem can be simulated based on recursive relationships that can be captured between the departure times of the customers from the successive stations. Basically, they exploited the simplicity of a tandem line to form those recursive relationships and performed the simulation with no event list whatsoever. In the process they realized speed-ups of up to 80% in their run time while estimating certain performance measures. We cannot avoid the generation of random numbers and random variates, and have to generate them as we do in DES.

In formulating their fast simulation recursions, Chen and Chen (1990) had assumed reliable servers, First Come, First Served (FCFS) queue discipline, and single class of customers. In other complex cases, say with state-based decision making one may have to look into how the system can be modeled for purposes of hybrid simulation, and which parts of the system are to be modeled for fast simulation and which parts are to be modeled for DES (Duse 1994). The aim of hybrid simulation is to reduce the number of events in the event list and thus reduce the simulation execution time. Also, this effort may not be worthwhile in reducing the execution time for some complex systems where 'many' portions of the system have to be modeled using DES.

Duse (1994), in his work, extended the body of knowledge in fast simulation by simulating assembly, merge, split, parallel server, and unreliable station topologies. An assembly server has a certain number of components coming into component buffers. Once one of each type of component is available for the assembly server to work on, the components are assembled in the assembly server. A merge node, on the other hand, takes in customers to be serviced from different merging lines. The customer which has finished the earliest in the last workstation of the various merging lines waits at the buffer at the merge node or is served at the merge node. In the case of a split topology, a tandem line splits into two or more tandem lines. A parallel server workstation has number of servers in parallel, with a single common buffer in front of them.

Some sample fast simulation algorithms - for a tandem line with finite buffer, an assembly station with infinite buffer, an assembly station with finite buffer and merge topologies - given by Chen and Chen (1993) and Duse (1994) are included in Appendix I. Also included is the algorithm for the simulation of the parallel server topology, based on Duse's (1994) discussion. Though these algorithms of Chen and Chen, and Duse have been presented in detail in the appendix, we present next a brief explanation of the underlying rationale of fast simulation as applied to the classic case of a tandem line.

The following relations were identified for a single server tandem line by Chen and Chen (1993). The, relations carry the assumption of a reliable server and FCFS discipline, and a single customer class.

Let

$D_{i,j}$ = departure time of the j^{th} customer from the i^{th} station

$S_{i,j}$ = service time of the j^{th} customer at the i^{th} station

$E_{i,j}$ = service end time of the j^{th} customer at the i^{th} station

Infinite buffer case

The service start time of the j^{th} customer at the i^{th} station

= max (departure time of customer j at station i-1, departure time of customer j-1 at station i)

= max ($D_{i-1,j}$, $D_{i,j-1}$)

The service end time, $E_{i,j}$ = max ($D_{i-1,j}$, $D_{i,j-1}$) + $S_{i,j}$

In the infinite buffer case, the service end time ($E_{i,j}$) is also the departure time ($D_{i,j}$). However, when we have finite buffers, the service end time need not be the same as the departure time and the departure time may be computed as in Chen and Chen (1990) where they have adopted a "customer-by-customer" view. What this means is that they take a customer and simulate that customer's flow along the tandem line. Once that customer is completely simulated through the tandem line, the next customer is taken for simulation.

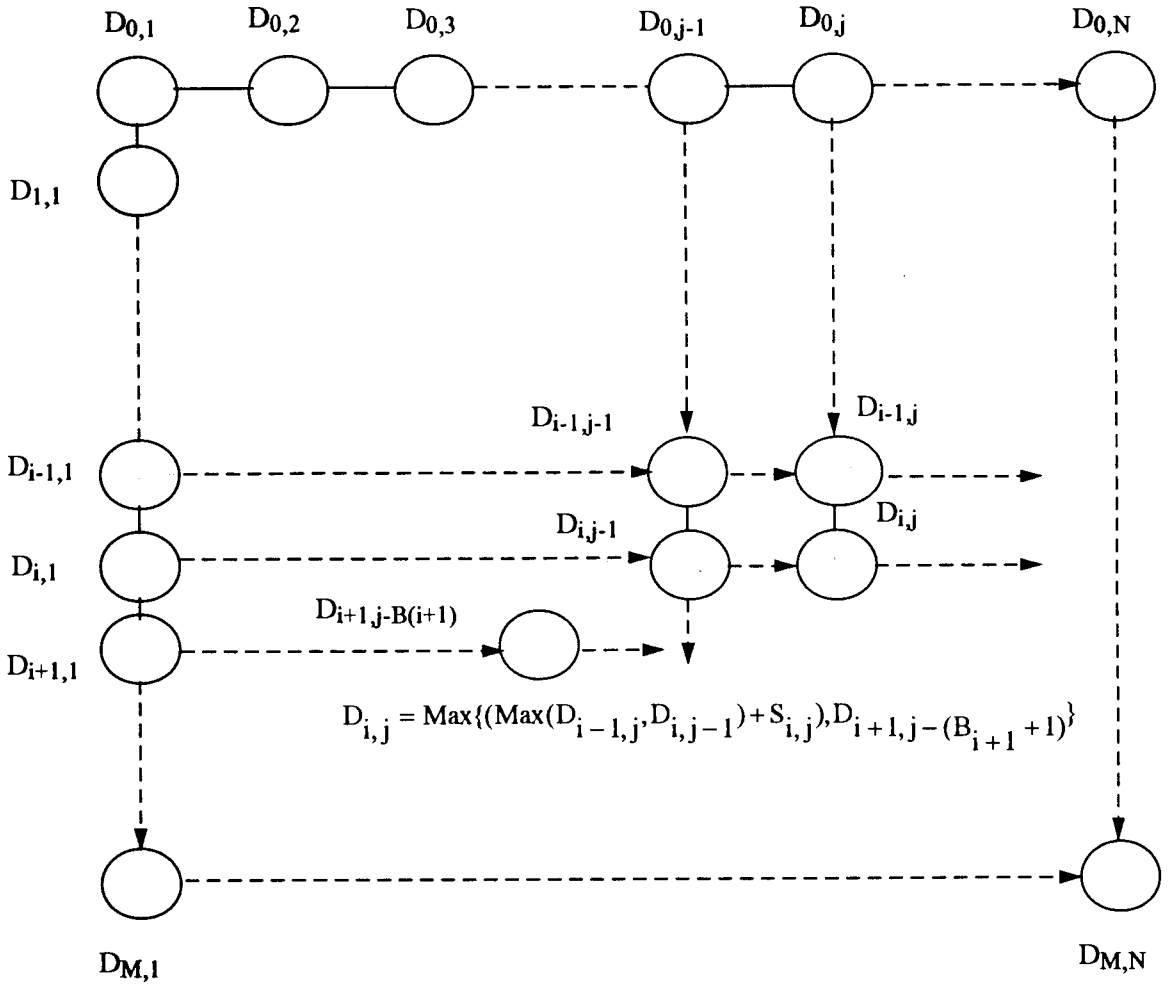
Another way of putting this is when the n^{th} customer is being simulated, all previous n-1 customers have been simulated and quantities associated with them needed for statistics collection have been recorded. Let us take the n^{th} customer. That customer, after finishing service in station 1, will be blocked if there is no space available in the input buffer of station 2. Let us now say the input buffer of station 2 has a capacity B = 3 (excluding the one space in the station). It is now clear that there will be space available in the input buffer of station 2, only when the $n-4^{th}$ customer departs from that station. We may choose to call the $n-4^{th}$ customer the "preceding buffer-relevant" customer and the n^{th} customer the "later buffer-relevant" customer. The term "buffer-relevant" is used to denote the fact that they are connected through their departure times'

dependence. Thus, to determine the departure time of customers from station 1, we need to know the departure times of preceding buffer-relevant customers from the following node. This is the basic idea in modeling any finite buffer case, be it single server or parallel server station, assembly, or merge or any other topology. Use of the "customer-by-customer" view gives us the ability to determine departure time for the n-4th customer from station 2 before we have to determine the departure time of nth customer from station 1.

Summing up the above concept in equation form

$$D_{i,j} = \max\{E_{i,j}, D_{i+1,j-(B_{i+1}+1)}\}$$

where B_{i+1} is the buffer capacity of $i+1^{th}$ station excluding the one in the server. The above explanation is picturised in Figure 1.



M = The total number of stations in the tandem line

N = The total number of customers to be simulated through the tandem line

Figure 1. The relationship between customer departure times in a tandem line with finite buffers.

Adapted from Chen and Chen (1990)

However, when we have the sequence of customers changing at any station, the customer-by-customer approach will not work as it is. We need to re-index customers. This is what we attempt to do in topologies like merge and parallel station.

- The merge node is simulated using a switching mechanism to toggle between the various merging lines. The service end times at the end nodes of all merging lines are compared to get the lowest service end time among them. The line corresponding to this service end time would be the present merging line, and its present customer is processed through the merge node. Even as the merge node is being simulated, the next customer at the previous merging line can be simulated. Its service end time at the last node of that line is obtained for comparison with service end times of other lines. This way the simulation is kept going, by getting the forthcoming merging line every time the present merging line sends a customer to the merge node.
- The parallel server workstation case creates a tricky situation where customer indices are not retained as in the previous case. To handle this situation Duse (1994) used the abstraction of 'customer-by-customer-with-switching', where customers are taken into a parallel set of servers and their service end times are obtained. Then the customer with the least service end time is sent out of the parallel server workstation as the next departing customer (though it may not have been the first customer to have come to the workstation). The next customer is brought into the depleted server in the parallel server workstation and its service end time is obtained and the procedure is continued.
- For fast simulating an assembly server, Duse (1994) uses the abstraction of queue removal time. As components come into the component buffers their arrival times

are recorded and the maximum of the component arrival times is taken as the kit formation time. Once we have the kit formation time, maximum of the kit formation time and the previous departure time from the assembly server yields the next assembly start time. Addition of the assembly service time to this assembly start time gives the new assembly service end time. This assembly end time is the departure time and is used to update the component buffer departure times.

- The only pure fast simulation work available that goes into a methodology for simulation of larger systems is Hunt's acyclic fast (Hunt 1994, Hunt and Foote 1995), that has been dealt with previously in Chapter I.

Statistics Collection and Execution Efficiency

Fast simulation does not involve approximations and hence should give results identical to those of DES. Certain statistics like average queueing times can be easily collected using fast simulation. However other state-based statistical measures like queue length distribution require some extra computations; as state-based decision making is not natural to the outlook taken by fast simulation. It then becomes a question of where to draw the line in the range of statistics so as to not go overboard on the execution time. One may use to one's advantage analytical results, such as Little's law (Glynn and Whitt 1989), that queueing theory has given to us, to eliminate some statistics collection procedures, and the associated time. Also, calculation of the average queue lengths in this indirect way is in many ways better than traditional computation of queue lengths (Glynn and Whitt 1989). However, it should be noted that the savings in the execution time in fast simulation is not due to the reduction in the type of statistics

independently collected. It is due to the essential mechanism that drives this type of simulation; namely, the absence of overhead related to search and insertion in the event list. Instead, we only have simple comparison and addition operations that steer the recursive relationships that capture the system dynamics.

It is to be noted that there are numerous other techniques that attempt to reduce the computational time of a simulation run. For example, the parallel/distributed simulation techniques use hardware improvements to reduce the computational time (Bhuskute 1993). Variance reduction techniques and regenerative simulation method (Bratley et al. 1987) reduce the run time by reducing the total run length needed to produce estimates with a desired statistical accuracy. Typically, they borrow ideas from concepts in probability theory, stochastic processes and statistics. On the other hand, fast simulation draws its execution efficiency from modeling the dynamics of the system using a methodology (namely, the formulation of recursive logical relationships to model the system element interactions) fundamentally different from the event-oriented approach.

Fast Simulation - Unanswered Questions

The following are some of the questions generated from previous research efforts (mostly Duse's (1994) research).

1. Is the world-view 'customer-by-customer-with-switching' (Duse 1994) general enough to handle all topologies and combinations of them? If not, what modifications to the above view may succeed in handling a combination of these topologies?

2. Presently, the statistics one collects in fast simulation are limited to those that can be collected without significantly losing their execution efficiency. What other statistics (like queue length distributions) can be collected without adversely affecting the computational efficiency?
3. What would be the issues involved in developing trade-offs between loss of accuracy and additional savings in execution time?
4. How can we predict speed-ups in fast simulation? Hunt (1994) has devised ways and means for predicting speed-ups in the context of general fast.
5. Ideally, only for individual topologies the execution time of fast simulation should increase linearly with the increase in system size. Even so, what is the functional relationship between the system size increase and the increase in execution time.
6. What are the issues involved in the judicious integration of various approaches such as parallel discrete event simulation (PDES), fast simulation, metamodeling, etc. to gain the maximum possible computational efficiency?

One may stress at this point that the focus of this research is to answer the question- *"If one develops fast simulation models of typical manufacturing network building blocks, then can these models be integrated to create a fast simulation model of a system which contains a logical combination of such network building blocks?"*

CHAPTER III

STATEMENT OF THE RESEARCH

As mentioned earlier, the previous research efforts in fast simulation did not deal with the combinations of various manufacturing topologies. They were successful in identifying possible world views for effectively handling the various topologies in isolation (Chen and Chen 1990, Duse 1994). Once this was accomplished Duse's (1994) research effort turned its attention to handling complex manufacturing scenarios with hybrid simulation. One other research (Hunt 1994) did not go into handling complex manufacturing systems containing the various manufacturing topologies investigated, but attempted to handle job shop like systems which can be considered as a combination of tandem lines. The same work simulated its prototype job shop of tandem lines using general fast- a fast simulation technique that retains the event list of DES. This work also explained the superiority of general fast when handling queue disciplines other than FCFS (especially, earliest due date), in job shop situations.

Research Goal

The goal of this research was to make a contribution to the evolving field of fast simulation by considering a general manufacturing system that is a combination of various network topologies (that in isolation have been modeled before) and handling the simulation of this system using pure fast simulation.

Research Objectives

The main objectives of this research are listed below.

1. To investigate issues related to the combination of various network topologies into a reasonable manufacturing system for the purpose of simulating the same using fast simulation.
2. To investigate the potential of a new "pulling customers into network when needed" approach.
3. To demonstrate the potential of the new approach using a prototype manufacturing system.
4. To investigate the functional relationship between the execution time of fast simulation and system size.
5. To disseminate knowledge about fast simulation in the research community.

Research Scope and Limitations

The scope of this work was limited to integrating manufacturing topologies for the purpose of fast simulating them. The prototype system consisted of only topologies which have already been investigated. One such prototype system is shown and discussed in Chapter V. Other than tandem lines, typically, topologies that Duse (1994) had studied, such as merge, assembly, and parallel server workstation were used. Other possible topologies, including batch nodes and split that may be fast simulated without much ado, were not studied since no interesting issues were identified to be highlighted using them. Another important clarification that can be made here is that this work did

not go into queue disciplines other than FCFS. No deliberate attempts at predicting speed-ups were made.

No attempt was made to delve into hybrid simulation. If in some rare cases, event listing (such as in general fast) was to have been used, it was considered. However, its usage was precluded, owing to the finding of a better 'pure' fast simulation methodology.

Research Contributions

The primary contribution of this research was the development of a fast simulation methodological framework that would be effective in simulating manufacturing networks that can be configured from common manufacturing topologies, and processing parts on a FCFS basis. This would help to fill the present gap in the body of knowledge about integration of various topologies while simulating them using fast simulation. While Duse (1994) had successfully identified topologies that can be fast simulated, he then went on to investigate hybrid simulation for simulation of a combination of the topologies. He had indicated the opportunity for future researchers to identify other topologies/combinations of them that can be fast simulated. That is what this research has taken up and achieved.

CHAPTER IV

PRELIMINARIES: PERFORMANCE MEASURES, IMPLEMENTATION LANGUAGE, AND PHASES OF RESEARCH

Performance Measures

Fast simulation performance measures can be broadly classified into the following two categories:

Quantitative Measures

Simulation execution time: This can be considered to be a direct measure of the execution efficiency of the simulation. Of course, the gain in execution time can be measured only in the context of hardware and language used. In the case of fast simulation, execution time is the primary performance measure (Duse 1994).

Average number of computational steps in the algorithms: As can be seen, these are factors which contribute to the speed-up. These factors are absolute measures, as they are independent of the hardware and language used. Relying heavily on these instead of on execution time, may not be an objective way of determining the proper performance measure. This is because sometimes the computations involved in fast simulation of complex systems may be more time consuming than in DES, though they may seem to involve less number of algorithmic steps.

Qualitative measures

Duse, in his work (1994), has outlined the following qualitative measures that can be used to judge a fast simulation model.

1. Difficulty/awkwardness of model generation.
2. Complexity of model management.
3. Ease of model modification for experimentation.

All the above measures may be used to evaluate the proposed methodology. However, since the aim of the research was not to go into aesthetics of model generation, no explicit discussions shall be included. We will use the execution time and the functional relationship it has to the system size as the performance measures to judge our fast simulation methodology.

Selection of Implementation Language

The seminal work in this area by Chen and Chen (1990 and 1993) used a general purpose programming language, namely C. However, the later works (Duse 1994, Hunt 1994) used an object oriented programming (OOP) language, SMALLTALK, (Kreutzer 1986, Goldberg and Robson 1989) for implementation. The usage of an OOP language may to some extent reduce the stress one may have to lay on statistics collection computations.

This research was implemented in C in Microsoft Visual C/C++ environment where statistics collection routines would be more clear on scrutiny. However, we may not expect the same degree of modeling reusability and flexibility as in a pure OOP implementation.

The tediousness of coding in a general purpose language is worth the effort, as very little knowledge is available regarding issues/problems one may be faced with while implementing fast simulation algorithms (other than that for tandem lines) in it.

Typically, the issues/problems are expected to be more pronounced in handling statistics collection mechanisms and the related data storing methods. Another point to be noted is that general purpose languages are much faster than OOP languages. Obviously, any effort to show the viability of general purpose languages for fast simulation would help researchers to exploit the speed-up a general purpose language provides over OOP languages.

Research Phases

Phase I

Implementation of each network topology in isolation. Identification of topologies which can be combined easily (e.g. we can couple assembly with feeder tandem lines).

Phase II

Constructing a fairly complex manufacturing system by combining at least one of each type of topology one has simulated in isolation and developing an approach to fast simulate the same.

Phase III

Verification of the fast simulation models using corresponding DES models. They should give the same results, as there are no approximations involved in fast simulation. This effort in verification was also accompanied by performance evaluation of the results. By performance evaluation we mean, comparing the execution times of a fast simulation model and the corresponding DES model. Because the fast simulation was handled so effectively as to touch the ideal case of preserving the linear growth in execution time with increase in the 'size' of any particular topology, DES model building

in C was dispensed with. The verification of the output was done with a DES model in a simulation language, SLAM II (Pritsker 1986).

CHAPTER V

DEVELOPMENT OF THE NEW METHODOLOGY

As we began to consider the development of a new methodology to fast simulate systems that have combinations of various topologies as its building blocks, the following issues surfaced. (A building block, is typically, a manufacturing network topology that may constitute a logical component of a larger manufacturing system.)

1. At every building block we have to know what the next building blocks are, so as to decide what to do with a customer as it leaves the present building block.
2. We have to know the buffer limitations in the next building block; only taking into consideration the buffer limitations can anything be done with a customer once it has finished service at the present building block.
3. Not all servers have just one buffer associated with them. For example, a single assembly server has as many buffers associated with it as there are components. In the same way a parallel server workstation has many servers but with only one buffer associated with those servers.
4. We also have to think of how the dynamics of the system is to be taken care of in its entirety.
5. There are other executional details that have to be addressed - such as, how statistics are to be collected. If statistics are not collected at the right time and the values so collected deleted, we may run into serious memory problems.

6. We should have a generic way of capturing a system as input to the fast simulation program.

The New Methodology

With the above issues in mind, let us discuss the new methodology and the implications it will have in the execution of a simulation program based on the methodology. Consider a simple system with just two building blocks- an assembly server and a merge node combined together as shown in Figure 2.1. For convenience sake, let the number of components feeding into the assembly server be two, which also means the number of buffers before the assembly server is two. Let the merge node accept customers from two streams, one coming from the assembly server and the other coming in from the lone workstation as shown in Figure 2.1.

Let the components being assembled be c_1 and c_2. Once they are assembled, let the unit be called a_1. The components leaving the lone workstation enter the merge node and are called c_3. Now we can simulate this simple system as follows.

We generate an arrival time for c_1 and compare it with the departure time of the previous buffer-relevant customer. As defined in Chapter II, a previous buffer-relevant customer is a customer which constrains the departure of the present customer from a workstaion into the buffer of the next workstation. If the arrival time is less than the departure time of the preceding buffer-relevant customer then the customer is lost, c_1 lost count is incremented and another arrival time is generated. If this arrival time is more than the departure time of the preceding buffer-relevant customer, then this arrival time is updated as the component arrival time for c_1.

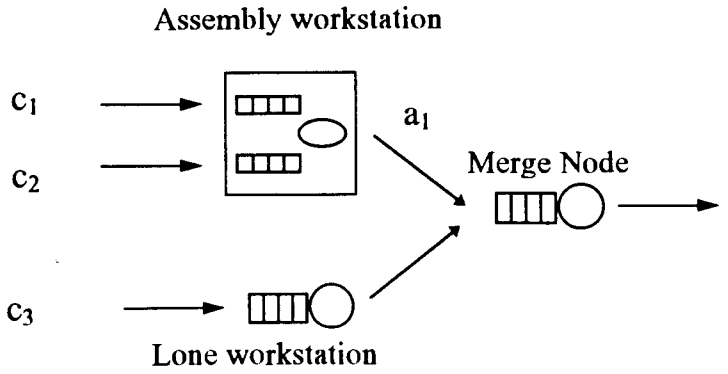


Figure 2.1. A Simple Prototype

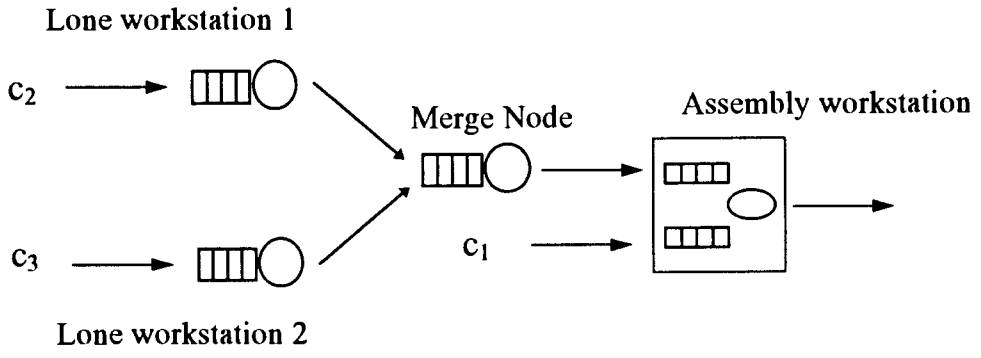


Figure 2.2. An Alternate Prototype

In the same way, arrival time for c_2 is generated. The maximum of the arrival times of these two components is taken as the kit formation time or the earliest start time. The maximum of this kit time and the departure time of the previous assembled component is the next assembly service start time. Adding the assembly service time to this start time gives the next departure time. This departure time is used to update the departure time arrays which hold the departure times of the preceding buffer-relevant customers. The departure times of those preceding buffer-relevant customers are held so that the forthcoming arrival times may be compared with those departure times to compute the loss count or arrival times.

However, the above approach will not work per se! For the following reason- we will not know about the availability of space in the next buffer (at the merge node), as by the time a single assembled component a_1 is ready to go into the merge node, many c_3's may have arrived and been waiting/processed in the merge node. We have to make some changes to the above logic of getting the departure times at the assembly server without any other consideration, if we have to maintain the simulation logic's fidelity to reality. A way of doing this would be to postpone the computation of the departure time at the assembly station until later when we have enough data to make the decision. So we just stop the assembly server simulation with the calculation of the assembly service end time.

We simulate the arrival of component c_3 to the lone workstation in merge stream 2. The service end time at that station is obtained and that is compared with the assembly service end time. The customer with the lesser service end time is taken into the merge node after comparison with the buffer availability time. The arrival time into the merge

node is the maximum of the service end time and the buffer availability time at the merge node. This buffer availability time itself is equal to the departure of the preceding buffer-relevant customer from the merge node. Let us say that the lone workstation has the lesser service end time; so the customer from this workstation is taken into the merge node and its departure time (this being equal to the service end time at the merge node) is generated. With the completion of this step, one customer has been simulated through the system. Now, the lone workstation has been depleted of a customer; so another arrival into the lone workstation is generated and the new service end time at that workstation is compared with the service end time at the assembly server and the customer with the lesser service end time is once again taken into the merge node. The above discussion of the methodology is captured in an algorithmic form in Figure 3.

The point to be stressed here is that, while we are trying to simulate systems as shown in Figure 2.1, we should not only be sensitive to topologies present, but should also be thinking about the order in which they have come to be present. For example in Figure 2.2, the merge node comes first, and customers from the merge node form one of the two components being assembled in an assembly server. The other assembly component comes from the lone workstation 2. In such a case we would keep simulating one customer out of the merge node, as we simulate one arrival of c_1, and then keep 'assembling' them together.

The rationale behind the proposed approach can be summed up as follows- "once we have knowledge of the fate of previous customers on forthcoming stations, we can decide the fate of customers at any station." This approach may be called "***pulling customers into the network as and when needed.***" The above approach, in some ways,

1. Accept one arrival of c_1 after comparison of its arrival time with the departure time of the preceding buffer-relevant customer in the station into which the customer arrives;

2. In the same way accept one arrival of c_2;

3. Kit time = maximum (arrival time of c_1, arrival time of c_2);

4. Assembly service end time = maximum (Kit time, previous departure time at Assembly) + Assembly service time;

5. Accept one arrival of c_3 in the same way in which c_1 and c_2 arrivals were simulated;

6. Service end time at lone workstation = maximum (arrival time of c_3, previous departure time at that workstation) + Service time;

7. The next merging station = The station which had the least service end time;

8. Take merging station's customer as the next customer into the merge node after taking into account the buffer-relevancy comparisons;

9. Simulate the departure of that customer from the merge node;

10. Replenish the depleted merging station with next customer's service end time;

11. Repeat steps 7 to 10 till the required number of customers have been simulated through the merge node.

Figure 3. Fast Simulation Algorithm to Simulate Prototype System in Figure 2.1

generalizes the abstractions presented by Duse (1994). For example, Duse's (1994) approach of "customer-by-customer-with-switching" for a parallel server station is nothing but "pulling customers into a server in the parallel server station as and when the server is depleted of a customer." In the case of a merge topology too, customers are pulled into a merging line when it is depleted of a customer. In the case of an assembly topology, customers are, again, pulled into the system as and when needed, namely, when the next kit is to be simulated out of the assembly server. Adding some complexity to how the assembly server works, we can have more than one unit of a component going into the assembly server. Lets say, one unit of component c_1, and two units of c_2 go into a single assembly; then for every unit of c_1, two units of c_2 have to be pulled into the

assembly server. The above situation presents a clearer instance of "pulling customers into a server as and when needed."

While we have described the basic philosophy of the new approach using a small prototype system shown in Figure 2.1, a bigger, more complicated system may be designed as shown in Figure 4 for a proof-of-concept implementation of the above-mentioned approach. The bigger system has at least one specimen each of the important manufacturing network building blocks such as tandem line, assembly, merge, and parallel server workstation mentioned earlier.

Description of the Experimental Prototype

For ease of representation as a model input, a parameterized 3-dimensional addressing mechanism is used to describe the system. A high level unit called "level" is said to contain tandem lines in it, with the tandem lines containing the workstations. Though essentially the "level" in this work achieves what Hunt's "level" does, the scope of "level" is slightly broadened here to act more as an addressing abstraction than anything else. Any tandem line is represented as tandem<level #>.<line #> and any buffer is represented as buffer<level #>.<line #>.<workstation #>.

The experimental prototype has two tandem lines- tandem1.1 and tandem1.2 leading into an assembly station represented by the lone server in tandem2.1, with the component buffers, buffer2.1.1 and buffer2.2.1 accepting components from tandem1.1 and tandem1.2, respectively. Customers from this assembly station (which is taken to be at level 2) are processed by workstations in tandem3.1 at level 3, and the customers merge into tandem4.1. Tandem4.1 also has customers from tandem3.2 merging into it.

After tandem4.1 lies a parallel server workstation which is taken to be at level 5. The two parallel servers that form the parallel server workstation are represented as lone servers in tandem5.1 and tandem5.2, with buffer5.1.1 serving as the lone buffer associated with the parallel server station. After the parallel server station in level 5, comes tandem6.1 in level 6. How this schemata for representing the system is amenable as simulation program input is elucidated in Chapter VI.

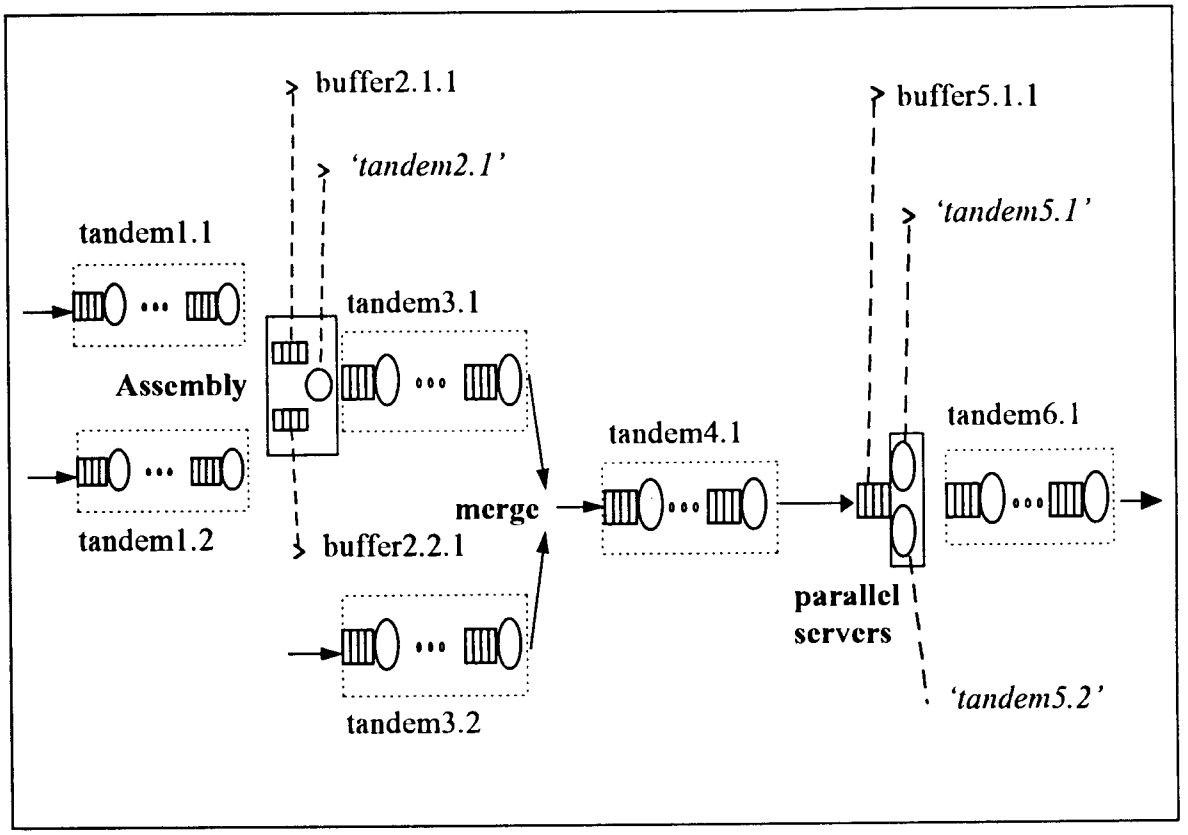


Figure 4. Fast Simulation - An Experimental Prototype

CHAPTER VI

FAST SIMULATION - IMPLEMENTATION

Design of Reusable Fast Simulation Building Blocks

The primary aim of this research was to find a methodological framework for fast simulation of general manufacturing systems as well as the design of a simulation program in a general purpose language for the same. However, the applicability of the methodological framework and the resulting implementational paradigm will largely be inhibited if every time a system has to be fast simulated, a time consuming general purpose language program has to be written. So all along the research phase it was considered necessary to be sensitive to, and to be on the look out for user friendly program structures/reusable program constructs. This is not merely a programming effort. The foundations of such reusable, modular structures have to begin from the essential outlook the fast simulation methodological framework should take. The memory addressing mechanism that we will describe later in this chapter was designed with reusability and user friendliness in mind. Actually, the mechanism provides us means to parameterize all variables so as to have user friendly, reusable programming routines. Also, the programming logic should be so manipulated to vest the routines with some independence so that they may be assembled together to build a simulation program quickly and easily.

In the prototype system shown, there are four different topologies that make up the system- tandem lines, assembly server, merge, and parallel server workstation. If we

can have independent modules to simulate each of these topologies so that they may be assembled together to make up the simulation program for simulating the topologies figuring in as a part of the system in any combination, that would be a sizable step towards developing a user friendly fast simulation "language". Forthcoming is a discussion on how such modules were constructed due to the outlook that we chose to take in fast simulation.

Tandem Line: The basic code here is the same as that proposed by Chen and Chen (1993). However, we can vest the module with a lot of independence if we do not attempt to get the departure time from the last server of the tandem line in the *tandem line module* itself. Wherever the tandem line module appears in the program it does what it is supposed to do except the last step of releasing the customer from the last server. This last step is handled in each forthcoming level according to the specifications/intent in that level.

Assembly: Once we have components in all the component buffers in the assembly server, we can get the kit completion time. Then the assembly start time and the assembly end time are obtained. Once this assembly end time is obtained, the departure time may be computed after taking into consideration the buffer availability times in the forthcoming stations. Then, the component buffer linked lists are updated with that departure time. It can be seen that, once within the *assembly module*, all information needed for the simulation program at that point of execution (like the component buffer arrays to be updated) are available within the module. This makes the *assembly module* independent.

Merge: The tandem line module is vested with the capability of getting the departure time from any tandem line/topology upstream to it. This way any line may be connected to any other topology/tandem line. In this context, capturing a merge scenario would mean connecting the merge node/line to the previous lines depending on the service end times of the customers at the merging servers. The merging line itself is decided by a simple conditional statement.

Parallel server workstation: We have to get the customers to the various servers in the workstation so that their service end times may be compared to decide upon the earliest departing customer. However, we have to update the lone buffer array associated with the workstation every time a departure takes place (no matter from what server of the parallel set). This is easily done by having a routine (that may be called "NextToParallel") that would update that lone buffer array (and only that) every time a departure from the parallel set of servers is obtained.

The simulation logic presented in the above discussion has been translated into C and is included in Appendix II. Basic knowledge of C may be useful for understanding the program; however, it has to be stressed that the above logic may be coded in any general purpose language.

The reusable building blocks may be used to quickly configure fast simulation models of prototypes in Figure 2.1 and Figure 2.2, as shown next.

```
Assembly Module; //simulates assembly
Lone Workstation Module; //simulates a single workstation
for (number of customers to be simulated through the merge node)
{
        if (Assembly Module Service End Time <= Lone Workstation Service EndTime)
        {
                Merge Node Module; //simulates a merge node
                Assembly Module;
        }
        else
        {
                Merge Node Module;
                Lone Workstation Module;
        }
}
```

Figure 5.1. Psuedocode showing modules that simulate the system in Figure 2.1

```
Lone Workstation(1) Module; //simulates Lone Workstation 1 (hence, parameter '1')
Lone Workstation(2) Module; //simulates Lone Workstation 1 (hence, parameter '2')
for (number of assemblies to be simulated)
{
 if (Lone Workstation(1) Service End Time < Lone Workstation(2) Service EndTime)
        {
        Merge Node Module;
        Assembly Module;
        Lone Workstation(1);
        }
 else
        {
        Merge Node Module;
        Assembly Module;
        Lone Workstation(2);
        }
}
```

Figure 5.2. Psuedocode showing modules that simulate the system in Figure 2.2

Executional Details of the Fast Simulation Methodology

Our aim here is to show that the fast simulation methodology developed to simulate a generic system consisting of common manufacturing network topologies is indeed "fast". What we mean by "fast" is that the recursion that works (and makes them faster) in the cases of the individual manufacturing network topologies works in exactly the same way in our methodology for fast simulation of larger systems. That is to say, there are no additional steps that we have to do to keep the logic of the simulation going to accommodate a generic system into a "fast simulation" methodology. Re-wording the above statement we can say, "given that we have fast simulation models of the manufacturing network topologies, can we develop a schemata for simulating them, only now, they all being a part of a bigger system." The following discussion should also clearly show how our methodology would indeed be "faster" for any combination of topologies when compared to the corresponding DES simulation.

Adopting the symbols that Chen and Chen (1993) use for discussion of their execution time for a tandem line, we have

cmp = comparison of two quantities and getting the minimum or maximum of them;

add = addition; and

rv = random variate generation.

Let us now step through the fast simulation of the prototype shown in Figure 4, in Chapter V. In the process we would have listed the computational steps involved and would see how the schemata simulates the various topologies while they form a bigger, buffered general system. Then it will be clear how the new methodology should indeed be faster than DES.

At *level 1*, we accept a customer through *tandem1.1*. We fast simulate it up to the last station where we compute only its service end time. For determining the departure time of that customer from the last workstation of *tandem1.1*, one **cmp** between service end time at the last workstation of *tandem1.1* and the entry time into the *buffer2.1.1* is needed. However this is spared at that level, and done at the next level. As indicated earlier in our discussion on modules, this procedure gives some independence to the tandem line module. In a similar fashion, a customer is simulated through *tandem1.2* till its service end time at the last workstation of *tandem1.2*. These two procedures take place for every customer taken into the tandem lines. Hence, we can see that this part of the program is just fast simulation of a simple tandem line.

Then the attention shifts to *level 2* for simulation of the assembly server. Components are taken into *buffer2.1.1* and *buffer2.2.1* by the **cmp** function which was spared in the *level 1* procedures. This **cmp** function compares service end times at the last workstations of *tandem1.1* and *tandem1.2* and the space availability at *buffer2.1.1* and *buffer2.2.1*, respectively. After this we have to compute the kit completion time (time when all components are available for next assembly). This is another **cmp** between the departure times from the last workstations of the tandem lines of the previous level. Then the service start time on the assembly is computed with a **cmp** between the kit completion time and the previous departure time from the assembly station. An **add** function between the assembly start time and the assembly time (generated by **rv** function) gives the assembly end time. This sums up the list of actions in *level 2*, assembly.

As can be seen there are no other additional steps to model the assembly when it is a part of a bigger system than when it stands alone; except for the minor changes wherein we leave the onus of getting a departure time from the previous level to the present level of the assembly. In the bargain we do not add any additional steps to the iteration, but merely perform a step at another place instead of where one would expect it to be performed. This is done to facilitate some modularity in the code. More specifically, we can have a function that simulates any tandem line till the service end time at its last workstation. This can be done only if the tandem line code should not be caring about what lies ahead of it in the system.

Next the focus shifts to *level 3*; more specifically to *tandem3.1*. *Tandem3.1* pulls the customer from the *level 2* assembly server and simulates the customer through itself, till the customer's service end time at the last workstation. Now we cannot proceed unless we get the service end time of a customer at the last workstation of *tandem3.2*. So a customer is processed through *tandem3.2*. Once this is done, a **cmp** between service end times of customers at the last workstations of *tandem3.1* and *tandem3.2* is made. The line with the lesser service end time at the last workstation is chosen as the present merging line. *Tandem4.1*, at the next level, now pulls a customer from the present merging line and takes it through till the last workstation of the *tandem4.1*. After that, the present merging line which is depleted of one customer, is replenished with the next customer so that the simulation may be kept going.

Then, *tandem5.1* with its lone workstation representing one server of a parallel server station pulls the customer from *tandem4.1* and its service end time is computed. Now we have to get a service end time at the lone workstation of *tandem5.2*

(representing the other server of the parallel server station). So another customer is pulled from *level 3* into *level 4* and then into *tandem5.2*. A **cmp** between the service end times at two stations which represent the two parallel servers, decides which customer should be pulled into the next level, namely *level 6*, with its *tandem6.1*. Just as in the case of the merging tandem lines, the parallel server which has a departure simulated, is filled with the next service end time. This means a customer is brought in from start of the system to reach the parallel server and the next service end time at that server is computed.

The customer is processed completely through *tandem6.1* when it exits the system. But note that we do not compute the departure time of a customer from the last workstation of a tandem line until we move to the next level. In the case of *tandem6.1*, since the service end time would be the departure time we may just add a line of code which would equate the departure time of a customer from the system to the service end time at the last workstation of *tandem6.1*.

Thus, we see that with no more than the steps used by Duse (1994), we can simulate the topologies while they form a larger, finite buffered system.

Memory Management of the Fast Simulator

Even though the memory space available in computers is increasing day by day, efficient memory management is still a challenge if not a problem. An efficient memory management procedure for the proposed methodology based on Chen and Chen's (1993) procedure is given below.

In their paper, Chen and Chen (1993) make repeated use of the same memory space. In their procedure, the storage cost is dependent on the number of stations and the sum of the buffer spaces in the system, and independent of the number of customers to be simulated. The question, then, to be asked is "can the same data management procedure be used here or not, and if not what changes/adaptations have to be made to achieve the same." With our convention for nomenclature, a workstation can be addressed by three parameters, the level, the tandem line, and the workstation number. So whatever variables we need to associate with every workstation for purposes of simulation/statistics collection we will have three dimensional arrays associated with them.

The variables so needed are

1. *departure time[level][tandemLine][WS]*
2. *service end time[level][tandemLine][WS]*
3. *start time[level][tandemLine][WS]*
4. *previous-departure time[level][tandemLine][WS]*
5. *service time[level][tandemLine][WS]*
6. *total service time[level][tandemLine][WS]*
7. *total queue time[level][tandemLine][WS]*
8. *total block time[level][tandemLine][WS]*

In C (the language of our implementation), to make things easier for manipulation we may choose to have the arrays used, start with array element '1' and may leave the '0^{th} element unused. However the existence of this '0^{th} array element may be used to our advantage to store the arrival time of a customer into a level, e.g.

departure time[level][tandemLine][0] = arrival time of a customer into the level. *departure time[level][tandemLine][1]* is used to denote the departure time at the first workstation of a line, so on and so forth.

Once the departure time of a customer from a workstation is scheduled to be generated, the present value of *departure time[level][tandemLine][WS]* is assigned to *previous-departure[level][tandemLine][WS]* (used for queueing time computations). The new departure time is then assigned to *departure time[level][tandemLine][WS]*. This way, we see that we can have repeated use of the same memory space (similar to Chen and Chen's procedure). Thus the memory space needed by departure times is the product of the number of stations plus 1 (for arrival mechanism) "(M+1)" and the size of data type of a departure time.

In the case of finite buffer sizes, each station needs a circular linked list which records the departure times used to trace blocking situations. The length of a circular linked list is equal to the buffer size (including one in service) of the station it belongs to. Whenever one departure time is generated, it is stored in both an array (the array *departure time[level][tandemLine][WS]*) and a circular linked list. The one stored in the array is replaced by departure of the next customer departing from the workstation (like in the infinite buffer case). On the other hand, the one stored in the circular linked list will be replaced by the departure of the "later block-relevant" customer after that is used for checking the blocking situation of the departure of the "later block-relevant" customer from the previous workstation. Thus in addition to the memory needed in the infinite buffer cases, we need more memory space for the circular linked lists. This

additional storage cost equals the product of the sum of the total buffer sizes of all stations and the size of data type of *departure time[level][tandemLine][WS].*

Storage cost (finite buffer) = (Number of workstations + 1 + sum of buffer sizes of all workstations) * SizeOf(data type of departure time)

There exists a potential problem associated with the memory space in the finite buffer cases. When the size of a buffer is very large, the circular linked list will become very long, and consequently large memory space is required. The solution to this problem is to remove the circular linked lists of the stations with very large buffer sizes (e.g. more than 1,000) if an infinite buffer approximation is deemed acceptable.

A brief explanation has to be given about how the assembly server and parallel server are captured for purposes of implementation. A problem arises because assembly server has one server and several component buffers associated with it. Likewise, the parallel server workstation has one buffer, but several servers associated with it. However, according to our schemata of representing the system, it can be seen that only one server can be associated with a buffer and vice versa. The implementational obstacle in the case of the assembly server was alleviated by having as many servers as there are component buffers, but treating only one server as the 'working' assembly server. The assembly code was tailored to meet this situation, by forcing all the component buffers to be updated as and when one customer gets out of the one 'working' assembly server. In a similar fashion, the parallel server workstation was represented by as many servers and buffers as there are parallel servers in the station, but treating only one buffer as the 'working' buffer. The parallel station code was tailored for this abstraction by forcing

the 'working' buffer to be updated every time a customer gets out of any of the parallel servers.

Verification of Fast Simulation Logic

The fast simulation program was tested for individual configurations such as tandem line and assembly and the results were compared with previous results. The simulation output obtained from the fast simulation program should give exactly the same results as the corresponding DES program in C if the random numbers generated in DES match exactly with the corresponding ones in the fast simulation program. If this is not the case, then the results should be the same, statistically. The latter holds in our case as SLAM II (Pritsker 1986) was used for the verification.

The system parameters for the experimental prototype are shown in Table 1. The mean and standard deviation of ten samples of the simulation end time for 50,000 customers passing through the system (shown in Figure 4) obtained through fast simulation and SLAM II are shown in Table 2. The null hypothesis that the means of simulation end time are the same was not rejected at the 95% confidence level. The z statistic for the above sets of data while comparing them for the null hypothesis that the means are the same, was 0.778. This is less than 1.96; hence we do not reject the null hypothesis at the 5% significance level.

LEVEL NUMBER	PARAMETERS

1
Tandem lines 1.1 and 2.1
	buffer1.1.1 = 1; buffer1.2.1 = 1;
	service time 1.1.1 = 0.05*; service time1.2.1 = 0.05;
	interarrival time1.1 = 0.7; interarrival time1.2 = 0.7*

2 Assembly Station	buffer2.1.1 = 5; buffer2.2.1 = 5; service time2.1.1 = 1;

3
Tandem lines 3.1 and 3.2
	buffer3.1.1 = 1; service time3.1.1 = 0.01;
	buffer3.2.1 = 1; service time3.2.1 = 0.01;
	buffer3.2.2 = 1; service time3.2.2 = 0.01;
	interarrival time3.2 = 0.7;

4 Tandem line 4.1	buffer4.1.1 = 1; service time4.1.1 = 0.01; buffer4.1.2 = 1; service time4.1.2 = 0.01;
5 Parallel server station	buffer5.1.1=4; service time5.1.1 = 0.5; service time5.2.1 = 0.5;
6 Tandem line 6.1	buffer6.1.1 = 6; service time6.1.1 = 0.5;

***mean of the exponential random variate generator**

Table 1. System Parameters for the Experimental Prototype

SLAM II	FAST SIMULATION
mean = 25156	mean = 25153
standard deviation = 69.15	standard deviation = 83.9
sample size = 10	sample size = 10

Table 2. Verification of Simulation End Time for Prototype System

A Discussion on Linearity of Execution Time of Fast Simulation

One basic expectation in pure fast simulation is that as the 'size' of any particular topology increases the simulation execution time increases linearly. However, we have to be sure to see the subtleties in the various ways the system size could increase and what would each mode of increase means in the context of execution efficiency. Before we go into any discussion in that direction, we summarize the findings of previous researches.

- For a tandem line (with single servers), as the number of servers increases, the execution time increases linearly for fast simulation, while it seems to increase exponentially for DES (Chen and Chen 1993).
- Regarding parallel server stations Duse (1994) has the following comments- "The execution time for both fast and discrete event simulation increases with the number of servers but the CPU time for discrete event simulation increases at a faster rate ("drastically") as compared to the fast simulation. Thus, the greater the number of servers at the parallel server station, the higher the savings achieved by employing fast simulation. The increase in execution time for fast simulation can be attributed to the "switching task". The higher the number of servers, the higher the time required to determine the part to which the focus of fast simulation needs be shifted. The increase in CPU time for discrete event simulation can be attributed to the fact that the average length of the event list increases with the increase in number of servers at the parallel server."

- Confirming the speed-up obtained for all topologies he studied, Duse adds, "Significant speed-up was achieved in all cases by the use of fast simulation."

The execution time increases linearly with the number of customers processed for both fast and discrete event simulation; but the rate of increase for discrete event simulation is higher than that for fast simulation.

Let us see why the above linearities should hold in our methodology too.

- Increase in the number of servers in one or more tandem lines.

  In this case, the loop which simulates the servers in a tandem line loops one more time for each server added to the tandem line. This results in a linear increase in the execution time.

- Increase in the number of tandem lines bringing in components into the assembly.

  Here the newly added tandem line is simulated just like the other tandem lines, making the execution time increase linear. However as it can be recalled, the tandem lines are simulated only till the service end time in their last workstations. From there onwards, the departure time of a customer from the last workstation of the line into the corresponding component buffer is computed through a single **cmp** operation. An additional component coming into assembly would mean an additional **cmp** operation. This results in a linear increase in execution time. Once all the components have come into the corresponding component buffers, the kit completion time is a **cmp** operation. With an increase in the number of components being assembled, the number of **cmp** operations also increases correspondingly. This results in a linear increase in execution time too.

- Increase in the number of tandem lines merging.

  As the number of tandem lines merging increases, the additional tandem lines should be simulated by Chen and Chen's procedure; this leads to a linear increase in execution time. Also, every additional merging tandem line increases the number of **cmp** operations for determining the present merging line by one. This leads to a linear increase in execution time too.

- Increase in the number of servers in a parallel server workstation.

  Here as the number of servers increases, number of **cmp** operations to determine the forthcoming departing customer increases correspondingly. This leads to a linear increase in the execution time.

All the above possibilities in system size increase were studied through simulation experiments. The above theories were verified with the experimental results graphed in Figures 6.1 through 6.4. Figure 6.1 plots the execution time against the number of workstations in the tandem line at level 4. Figure 6.2 plots the execution time against the number of assembly feeder lines that feed into the assembly at level 2. Figure 6.3 plots the execution time against the number of lines merging. For this experiment, level 1 and level 2 were done away with. Lines similar to tandem3.2 were made to merge into tandem4.1. Figure 6.4 plots the execution time against the number of parallel servers in the parallel server workstation in level 5. For purposes of experimentation, level 1, level 2 and level 3, were removed from the experimental prototype.

In figures 6.1, 6.3, and 6.4, the slight departure from the linearity is due to the differences in the execution times that can be attributed to the difference in the number of lost customers, and the resulting difference in the number of random variate generations.

While we are still in a discussion on linearity in the context of increase in system size in the above mentioned modes, one should also be sensitive to the question- what would the execution time increase be like, when the system size increase is accompanied with increase in system complexity, like, say, the increase is an additional topology. While one can think of ways and means to estimate the increasing trend in such a case (afterall, even an additional topology is handled by **cmp** operations), such a study would be beyond the scope of this work and may be left as a possible issue to be looked at by future researchers.

Figure 6.1. Increase in Execution Time with Increase in Size of a Tandem Line (100,000 customers)

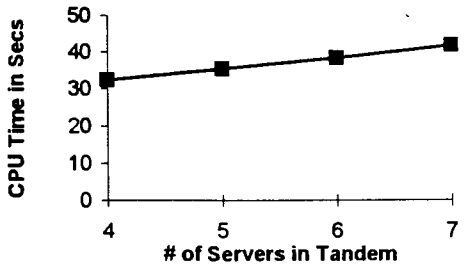


Figure 6.2. Increase in Execution Time with Increase in Size of Assembly Stage (100,000 customers)

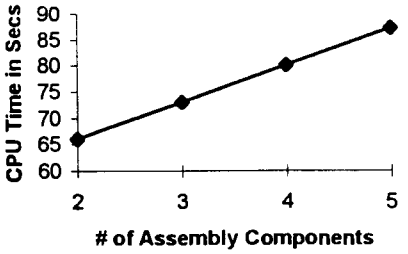


Figure 6.3. Increase in the Execution Time with Increase in the # of Merging Lines

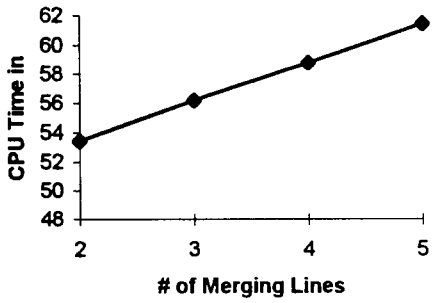
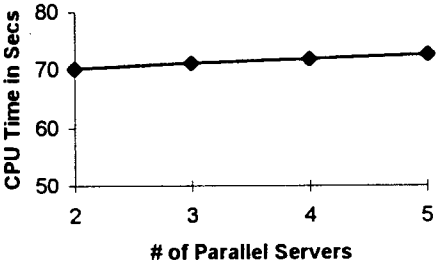


Figure 6.4. The Increase in Execution Time with Increase in the Number of Parallel servers



CHAPTER VII

RESEARCH SUMMARY, CONTRIBUTIONS, AND FUTURE RESEARCH

The first section of this chapter summarizes the research results and links the research outcomes with the research objectives defined in Chapter III. Next, the contributions of this research to the field of Industrial Engineering, specifically to "Modeling and Simulation -Body of Knowledge" are identified. This chapter concludes with a section which outlines possible areas, issues and problems for future research.

Research Summary

Five research objectives which were defined in Chapter III served as the prime directive for this research.

The first objective of this research was to investigate issues related to the combining of various network topologies into a reasonable manufacturing system for the purpose of simulating the same using fast simulation. Implicitly, that meant, "can various topologies be combined into a reasonable network for purposes of fast simulating the same?" For this purpose, a prototype system of considerable difficulty was designed, and ways and means (and ideas) to fast simulate the same were studied. All the preliminary issues were laid out as discussions and these discussions helped us conceive the proof-of-concept prototype systems (objective 1). Initially, the potential of the proposed methodology was investigated using two simple prototypes (objective 2). That also brought in issues of how the system should be represented for capturing the same as input data for the code to fast simulate and give out the output. A working 3-

dimensional parameterized method of representing the system was designed to capture the system for purposes of inputting data about the system for the simulation code to work on. Typical statistics collection variables were associated with each workstation. The new approach of "pulling customers into the network as and when needed" was implemented and tested for the bigger proof-of-concept prototype (objective 3).

In any attempt to investigate the functional relationship between execution time of fast simulation and system (objective 4), there are two issues to be considered-

1. Will this new extension of fast simulation produce the same simulation output as discrete event simulation?
2. Will the new methodology be efficient in execution?

1. For simulation output value verification, the prototype system was simulated in SLAM II and the output results were compared and statistical tests were conducted to show that there was no significant difference between the SLAM and fast simulation results. If a discrete event code with identical random number generators was developed then the values should be exactly identical.
2. The basic philosophy of recursion based simulation models being faster than DES is that, as system size increases (in individual topology sizes) the execution time increases linearly unlike in DES. So, if such linearity of execution time increase is noticed as was expected, then we can be sure that this methodology is indeed 'fast' simulation. Also it is to be noted that DES execution time increases tremendously (normally, exponentially, with a simple event list implementation) as system size and complexity increases. In this case, the expected linearity was confirmed numerically.

The dissemination of knowledge resulting from the documentation of this thesis and the archival publications that result from the work would satisfy objective 5.

Research Contributions

Fast simulation has been used previously for improving the simulation execution efficiency of topologies in isolation. However, the lack of a paradigm to apply fast simulation for larger systems would have inhibited the continued evolution of and subsequent applicability of fast simulation in research and industry. Researchers and industry personnel want faster, easier and more generic tools to study real systems which are increasingly becoming complex. This research has contributed positively in all the above three directions.

The basic idea is simple. The implementation is highly simple, elegant and easily understandable. The method is more generic in scope and applicability compared to the fast simulation techniques previously available.

The specific contributions of this research to the "Modeling & Simulation -Body of Knowledge" are-

1. New modeling abstraction of "pull customers into the network as and when needed" was developed. This contribution is a sizable development which should keep fast simulation alive and give it more applicability, visibility and attention.
2. Feasibility and viability of fast simulation in simulating FCFS general manufacturing networks was demonstrated.
3. Insights were provided for configuring and executing fast simulation of general manufacturing networks.

4. A user-friendly, 3-dimensional addressing mechanism was conceived as a model representation abstraction for fast simulation of general manufacturing networks. This enabled the development of independent modules that can be put together to simulate larger systems.
5. Laid the foundation for future research in the area of fast simulation and hybrid (fast/DES) simulation.

Listed below are some additional benefits of the new methodology, which though present in the previous fast simulation abstractions, are more pronounced in this case (of bigger, complex systems).

1. When we have finite buffers (as is the case in this study), deblocking can be complicated in the DES code; here the finite buffer constraint is handled implicitly in the simulation logic without a need for explicit functions for deblocking.
2. Previously only individual topologies could be simulated by the corresponding fast simulation code while they form a part of larger hybrid simulation system; now bigger system sectors can be simulated by fast simulation while they form part of a larger hybrid simulation system. This could prove to be an impetus to hybrid simulation.
3. This kind of fast simulation of larger systems lends itself well to parallel processing.
4. Fast simulation methodology would have as its selling point the fact that its execution time is nearly unaffected by increase in traffic rate. An increase in execution time due to extra random number sampling for lost customers in the finite buffer case is unavoidable and equivalent in DES and fast simulation.

Future Research

Several unanswered questions were identified in Chapter II and only a few of them were chosen for inclusion within the scope of this research. These remaining unanswered questions, along with any offshoots from this research effort, can set a possible course for future research in this area.

The following may be thought of as the significant issues that deserve inclusion in this future research agenda.

1. Tagging customers so that their time in system can be computed.
2. Inclusion of unreliable servers or other topologies which were not included in this research.
3. Issues of trade-offs between approximations for the system and the resulting speed-up.
4. Integration of various research efforts in this area of speeding up simulation, such as PDES, hybrid simulation, and metamodeling.
5. Investigation of applicability of fast simulation to systems with non-FCFS queue disciplines.
6. Studies on increase in execution time with increase in system complexity.

Hormonosky, C.M. (1990), "Implementation Issues: Using Simulation for Real-Time Scheduling, Control, and Monitoring," In *Proceedings of the 1990 Winter Simulation Conference*, O. Balci, R. Sadowski, and R. Nance, Eds. IEEE, Piscataway, NJ, 595-598.

Hunt, C.S. (1994), "Fast Simulation of Open Queueing Systems," M.S. Thesis, School of Industrial Engineering, University of Oklahoma, Norman, OK.

Hunt, C.S. and B.L. Foote (1995), "Fast Simulation of Open Queueing Systems," *Simulation 65*, 3, 183-190.

Kamath, M. (1994), "Recent Developments in Modeling and Performance Analysis Tools for Manufacturing Systems," In *Computer Control of Flexible Manufacturing Systems -Research and Development,* (Eds. Sanjay B. Joshi and Jeffrey S. Smith), Chapman & Hall, London, UK, 230-263.

Kelton, W.D. (1994), "Perspectives on Simulation Research and Practice," *ORSA Journal on Computing 6*, 2, 318-328.

Kreutzer, W. (1986), *System Simulation: Programming Styles and Languages*, Addison-Wesley, Reading, MA.

Nance, R.E. (1971), "On Time Flow Mechanisms for Discrete System Simulation," *Management Science 18*, 1, 59-73.

Paul, R.J., (1991), "Recent Developments in Simulation Modelling," *Journal of Operational Research Society 42*, 3, 217-226.

Pritsker, A.A.B. (1986), *Introduction to Simulation and SLAM II*, Third Edition, Halsted Press, New York, NY.

Reeves, C.M. (1984), "Complexity Analyses of Event Set Algorithms," *The Computer Journal 27*, 1, 72-79.

Viswanadham, N. and Y. Narahari (1992), *Performance Modeling of Automated Manufacturing Systems*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

APPENDIX I
(Algorithms for simulating individual topologies)

FAST SIMULATION OF A TANDEM LINE WITH SINGLE SERVER STATIONS

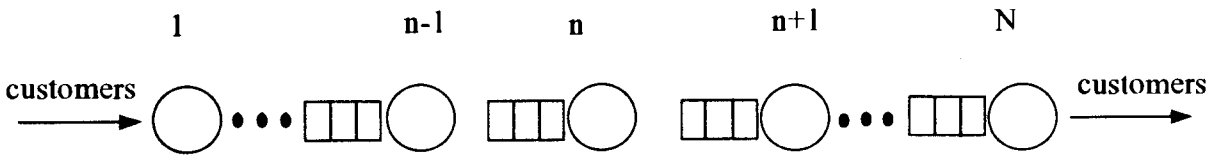


Figure 1.1. Tandem Queueing System

Description of the System

The system being studied consists of a series of single server stations processing customers consecutively. The customers enter the system through the first station, and get processed consecutively through all the stations and depart from the last one. It is to be noted that the order of customers, as they move through the tandem line, remains unchanged, and this property is taken advantage of while fast simulating them.

Assumptions

1. There are no separate set up times for the customers.
2. The customers are processed on an FCFS basis.
3. There is ample waiting space in front of the first workstation.

Nomenclature

i : Index of station.

j : Index for customer.

$T_{i,j}$: Service time of j^{th} customer on i^{th} station.

B_i : Input buffer capacity of i^{th} station (excluding the one space in the server).

$S_{i,j}$: Service start time of j^{th} customer on i^{th} station.

$E_{i,j}$: Service end time of j^{th} customer from i^{th} station.

$D_{i,j}$: Departure time of j^{th} customer from i^{th} station (same as $E_{i,j}$ if input queue of downstream station has infinite capacity).

U_i : Utilization of station i.

TIQ_i : Average waiting time in queue at station i.

BT_i : Average blocking time at station i.

M : The number of stations in the tandem line.

N : The number of customers for which simulation is to be run.

A_j : The arrival time of customer j which is generated by the random number generator and which is an input to the simulation.

TP : Throughput of the tandem line.

TIS : Average time in system of a customer.

The inputs are M, N, A_j, B_i, $T_{i,j}$.

Recursive Calculations

$$S_{i,j} = \max\{D_{i-1,j}, D_{i,j-1}\} \qquad [a]$$

The start time of customer j on station i is the maximum of departure time of that customer from the previous station i-1, and the departure time of previous customer j-1 from that station i.

$$E_{i,j} = S_{i,j} + T_{i,j}$$

Service end time of customer j on station i is the sum of the start time of that customer j on that machine i, and the processing time of that customer on that station.

$$D_{i,j} = \max\{E_{i,j}, D_{i+1,j-(B_{i+1}+1)}\} \qquad [b]$$

Departure time of customer j from station i is the maximum of service end time of that customer on that station and the departure time of customer 'j-(B_{i+1}+1)' from the next station i+1.

Intermediate Variables

$SumU_i$: Variable for statistics collection for utilization.

$SumTIQ_i$: Variable for statistics collection for waiting time in queues.

$SumBT_i$: Variable for statistics collection for blocking time in servers.

SumTIS : Variable for statistics collection for time in system.

Performance Measures

$U_i = SumU_i / D_{M,N}$

$TIQ_i = SumTIQ_i / N$

$BT_i = SumBT_i / N$

$TIS = SumTIS / N$

$TP = N / D_{M,N}$

Algorithm for Implementation

For j = 1 to N ** Beginning of Simulation **
do
{Generate arrival time of customer j, A_j
for node i = 1 to M do
{** Determine the time at which service can start at station i**
$S_{i,j} = \max\{D_{i-1,j}, D_{i,j-1}\}$ [a]
Generate processing time for i, $T_{i,j}$
$E_{i,j} = S_{i,j} + T_{i,j}$

$D_{i,j} = \max\{E_{i,j}, D_{i+1,j-(B_{i+1}+1)}\}$ [b]

** Intermediate variables collected for calculation of performance measures **

{

$SumU_i = SumU_i + T_{i,j}$

$SumTIQ_i = SumTIQ_i + (S_{i,j} - D_{i-1,j})$

$SumBT_i = SumBT_i + (D_{i,j} - (S_{i,j} + T_{i,j}))$

}

}

$SumTIS = SumTIS + (D_{M,j} - A_j)$

}

** Calculate performance measures **

{ for node i = 1 to M

do

{ $U_i = SumU_i / D_{M,N}$

$TIQ_i = SumTIQ_i / N$

$BT_i = SumBT_i / N$

}

$TIS = SumTIS / N$

$TP = N / D_{M,N}$

}

** Queue lengths can be calculated by Little's Law **

FAST SIMULATION OF A SINGLE SERVER ASSEMBLY QUEUEING SYSTEM WITH INFINITE KIT BUFFER LIMIT

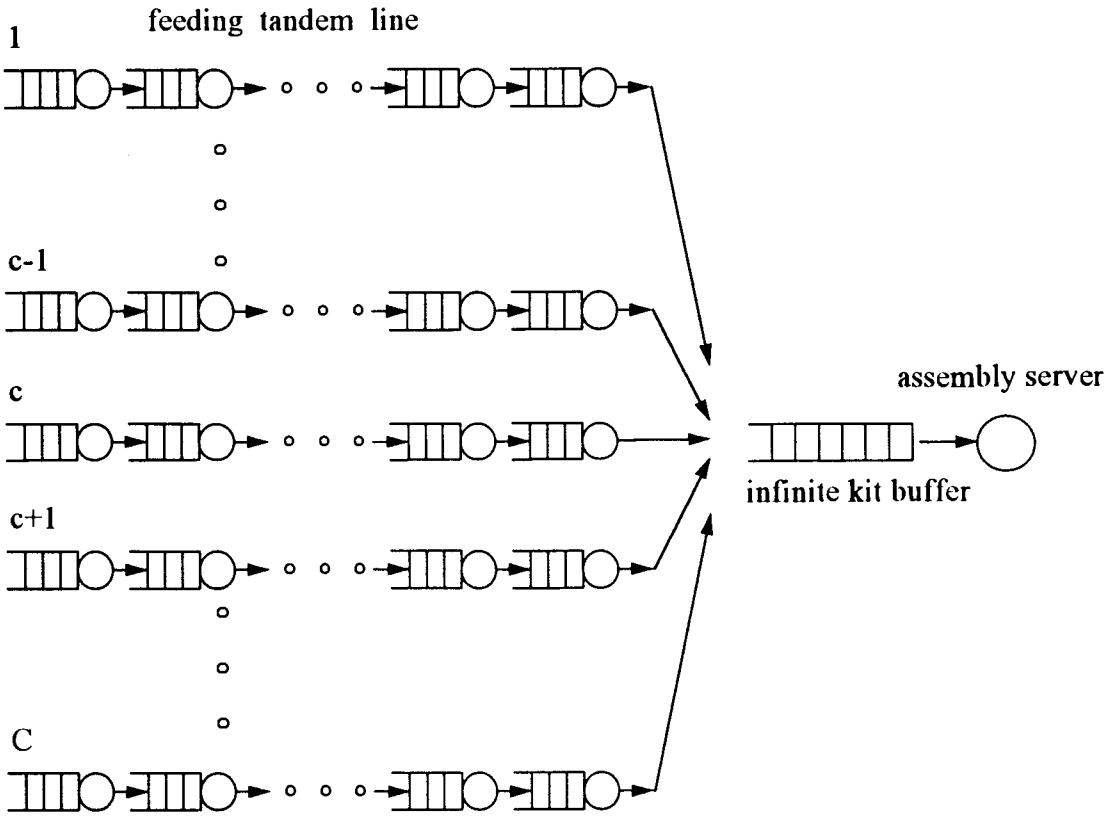


Figure 1.2. A Typical Single Assembly Queueing System

Description of the System

The system consists of C component types coming through C separate tandem lines. If one component of each type is available in the component buffers, they are combined to form a kit which waits in a kit buffer before being assembled into the final assembly by a single server assembly station.

Assumptions

1. There are no separate set up times for the assembly server.
2. The kits are assembled on an FCFS basis.
3. There is ample waiting space in front of the assembly server.

Nomenclature

n = Index for assemblies.

S_n = Service start time for assembly n.

K_n = Time kit n is formed.

T_n = Assembly time for assembly n.

D_n = Departure time of assembly n.

N = Number of assemblies which are to be simulated.

D_c^q = Departure time of q^{th} unit from the last machine of feeder line of component c.

c = Index for component type.

C = Number of component types.

q_c = Number of units of component c going into a single assembly.

U = Utilization of assembly server.

TP = Throughput of the assembly server.

TIS_c = Average time in system of component c.

CWT_c = Average waiting time of component c before kitting.

The inputs are T_n, N, C, and q_c.

Recursive Relationships used for Simulation

$$K_n = \max_c (D_c^q c) \qquad [a]$$

The kit formation time of kit n, K_n, is the maximum among the departure times of q_c units of corresponding component 'c's.

$$S_n = \max \{ K_n, D_{n-1} \} \qquad [b]$$

The service start time of an assembly n is the maximum of the kit formation time, K_n and the departure time of previous assembly, D_{n-1}.

$D_n = S_n + T_n$ [c]

The departure time of an assembly n is the sum of the service start time S_n and the assembly time T_n.

Performance Measures of Interest

$$U = \sum_{n=1}^{N} T_n / D_N$$

$$TP = N / D_N$$

$$TIS_c = \sum_{c} SumTIS_c/(N*q_c)$$

$$CWT_c = \sum_{c} SumCWT_c/(N*q_c)$$

Algorithm of Implementation

```
for n = 1 to N          ** Beginning of simulation of N assemblies **
{
  for c = 1 to C** Process all component types through their respective feeder lines **
      {
      Process qc units of type c through feeder line of c using the fast simulation model
      of the tandem line and store departure time of qc^th unit from last machine of the
      line, as Dc^q
      if Dc^qc > Dc-1^qc-1
      Kn = Dc^qc      ** Storing the max. arrival time as the time a kit n is formed **
      }
```

$S_n = \max \{K_n, D_{n-1}\}$ ** To compute the assembly start time for assembly n **

$D_n = S_n + T_n$ ** To compute the departure time for assembly n **

$SumT_n = SumT_n + T_n$ ** To store the sum of all assm. times for stat. colc. **

for c = 1 to n

{

$SumTIS_c = SumTIS_c + (q_c * D_n - \sum_{q=1}^{q_c} D_c^q)$ ** Store info for stat. colc. of TIS_c **

$SumCWT_c = SumCWT_c + (q_c * K_n - \sum_{q=1}^{q_c} D_c^q)$ **Store info for stat. colc. of CWT_c **

}

}

$U = SumT_n/D_N$ ** Computation of performance measures**

$TP = N/D_N$

for c = 1 to n

{

$TIS_c = SumTIS_c/(N*q_c)$

$CWT_c = SumCWT_c/(N*q_c)$

}

** Queue lengths can be calculated using Little's law **

FAST SIMULATION OF A SINGLE SERVER ASSEMBLY QUEUEING SYSTEM WITH FINITE KIT BUFFER LIMIT

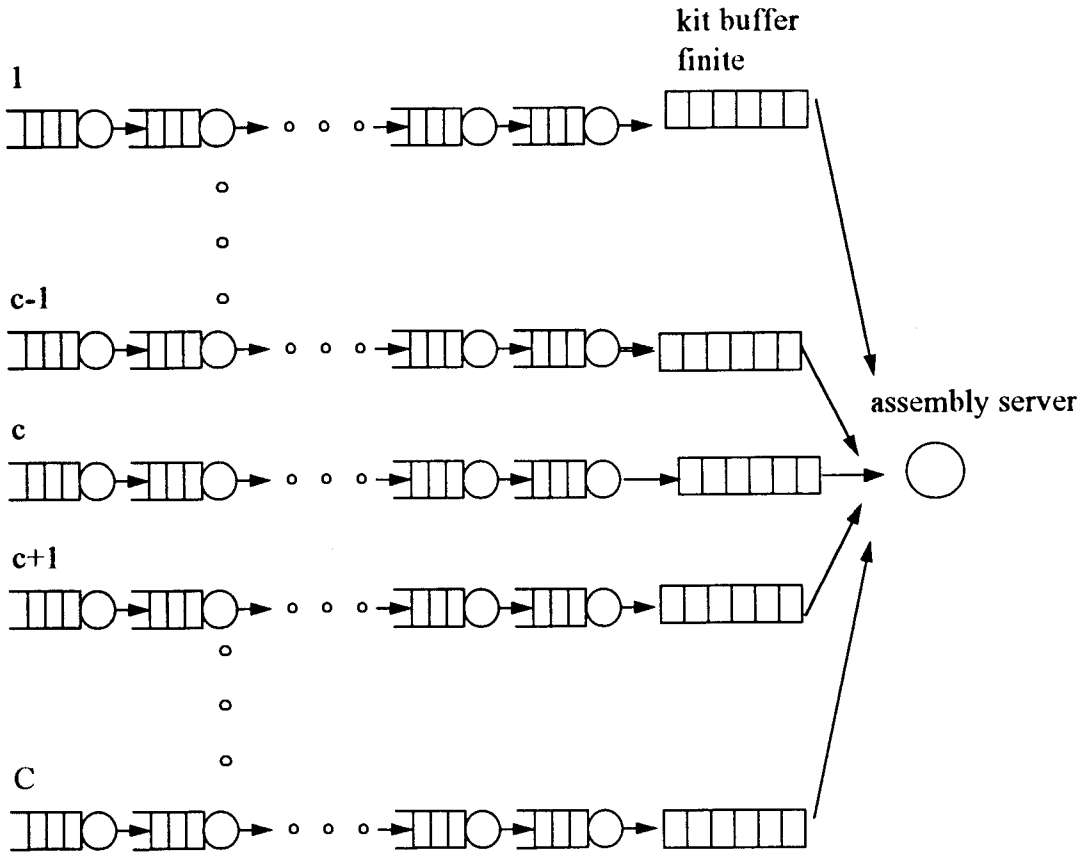


Figure 1.3. A Typical Single Assembly Queueing System with Finite Buffers

Description of the System

The system consists of C component types coming through C separate tandem lines. Each of these components go into separate kit buffers before the assembly node. If q_c units of each type c are available in the component buffers, they are assembled. If the kit buffer is full, the components wait in the tandem lines themselves, thus blocking any service on the last machines of the tandem lines.

Assumptions

1. There are no separate set up times for the assembly server.
2. The kits are assembled on an FCFS basis.

Nomenclature

n = Index for assemblies.

S_n = Service start time for assembly n.

K_n = Time kit n is formed.

T_n = Assembly time for assembly n.

D_n = Departure time of assembly n.

N = Number of assemblies which are to be simulated.

DT_c^q = Departure time of q^{th} unit from the last machine of feeder line of component c.

c = Index for component type.

C = Number of components types.

q_c = Number of units of component c going into a single assembly unit.

U = Utilization of assembly server.

TP = Throughput of the assembly server.

TIS_c = Average time in system of component c.

CWT_c = Average waiting time of component c before kitting.

B_c = Size of buffer before assembly server, for component c.

E_c^q = The service end time of unit q on last machine of tandem line c.

$Q_c[B_c]$ = The queue removal time array having size, B_c.

The inputs are T_n, N, C, B_c, and q_c.

Recursive Relationships used for Simulation

$DT_c^q = \max \{E_c^q, Q_c[B_c]\}$

The departure time of unit q of component c, from last machine of tandem line c is the maximum of its service end time and the departure time of previous customer from last space in the buffer B_c (as recorded in the queue removal time history array, $Q_c[B_c]$).

$K_n = \max_c (DT_c^q c)$

The kit formation time for kit n, is the maximum of these departure times of q_c^{th} units of corresponding component 'c's.

$S_n = \max \{ K_n, D_{n-1}\}$

The service start time of assembly n is the maximum of the departure time of previous assembly from assembly server and the kit formation time.

$D_n = S_n + T_n$

The departure time of assembly n is the sum of service start time S_n and the assembly time T_n.

Performance Measures of Interest

$U = \sum_{n=1}^{N} T_n / D_N$

$TP = N/D_N$

$TIS_c = \sum_c SumTIS_c/(N*q_c)$

$CWT_c = \sum_c SumCWT_c/(N*q_c)$

Algorithm for Implementation

for n = 1 to N ** Beginning of simulation of N assemblies **

{ for c = 1 to C** Process all component types through their respective feeder lines **

{

for q = 1 to q_c

{

Process a unit of type c through feeder line of c using the fast simulation model of the tandem line and store departure time of that unit from last machine of the line, as DT_c^q

$DT_c^q = \max \{E_c^q, Q_c[B_c]\}$ ** Determining the dept. time of unit q from last machine of tandem lines **

}

if $DT_c^q c > DT_{c-1}^q c-1$

$K_n = DT_c^q c$ ** Storing the max. arrival time as the time an assembly kit n is formed **

}

$S_n = \max \{ K_n, D_{n-1}\}$ ** To compute the assembly start time for assembly i **

$D_n = S_n + T_n$ ** To compute the departure time for assembly i **

$SumT_n = SumT_n + T_n$ ** To store the sum of all assm. times for stat. colc. **

for c = 1 to n

{ Update $Q_c[B_c]$ based on S_n

$SumTIS_c = SumTIS_c + (q_c * D_n - \sum_{q=1}^{q_c} DT_c^q)$ ** To store info needed for stat. colc. of TIS_c **

$$SumCWT_c = SumCWT_c + (q_c * K_n - \sum_{q=1}^{q_c} DT_c^q)$$

** To store info needed for stat. colc. of CWT_c **

}

}

$U = SumT_n / D_N$ ** Computation of performance measures**

$TP = N / D_N$

for c = 1 to n

{

$TIS_c = SumTIS_c/(N*q_c)$

$CWT_c = SumCWT_c/(N*q_c)$

}

** Queue lengths can be calculated using Little's law **

FAST SIMULATION OF MERGE TOPOLOGY

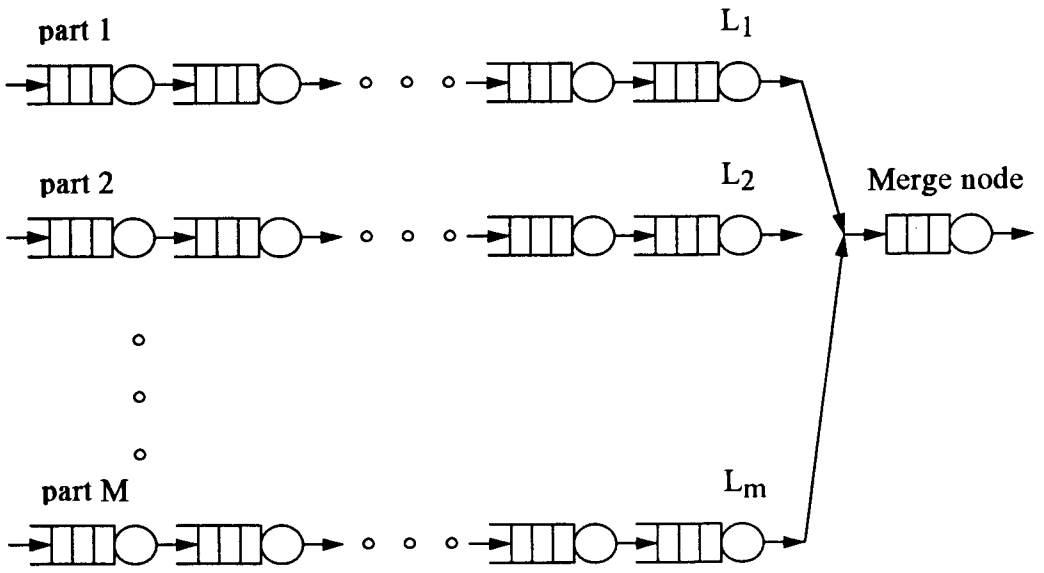


Figure 1.4. A Merge Topology

Description of the System

In this system, there are M separate tandem lines, which process M different parts, merging at a merge node. The merge node processes parts in the order of their completion times at the various tandem lines.

Nomenclature

m	=	Index of tandem line.
M	=	Index of the last tandem line (or) number of tandem lines.
E_m	=	Service end time of part on last machine of line m.
N	=	Number of parts to be completed at the merge node.
m^*	=	$\arg\min \{E_1, E_2, \ldots, E_M\}$
DT_{m^*}	=	Departure time of part from the last machine of merging line.
D_n	=	Departure time of part n from merge node.
B	=	Buffer size of merge node.

T_n = Processing time at merge node.

U = Utilization of merge node.

BT = Blocking time before merge node.

TP = Throughput of merge node.

QWT = Queue waiting time before merge node.

The inputs are T_n, M, N, and B.

Recursive Relationship used for Simulation

$$m^* = \underset{\min}{\arg} \{E_1, E_2, \ldots, E_M\}$$

The merging line is the one corresponding to the minimum service end time among E_ms.

$$DT_{m^*} = \max\{E_{m^*}, D_{n-(B+1)}\}$$

The departure time of a part n from the last machine of the merging line is the maximum of the service end time of that part and the departure time of 'n-(B+1)'th part from the merging node.

$$S_n = \max\{DT_{m^*}, D_{n-1}\}$$

The service start time is the maximum of the departure time of the part from the last machine of the merging line and the departure time of the previous part from the merge node.

$$D_n = S_n + T_n$$

The departure time of part from merge node is the sum of the service start time and the processing time T_n.

Determine next $m = m^*$ and the corresponding E_{m^*}.

Algorithm for Implementation

for m = 1 to M

do

{

Determine E_m after processing a single part through each of the lines

}

for n = 1 to N

do

{

$E_{m*} = \min (E_m)$ for $\forall$ m

$DT_{m*} = \max (E_{m*}, D_{n-(B+1)})$

$S_n = \max\{DT_{m*}, D_{n-1}\}$

$SumBT = SumBT + (DT_{m*} - E_{m*})$

** To add up the values of blocking times for statistics collection **

$SumQWT = SumQWT + (S_n - DT_{m*})$

** To add up the values of queue waiting times
in front of the merge node for QWT stat. colc. **

Generate processing time T_n

$D_n = S_n + T_n$

$SumT_n = SumT_n + T_n$

** To add up the values of processing times at merge node
to compute the utilization of merge node **

Determine next E_m for m = m*

** Simulate the processing of a part thro' the
line which is the present merge line **

}

** Computation of performance measures **

$U = SumT_n/D_N$

$BT = SumBT/N$

$TP = N/D_N$

$QWT = SumQWT/N$

** Queue lengths can be calculated by Little's law **

FAST SIMULATION OF PARALLEL STATION TOPOLOGY

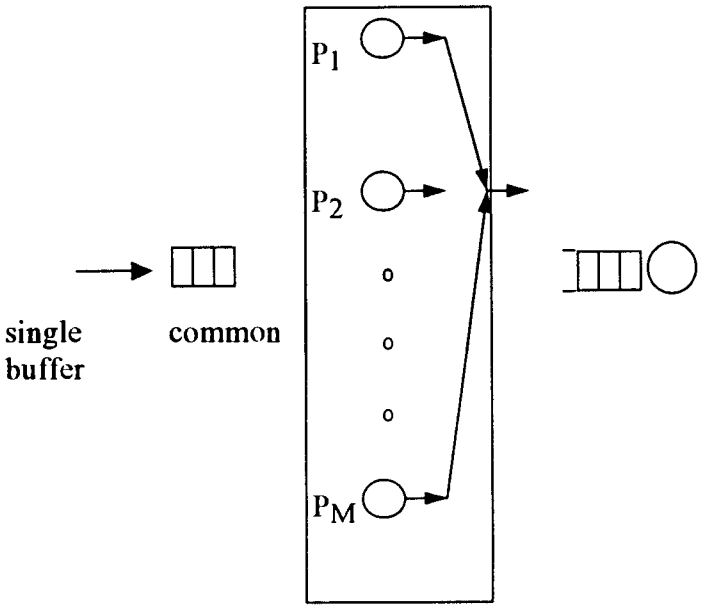


Figure 1.5. A Parallel Server Workstation

Description of the System

In this case, there are M separate servers in a parallel server worstation. The parallel server workstation takes in customers and the customer which finishes earliest in any of the servers in the station is released from the station.

Assumptions

1. There is a single common buffer associated with all the servers in the parallel server.

Nomenclature

m	=	Index of the server in the parallel server workstation.
M	=	Index of the last server in the parallel server workstation.
E_m	=	Service end time of a part in a server m.
N	=	Number of parts to be completed at the parallel server workstaion.
m^*	=	$\arg\min \{E_1, E_2, \ldots, E_M\}$

DT_{m^*} = Departure time of part from the parallel server workstation.

B = Buffer size of the single common buffer in front of the parallel server workstation.

S_m = Service start time at server m.

T_m = Processing time at server m.

U = Utilization of parallel server workstation.

TP = Throughput of parallel server workstation.

QWT = Queue waiting time before the parallel server workstation.

The inputs are A_n, T_m, M, N, and B.

Recursive Relationship used for Simulation

$$m^* = \arg\min \{E_1, E_2, \ldots, E_M\}$$

The customer to be released is the one corresponding to the minimum service end time among E_ms.

$DT_{m^*} = E_{m^*}$

The departure time of a customer from the server m* is the same as the service end time at that server, when the forthcoming station has an infinite buffer.

Determine next m = m* and the corresponding E_{m^*}.

Algorithm for Implementation

```
for m = 1 to M
do
{
        Generate arrival time An

        Sm = An        ** The first service start time on any of the servers is the arrival
                                time of customers **

        Em = Sm + Tm
```

Determine E_m after processing a single part through each of the servers

$SumT_n = SumT_n + T_m$

$SumT_n$ is a statistics collection term used to aggregate statistics for the utilization of the parallel server station

}

for n = 1 to N

do

{

$E_{m*} = \min (E_m)$ for $\forall$ m

$DT_{m*} = E_{m*}$

Generate arrival time A_n

$S_{m=m*} = \max\{DT_{m=m*}, A_n\}$

$E_{m=m*} = S_{m=m*} + T_m$

$SumT_n = SumT_n + T_m$

$SumQWT = SumQWT + (S_n - Dt_{m*})$

Add up the values of queue waiting times

S_n is the start time of any job whose queue time is collected

}

** Computation of performance measures **

$U = SumT_n/D_N$

$TP = N/D_N$

$QWT = SumQWT/N$

** Queue lengths can be calculated by Little's law **

APPENDIX II
(Source code for simulating the experimental prototype)

```
//Init.h -initializes variables and seeds used;
void Initialize()
{ for (int level =1; level<=8; level++)        //initializing variables for 8 levels, 3 tandem lines in each
                                               //level and 6 workstations in each tandem line
    { for (int prsntTL =1; prsntTL <=3; prsntTL++)        //prsntTL = present tandem line
        {for (int prsntWS =1; prsntWS <=6; prsntWS++)     //prsntWS = present workstation
            {
              b[level][prsntTL][prsntWS]=0;      //buffer size of a server + space in the server
              s[level][prsntTL][prsntWS]=0.0;            //service time at a server
              e[level][prsntTL][prsntWS]=0.0;            //service end time at a server
              d[level][prsntTL][prsntWS]=0.0;            //departure time at a server
              PrevD[level][prsntTL][prsntWS]=0.0;        //previous departure time at a server
              Wq[level][prsntTL][prsntWS]=0.0;           //quantity collecting queueing times
              Ws[level][prsntTL][prsntWS]=0.0;           //quantity collecting service times
              Wb[level][prsntTL][prsntWS]=0.0;           //quantity collecting blocking times
              Wi[level][prsntTL][prsntWS]=0.0;           //quantity collecting idle times
              nt[level][prsntTL][prsntWS]=0.0;           //quantity collecting node times
             w[level][prsntTL][prsntWS].QLength=0;       //buffer size of a server
             w[level][prsntTL][prsntWS].SvcTMean=0.0;//service time mean
            }
          LossCount[level][prsntTL]=0;                   //loss count at a tandem line
          INTARRT[level][prsntTL]=0.0;                   //interarrival time into a tandem line
          NoOfWS[level][prsntTL]=0;                      //# of workstations in a tandem line
          partnow[level][prsntTL]=1;                     //present customer in a tandem line
          TotalLife[level][prsntTL]=0.0;                 //total time spent in a tandem line by a customer
          TP[level][prsntTL]=0.0;                        //through put in a tandem line
          ArrGenCount[level][prsntTL]=0;                 //# of arrivals generated into a tandem line
        }
       kt[level]=0.0;                                    //kit time at an assembly server
    }
  SerGenCount=0;                                         //# of service times generated
GenerateSeeds();                                         //Generate seeds for random # generation
}


void GenerateSeeds()
{for (int level =1; level<=8; level++)
    {for (int prsntTL =1; prsntTL <=3; prsntTL++)
      {for (int prsntWS =0; prsntWS <=6; prsntWS++)
        {
         initSeed[level][prsntTL][prsntWS]=123457;       //initial seed for all generators
         seed0=short(1+RandU(initSeed[level][prsntTL][prsntWS])*(32766));
         seed[level][prsntTL][prsntWS]=long(1+GetSeed(seed0)*(MAXLONGINT-1));
        }
      }
    }
}

double GetSeed(short& sd) //16 bit generator
{
  double gSeed;
  sd=(sd*5997)+1;
  if(sd<0) sd=sd+32767+1;
  gSeed=sd*invMaxInt;
   return gSeed;
}
```

```
double RandU(long& sd)    //32 bit generator
{
  double Rand;
  const double a=16807,q=127773.0,r=2836.0;
  ldiv_t dSeed;
  dSeed=ldiv(sd,(long)q);
  sd=(long)(a*(sd-(long)(dSeed.quot*q)))-(long)(dSeed.quot*r);
  if(sd<0)
    sd=sd+MAXLONGINT;
  Rand=sd*ONEBYMAXLONGINT;
  return Rand;
}
```

//Routnoif.h - This file contains all simulation routines needed for random number generation, //simulation of various topologies, and statistics collection

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <malloc.h>
#include <assert.h>
#include <time.h>

#define TOTWS  10              //can model 9 w/s
#define TOTTL  4               //can model 3 tandem lines in each level
#define LEVEL  9               //can model 8 levels

typedef struct DT *DTPtrType;      //holds depart.time of a part and an address to
                                   //next part record
typedef struct DT DTRecord;
//used to generate the circular link list which holds the previous buffer-relevant customers
struct DT { double     dt;
        DTPtrType  next;
          };

typedef struct asma *asmaType;

struct asma { DTPtrType LL[TOTWS];     //get a circular linked list in 2 dimension to hold the
                                       //departure
              DTPtrType dptr;          //times at various workstations
              DTPtrType ptrend;
            };

typedef struct asma asmaRecord;
asmaType preA[LEVEL][TOTTL];//2 dimensional array to hold the circular linked list mentioned
                            //in struct asma
typedef struct WS WSRecordType;

struct WS                          //struct that holds workstation information
       {
        int QLength;               //buffer size of server
        double SvcTMean;           //mean service time of server
       };
```

```
WSRecordType w[LEVEL][TOTTL][TOTWS];      //contains workstation information

int    b[LEVEL][TOTTL][TOTWS] ;           //buffer size + 1; for establishing circular link list
double s[LEVEL][TOTTL][TOTWS] ;           //service time of workstation
double e[LEVEL][TOTTL][TOTWS] ;           //service end time at workstation
double d[LEVEL][TOTTL][TOTWS] ;           //departure time from workstation
double PrevD[LEVEL][TOTTL][TOTWS] ;       //previous departure time at workstation

double Wq[LEVEL][TOTTL][TOTWS] ;          //statistics collection variable for queueing times
double Ws[LEVEL][TOTTL][TOTWS] ;          //statistics collection variable for service times
double Wb[LEVEL][TOTTL][TOTWS] ;          //statistics collection variable for blocking times
double Wi[LEVEL][TOTTL][TOTWS] ;          //statistics collection variable for idle times
double nt[LEVEL][TOTTL][TOTWS] ;          //statistics collection variable for time at node
long seed[LEVEL][TOTTL][TOTWS];           //seed for random number generator

long  LossCount[LEVEL][TOTTL];          //loss count at a tandem line
double INTARRT[LEVEL][TOTTL];           //inter-arrival time of parts coming from outside system
int   NoOfWS[LEVEL][TOTTL];             //# of workstations in a tandem line
long  partnow[LEVEL][TOTTL];            //the present customer being simulated

double TotalLife[LEVEL][TOTTL];         //total time a customer spends in a tandem line
double      TP[LEVEL][TOTTL];           //thro' put of a line

void AllInputs();                       //reads system parameter input from input file
void Input(int level,int lineIndex);    //reads input file for each line
void GenerateDepartureTimeLinkList(int level, int prsntTL,int prsntWS); //generates link list
void Initialize();                      //initializes variables
void fillup();                          //fills up network so toggles may be set going
void Simulate();                        //simulates the  network
void AllOutputs();                      //prints simulation output to output files
void Output(int level, int lineIndex);  //prints sim. o/p to output file for a single line
double max(double p,double q);          //computes max of two quantities

char OutputFile[13];                    //takes in output file name
char InputFile[13];                     //takes in input file name
double kt[LEVEL];                       //kit time at as assembly in any level

int level;                              //declares level identification
int prsntTL;                            //declares tandem line

void LLUpdate(int level,int prsntTL,int prsntWS);   //updates circular link list after a new dep.
void StatColct(int level,int prsntTL,int prsntWS);  //collects statistics
void Service(int level,int prsntTL,int prsntWS);    //generates service time
void Arrival(int level,int prsntTL);                //generates arrivals
void InSerDep(int level,int prsntTL,int prsntWS);   //simulates a dep. at a inner server in a line
void EndOfServiceAtEndNode(int level,int prsntTL,int prsntWS);   // computes end of service at
                                                                 //end node in a line
void BackEndNodeDep(int level,int prsntTL,int BackprsntTL);      //computes the departure time
                                                                 //from the prev. end node
void BackEndNodeUpdate(int level,int prsntTL,int BackprsntTL);   //updates the prev. end node's
                                                                 //circular link list
void BackEndNodeStatColct(int level,int prsntTL,int BackprsntTL); //collects stat.s for prev.
                                                                  //end node
void ParaBackEndNodeDep(int level,int prsntTL,int BackprsntTL);  //computes the departure time
                                                                 //from a previous parallel station
void ParaBackEndNodeUpdate(int level,int prsntTL,int BackprsntTL); //updates the link list of
```

```
                                                        //parallel station
void ParaBackEndNodeStatColct(int level,int prsntTL,int BackprsntTL);      //collects statistics
                                                        //for parallel station
void InTand(int level, int prsntTL,int BackprsntTL );      //simulates a tandem line which gets parts
                                                        //from other workstations
void OutTand(int level, int prsntTL);      //simulates a tandem line that gets parts
                                           //that gets parts from outside system
void Assem(int level);                     //simulates assembly

long ArrGenCount[LEVEL][TOTTL];            //to count the # of times expo gen. called for the sake of
                                           // getting arr.s
long SerGenCount; //to count the # of times rand # gen. called for the sake of getting serv. time

//begin procedure for random number generation
 long MAXLONGINT= (long)(pow(2,31)-1);                  //# used for random # generation
double ONEBYMAXLONGINT=(1.0/(pow(2,31)-1.0));           //# used for random # generation
double invMaxInt=1/32767.0;                             //# used for random # generation

double expo(double Mean, int level, int prsntTL, int prsntWS);      //generates exponential random #
void GenerateSeeds();                                   //generates seed to generate random #
double GetSeed(short& sd);                              //functions to get initial seed
double RandU(long& sd);                                 //function to generate random #
short seed0;
long initSeed[LEVEL][TOTTL][TOTWS];                     //initial seed input

double expo(double Mean, int level, int prsntTL, int prsntWS)      //exponential random # generator
{
  double Prob,expo;
  Prob = RandU(seed[level][prsntTL][prsntWS]);
  if(Prob<=ONEBYMAXLONGINT) Prob=RandU(seed[level][prsntTL][prsntWS]);
  expo = (-Mean)*log(Prob);
  return expo;
}
//end procedure for random number generation

double max(double p,double q)              //the cmp function
  {
    if (p>q) q=p;
    return(q);
  }

void LLUpdate(int level,int prsntTL,int i) //linklist update for an inner server in a line
{
(preA[level][prsntTL]->LL[i])->dt=d[level][prsntTL][i];
preA[level][prsntTL]->LL[i]=(preA[level][prsntTL]->LL[i])->next;
}

void StatColct(int level,int prsntTL,int i)   //stat colc. (blocking, queueing and idle time) for an
                                              //inner server
{
Wi[level][prsntTL][i]+=max(0,d[level][prsntTL][i-1]-PrevD[level][prsntTL][i]);
Wb[level][prsntTL][i]+=max(0,d[level][prsntTL][i]-e[level][prsntTL][i]);
Wq[level][prsntTL][i]+=max(0,PrevD[level][prsntTL][i]-d[level][prsntTL][i-1]);
}

void Service(int level,int prsntTL,int i)  //service time generation and service stat colc.
```

```
{
s[level][prsntTL][i]=expo(w[level][prsntTL][i].SvcTMean, level, prsntTL, i);
Ws[level][prsntTL][i]+=s[level][prsntTL][i];
++SerGenCount;
}

void Arrival(int level,int prsntTL)  //arrival generation for outTands and loss count colc.
{
      d[level][prsntTL][0]=d[level][prsntTL][0]+expo(INTARRT[level][prsntTL],level,prsntTL,0);
      ++ArrGenCount[level][prsntTL];
while (d[level][prsntTL][0]<(preA[level][prsntTL]->LL[1])->dt)
          {
d[level][prsntTL][0]=d[level][prsntTL][0]+expo(INTARRT[level][prsntTL],level,prsntTL,0);
            ++LossCount[level][prsntTL];
            ++ArrGenCount[level][prsntTL];
          }
}
void InSerDep (int level,int prsntTL,int i) //inner server departure calculation routine
{
 PrevD[level][prsntTL][i]=d[level][prsntTL][i];
 e[level][prsntTL][i]= max(d[level][prsntTL][i-1],d[level][prsntTL][i])+s[level][prsntTL][i];
 d[level][prsntTL][i]=max(e[level][prsntTL][i],(preA[level][prsntTL]->LL[i+1])->dt);
 nt[level][prsntTL][i]+=d[level][prsntTL][i]-d[level][prsntTL][i-1];
}

void EndOfServiceAtEndNode(int level,int prsntTL,int I)      //can't get the departure time for
 {                                                           //end node..so get end time
  Service(level,prsntTL,i);
  e[level][prsntTL][i]=max(d[level][prsntTL][i-1],d[level][prsntTL][i])+s[level][prsntTL][i];
 }

void BackEndNodeDep(int level,int prsntTL,int BackprsntTL) //gets departure time from end node
{                                                           //of previous topology
PrevD[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]=
d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]];
d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]] =
max(e[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]],(preA[level][prsntTL]->LL[1])->dt);
++partnow[level-1][BackprsntTL];
nt[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]+=
d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]-d[level-1][BackprsntTL][NoOfWS[level-
                                                                    1][BackprsntTL]-1];
}

void BackEndNodeUpdate(int level, int prsntTL, int BackprsntTL)    //updates linked list of end node
{                                                                   // of previous level
(preA[level-1][BackprsntTL]->LL[NoOfWS[level-1][BackprsntTL]])->dt=
d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]];
preA[level-1][BackprsntTL]->LL[NoOfWS[level-1][BackprsntTL]]=
(preA[level-1][BackprsntTL]->LL[NoOfWS[level-1][BackprsntTL]])->next;
}

void BackEndNodeStatColct(int level,int prsntTL,int BackprsntTL)   //stat colc.(blocking and)
{                                                                  // queueing for an inner server
Wi[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]+=
max(0,d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]-1]-PrevD[level-
                                    1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]);
```

```
Wb[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]+=
max(0,d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]-e[level-1][BackprsntTL]
                                                    [NoOfWS[level-1][BackprsntTL]]);
Wq[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]+=
max(0,PrevD[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]]-d[level-
                                        1][BackprsntTL][NoOfWS[level-1][BackprsntTL]-1]);
}

void ParaBackEndNodeDep(int level,int prsntTL,int BackprsntTL)    //function that gets departure
{                                                                  // from parallel server
PrevD[level-1][BackprsntTL][1]=d[level-1][BackprsntTL][1];
d[level-1][BackprsntTL][1]=max(e[level-1][BackprsntTL][1],(preA[level][prsntTL]->LL[1])->dt);
++partnow[level-1][BackprsntTL];
}

void ParaBackEndNodeUpdate(int level, int prsntTL, int BackprsntTL)        //updates link list of
{                                                                          //parallel server
(preA[level-1][1]->LL[1])->dt=d[level-1][1][1];
preA[level-1][1]->LL[1]=(preA[level-1][1]->LL[1])->next;
}

void ParaBackEndNodeStatColct(int level,int prsntTL,int BackprsntTL)       //stat colc. (blocking for
{                                                                          // second parallel server)
Wi[level-1][1][1]+=max(0,d[level-1][1][0]-PrevD[level-1][BackprsntTL][1]);
Wb[level-1][BackprsntTL][1]+=max(0,(preA[level][prsntTL]->LL[1])->dt-e[level-1][BackprsntTL][1]);
Wq[level-1][1][1]+=max(0,PrevD[level-1][BackprsntTL][1]-d[level-1][1][0]);
}

void InTand(int level, int prsntTL, int BackprsntTL)          //simulates an inner tandem line
{                                                             //can also act as a merge line
        BackEndNodeDep(level,prsntTL,BackprsntTL);
        BackEndNodeUpdate(level, prsntTL, BackprsntTL);
        BackEndNodeStatColct(level, prsntTL, BackprsntTL);
        d[level][prsntTL][0] = d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]];
       for (int i=1;i<NoOfWS[level][prsntTL]; i++)
        {
          Service(level,prsntTL,i);
          InSerDep(level,prsntTL,i);     //inner server departure
          LLUpdate(level,prsntTL,i);
          StatColct(level,prsntTL,i);
        }
        EndOfServiceAtEndNode(level,prsntTL,i);
 }

void OutTand(int level, int prsntTL)          //simulates a tandem line into which arrivals take place
{
 Arrival(level, prsntTL);
 for (int i=1;i<NoOfWS[level][prsntTL]; i++)
  {
    Service(level,prsntTL,i);
    InSerDep(level,prsntTL,i);     //inner server departure
    LLUpdate(level,prsntTL,i);
   StatColct(level,prsntTL,i);
  }
    EndOfServiceAtEndNode(level,prsntTL,i);
}
```

```
void Assem(int level)                    //simulates an assembly server
{
  for (int prsntTL =1; prsntTL <=2; prsntTL++)
     {
      BackEndNodeDep(level, prsntTL, prsntTL);
      BackEndNodeUpdate(level,prsntTL, prsntTL);
      BackEndNodeStatColct(level, prsntTL, prsntTL);
      d[level][prsntTL][0]= d[level-1][prsntTL][NoOfWS[level-1][prsntTL]];
     }
    kt[level]=max(d[level][1][0],d[level][2][0]);      //kit time computation
    Service(level,1,1);
     e[level][1][1]=max(kt[level],d[level][1][1])+s[level][1][1];
     PrevD[level][2][1]= d[level][2][1];
     d[level][2][1]=max(e[level][1][1],preA[level+1][1]->LL[1]->dt);
     LLUpdate(level,2,1);
    Wq[level][2][1]+=max(0,PrevD[level][2][1]-d[level][2][0]);
    ++partnow[level][2];
}
void Merge(int level, int prsntTL, int BackprsntTL)          //simulates a lone merge node
{
      BackEndNodeDep(level, prsntTL, BackprsntTL);
      BackEndNodeUpdate(level,prsntTL, BackprsntTL);
      BackEndNodeStatColct(level, prsntTL, BackprsntTL);
  d[level][prsntTL][0]=d[level-1][prsntTL][NoOfWS[level-1][prsntTL]];
  Service(level,prsntTL,1);
  e[level][prsntTL][1]=max(d[level][prsntTL][0],d[level][prsntTL][1])+s[level][prsntTL][1];
}

void Parallel(int level,int prsntTL, int BackprsntTL)        //simulates a parallel server workstation
{
      BackEndNodeDep(level, 1, BackprsntTL);
      BackEndNodeUpdate(level, 1, BackprsntTL);
      BackEndNodeStatColct(level, 1, BackprsntTL);
  d[level][1][0]=d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]];
  Service(level,prsntTL,1);
  e[level][prsntTL][1]=max(d[level][1][0],d[level][prsntTL][1])+s[level][prsntTL][1];
}

void NextToPara(int level, int prsntTL, int BackprsntTL)//simulates a server next to the parallel
{                                                        //server and updates linked list of parallel server
     ParaBackEndNodeDep(level, prsntTL, BackprsntTL);
     ParaBackEndNodeUpdate(level, prsntTL, BackprsntTL);
     ParaBackEndNodeStatColct(level, prsntTL, BackprsntTL);
  d[level][prsntTL][0]=d[level-1][BackprsntTL][NoOfWS[level-1][BackprsntTL]];
  Service(level,prsntTL,1);
  e[level][prsntTL][1]=max(d[level][prsntTL][0],d[level][prsntTL][1])+s[level][prsntTL][1];
}
```

```
//AllInput1.h - reads files containing system parameter inputs
void AllInputs()  //takes in the input names of files that contain information about the various
{                 //topologies
for (int level =1; level <=2; level++)
   {
    for (int prsntTL=1; prsntTL<=2; prsntTL++)
    {
     cout<<"give the i/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
     cin>>InputFile;
     Input(level,prsntTL);
    }
   }
 for ( level =3; level <=3; level++)
   {
    for (int prsntTL=1; prsntTL<=2; prsntTL++)
    {
     cout<<"give the i/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
     cin>>InputFile;
     Input(level,prsntTL);
    }
   }
 for ( level =4; level <=4; level++)
   {
    for (int prsntTL=1; prsntTL<=1; prsntTL++)
    {
     cout<<"give the i/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
     cin>>InputFile;
     Input(level,prsntTL);
    }
   }
 for ( level =5; level <=5; level++)
   {
    for (int prsntTL=1; prsntTL<=2; prsntTL++)
    {
     cout<<"give the i/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
     cin>>InputFile;
     Input(level,prsntTL);
    }
   }
 for ( level =6; level <=7; level++)
   {
    for (int prsntTL=1; prsntTL<=1; prsntTL++)
    {
     cout<<"give the i/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
     cin>>InputFile;
     Input(level,prsntTL);
    }
   }
}

void Input(int level,int prsntTL)    //takes in data for a  topology at a given level
{
 ifstream inFile(InputFile,ios::in);
 assert(inFile!=0);
 int dummy;
 inFile>>INTARRT[level][prsntTL];
```

```
 inFile>>NoOfWS[level][prsntTL];
 preA[level][prsntTL]=(asmaType)malloc(sizeof(asmaRecord));      //mallocs memory for asmaRecord
 for(int i=1;i<=NoOfWS[level][prsntTL];i++)
    {
     inFile>>dummy;
     inFile>>w[level][prsntTL][i].QLength;
     inFile>>w[level][prsntTL][i].SvcTMean;
     b[level][prsntTL][i]=w[level][prsntTL][i].QLength + 1;
     GenerateDepartureTimeLinkList(level, prsntTL, i);
    }
}

//generates circular linked  list which will hold the preceding buffer-relevant customers
void GenerateDepartureTimeLinkList(int level, int prsntTL, int prsntWS)
  {
   preA[level][prsntTL]->LL[prsntWS]=(DTPtrType)malloc(sizeof(DTRecord));

   (preA[level][prsntTL]->LL[prsntWS])->dt=(double)0;
   preA[level][prsntTL]->ptrend=preA[level][prsntTL]->LL[prsntWS];
   for (int j=2;j<=b[level][prsntTL][prsntWS];j++)
    {  preA[level][prsntTL]->dptr=(DTPtrType)malloc(sizeof(DTRecord));
       (preA[level][prsntTL]->dptr)->dt=(double)0;
       (preA[level][prsntTL]->ptrend)->next=preA[level][prsntTL]->dptr;
       preA[level][prsntTL]->ptrend=(preA[level][prsntTL]->ptrend)->next;
    }
   (preA[level][prsntTL]->dptr)->next=preA[level][prsntTL]->LL[prsntWS];
  }
```

```
//AlOtpts1.h -calls the appropriate output file for the various workstations
void AllOutputs() //gets the output file names where the system performance measures are printed
{
  for (int level =1; level <=2; level++)
    {
    for (int prsntTL=1; prsntTL<=2; prsntTL++)
    {
     cout<<"give the o/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
     cin>>OutputFile;
     Output(level,prsntTL);
    }
   }
  for ( level =3; level <=3; level++)
    {
    for (int prsntTL=1; prsntTL<=2; prsntTL++)
    {
     cout<<"give the o/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
     cin>>OutputFile;
     Output(level,prsntTL);
    }
   }
 for ( level =4; level <=4; level++)
   {
   for (int prsntTL=1; prsntTL<=1; prsntTL++)
   {
    cout<<"give the o/p file name for level "<<level<<" and for prsntTL "<<prsntTL<< " : ";
    cin>>OutputFile;
    Output(level,prsntTL);
   }
  }
 for (level =5; level <=5; level++)
    {
    for (int prsntTL=1; prsntTL<=2; prsntTL++)
    {
     cout<<"give the o/p file name for level "<<level<<" and for prnstTL "<<prsntTL<< " : ";
     cin>>OutputFile;
     Output(level, prsntTL);
    }
   }
 for ( level =6; level <=6; level++)
    {
    for (int prsntTL=1; prsntTL<=1; prsntTL++)
    {
     cout<<"give the o/p file name for level "<<level<<" and for prnstTL "<<prsntTL<< " : ";
     cin>>OutputFile;
     Output(level, prsntTL);
    }
   }
}

void Output(int level, int prsntTL) //calculates the sim. o/p measures and prints it to files
{
TP[level][prsntTL]=(double)partnow[level][prsntTL]/d[level][prsntTL][NoOfWS[level][prsntTL]];
//calculates throughput of a line
for (int i=1;i<=NoOfWS[level][prsntTL];i++)
TotalLife[level][prsntTL] =
```

```
TotalLife[level][prsntTL]+Ws[level][prsntTL][i]+Wb[level][prsntTL][i]+Wq[level][prsntTL][i];
ofstream outF(OutputFile,ios::app);
assert(outF!=0);
outF<<"The Fast Simulation Report of tandem line "<<prsntTL<<"in level "<<level<<endl;
  outF<<"-------------------------------------------------------\n"<<endl;
  for ( i=1; i<=NoOfWS[level][prsntTL]; i++)
  {
   outF<<"For w/s "<<i<<endl;
   outF<<"----------"<<endl;
   outF<<"The queue capacity is :"<<w[level][prsntTL][i].QLength<<endl;
  outF<<"The ave. queueing time is :"<< Wq[level][prsntTL][i]/(double)(partnow[level][prsntTL])
                              <<endl;
   outF<<"The ave. queueing length is:"<<
         Wq[level][prsntTL][i]*TP[level][prsntTL]/(double)(partnow[level][prsntTL])<<endl;
   outF<<"The ave. service time is :"<<Ws[level][prsntTL][i]/(double)(partnow[level][prsntTL])
                              <<endl;
   outF<<"The ave. idle time is :"<<Wi[level][prsntTL][i]/(double)(partnow[level][prsntTL])<<endl;
   outF<<"The ave. node time is :"<<nt[level][prsntTL][i]/(double)(partnow[level][prsntTL])<<endl;
   outF<<"The ave. blocking time is :"<<Wb[level][prsntTL][i]/(double)(partnow[level][prsntTL])
                              <<endl;
   outF<<"The utilization of node is :"<<
         Ws[level][prsntTL][i]/d[level][prsntTL][NoOfWS[level][prsntTL]]<<endl;
  }
   outF<<"------------------------------"<<endl;
   outF<<"The %loss in the tandem line "<<prsntTL<<" is :"
         <<(double)(LossCount[level][prsntTL]*100)/(double)partnow[6][1]<<endl;
   outF<<"The thro' put is :"<<TP[level][prsntTL]<<endl;
   outF<<"Time In System is :"<<TotalLife[level][prsntTL]/(double)(partnow[level][prsntTL])
                                    <<endl;
   outF<<"Simulation end time is :"<<d[level][prsntTL][NoOfWS[level][prsntTL]]<<endl;
   outF<<"The Partnow is :"<<partnow[level][prsntTL]<<endl;
   outF<<"\n"<<endl;
   outF.close();
}
```

```
//The main program that simulates the experimental scenario

#include "routnoif.h"      //contains all functions that when put together will simulate scenario
#include "init.h"          //initializing all variables
#include "AllInpts1.h"     //contains functions for inputting system parameters
#include "AlOtpts1.h"      //contains functions for outputting system performance measures

double elapsed_time;       //to measure simulation run time
long start, finish;        //begin and end of simulation execution time

void MergeStream1();       //simulates one stream merging into the merge node
void MergeStream2();       //simulates second stream merging into the merge node
void FireMerge();          //gets the toggle in merge node going

void ParaStream1();        //simulates one server in parallel server station
void ParaStream2();        //simulates second server in parallel server station
void FirePara();           //gets the toggle in parallel server station going

void main()
{
 Initialize();             //function in init.h; initializes all variables
 AllInputs();              //function in AllInpts1.h; reads system parameters from input file
 fillup();                 //fills up the network so the toggles may be set
 time ( &start);           //marks begin of simulation execution time
 Simulate();               //simulates network
 time ( &finish);          //marks begin of simulation execution time
 elapsed_time = difftime(finish,start);      //the difference between begin and end of exe. time
cout<<"\n The time taken by the program to execute the sim. is = "<< elapsed_time<<endl;
AllOutputs();              //function in AlOtpts1.h; prints simulation output
}
void fillup()              //fills up the network so the toggles may be set
{
 MergeStream1();           //simulates tandem1.1 and tandem1.2 the assembly at level 2 and tandem3.1
 MergeStream2();           //simulates tandem3.2
 ParaStream1();            //simulates upto parallel server 1 -tandem5.1
 ParaStream2();            //simulates upto parallel server 2 -tandem5.2
 FirePara();               //gets a customer out of the parallel server workstation and releases it thro'
                           //tandem6.1 and out of the systm
}

void Simulate()                    //simulates network
{
while (partnow[6][1]<50000)        //simulate for 50000 parts
 {
 FirePara();
 }
}

void ParaStream1()         //simulates one server in parallel server station
{
  FireMerge();
  Parallel(5,1,1);
}

void ParaStream2()         //simulates second server in parallel server station
```

```
{
  FireMerge();
  Parallel(5,2,1);
}

void FirePara()              //gets the toggle in parallel server station going
{
 if (e[5][1][NoOfWS[5][1]] <= e[5][2][NoOfWS[5][2]]) //toggle based on serv. end times at parallel
                                                      //servers
    {
    NextToPara(6,1,1);
    ParaStream1();
    InTand(7,1,1);
    }
 else
    {
    NextToPara(6,1,2);  //gets departure from a server in the parallel server w/s
    ParaStream2();
    InTand(7,1,1);
    }
}

void MergeStream1()          //simulates one stream merging into the merge node
{
  OutTand(1,1);              //simulates tandem1.1
  OutTand(1,2);              //simulates tandem1.2

  Assem(2);                  //simulates assembly server at level 2.
  InTand(3,1,1);             //simulates tandem3.1
}

void MergeStream2()          //simulates second stream merging into the merge node
{
  OutTand(3,2);              //simulates tandem3.2
}

void FireMerge()             //gets the toggle in merge node going
{
  if (e[3][1][NoOfWS[3][1]] <= e[3][2][NoOfWS[3][2]])     //toggle based on service
    {                                                     //end time at stations before
    InTand(4,1,1);                                        //merge node
    MergeStream1();
    }
 else
    {
    InTand(4,1,2);
    MergeStream2();
    }
}
```

VITA

Ram Sreenivasan

Candidate for the Degree of

Master of Science

Thesis: FAST SIMULATION OF GENERAL MANUFACTURING NETWORKS

Major Field: Industrial Engineering and Management

Biographical:

Education: Graduated from Alpha Matriculation High School, Madras, India in May 1989; received Bachelor of Engineering degree in Mechanical & Production Engineering from Annamalai University, India in June 1993. Completed the requirements for the Master of Science degree in Industrial Engineering at Oklahoma State University in May1996.

Experience: Employed by Oklahoma State University, School of Industrial Engineering and Management as Graduate Research Assistant, and Graduate Teaching Assistant, 1994 to 1995.

Professional Memberships: Alpha Pi Mu, The National Industrial Engineering Honors Society.