

DISTRIBUTED SIMULATION FRAMEWORK BASED ON
LOAD BALANCED IMPLEMENTATION OF STANDARD CLOCK ALGORITHM
WITH WEB BASED EXTENSIONS

by

Osman Nuri Darcın

B.S. in CmpE., Boğaziçi University, 1987

M.S. in CmpE., Boğaziçi University, 1991



Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of

Doctor

of

Philosophy

Boğaziçi University

2003

ACKNOWLEDGMENTS

First of all, I would like to thank Professor Ali Rıza Kaylan for his great supervision and support throughout this thesis study. I admired his way of motivating me and without whom this thesis would never come to an end.

I want to thank all the members of my thesis committee for their interest in my study in the times of their heavy work load as well as for their valuable suggestions which improved the level of the dissertation.

Next, I want to express my deepest gratitude to my mother and my family for their morale support. Thus it is to them this thesis is dedicated.

The main bulk of the experimental work was carried out using the resources belonging to various departments of Boğaziçi University. I want to express my gratitude for kindly allowing me the use of their machines.

Last but not least, I want to thank to all members of MIS Department as well as my friends for their support and encouragement throughout this time.

ABSTRACT

DISTRIBUTED SIMULATION FRAMEWORK BASED ON LOAD BALANCED IMPLEMENTATION OF STANDARD CLOCK ALGORITHM WITH WEB BASED EXTENSIONS

Standard Clock approach is used to simulate a number of parametric variants of a single system. In this thesis, a simulation engine based on the distributed implementation of the Standard Clock approach on networks of heterogeneous UNIX workstations is developed. The objective is to examine the scalability of the implementation and study the effect of load balancing. Two different heuristic load balancing techniques are proposed: (1) a static load balancing that is based on estimated cost of each variant and (2) a dynamic load balancing that migrates variants between workstations, based on their estimated performance during the simulation process. Simple queueing models are used to study the performance. Numerical results obtained from real-time simulations on a network of up to 7 workstations are used to investigate the speedup and the efficiency of both the implementation and the load balancing techniques. As more workstations are added to the simulation environment, a sublinear speedup is obtained. In addition, the load balanced distribution has produced an improvement up to 8 per cent compared to random distribution. Secondly, a web based user interface is integrated to this engine to provide an easy to use and practical experimentation platform. The complete system mainly consists of a web based graphical user interface that communicates with a powerful server that runs the engine. The user interface allows creating simulation models of queueing network, performing simulation experiments using the simulation engine, and performing simple output analysis in a platform independent manner. Finally, a prototype for a web based simulation optimization tool is developed by extending the system with the power of experimental design methodology and response surface method. Hence, this tool provides a way of planning which variants to simulate in order to quickly reach the neighbor of the optimal solution. The applicability of this simulation optimization approach and the features of the interface are illustrated using a two-node Jackson network and a manufacturing system as examples.

ÖZET

STANDART SAAT ALGORİTMASININ YÜK DENGELİ GERÇEKLEŞTİRİLMESİNE DAYALI DAĞITIK BENZETİM ORTAMI VE ÖRÜN TABANLI EKLEMELERİ

“Standart Saat” yöntemi, bir modelin farklı parametrelili değışkelerinin benzetiminin gerekleřtirilmesinde kullanılır. Bu tezde ilk olarak, yöntemin oktörel UNIX ağı ortamı üzerinde gerekleřtirilmesiyle “Standart Saat” tabanlı dağıtık benzetim aracı geliřtirilmiřtir. Bu alıřmanın amacı, dağıtık benzetim uygulamasının öleklenebilirliğini ve yük dengelemesinin etkisini arařtırmaktır. İki farklı sezgisel yük dengeleme tekniğı önerilmiřtir: (1) her değışkenin maliyet tahminine dayalı statik yük dengelemesi ile (2) benzetim sürecinde başarımlarına dayalı olarak değışkeleri iş istasyonları arasında geiren dinamik yük dengelemesi. Başarım kuyruk ağı örnekleri kullanılarak incelenmiřtir. Yedi makineye kadar yapılan gerek zamanlı deneylerden elde edilen sayısal sonuçlarla hızlanma ile gerekleřtirilme ve yük dengeleme verimliliğı incelenmiřtir. İş istasyonu sayısı arttıka doğrusala yakın hızlanma elde edilmiřtir. Ayrıca yük dengelenmesi ile rassal dağıtıma göre yüzde 8’e kadar değışen iyileřtirme saėlanmıřtır. alıřmanın ikinci ařamasında, benzetim aracına örün tabanlı bir grafik ara birimi eklenerek kolay kullanımlı ve pratik bir deney yürütme platformu oluřturulmuřtur. Böylece, tüm sistem güçlü bir sunucu üzerinde alıřan benzetim aracı ile bu araçla iletiřim kuran örün tabanlı bir arayüzden oluřmaktadır. Kullanıcı arayüzü kuyruk ağıları için model geliřtirme, istenilen deneyleri benzetim aracını kullanarak gerekleme ve basit ıktı analizi yapmayı platform bağımsız olarak gerekleřtirmeyi saėlar. Son ařamada geliřtirilen benzetim ortamına deneysel tasarım yöntemi ve yanıt yüzeyi yöntemi eklenerek örün tabanlı bir benzetim eniyileme aracı prototipi oluřturulmuřtur. Bu araç, daha az ancak etkili deneylerin řeilerek yapılmasını ve kısa sürede eniyi sonuçlarının bulunduğı alana ulařılmasını saėlamıřtır. Bu benzetim eniyileme yaklařımının uygulanabilirliğı ve sistemin özellikleri iki düğümlü bir Jackson ağı ve bir imalat sistemi örnekleri üzerinde gösterilmiřtir.

TABLE OF CONTENTS

ACKNOWLEDGMENTS.....	iii
ABSTRACT	iv
ÖZET.....	v
LIST OF FIGURES.....	ix
LIST OF TABLES	xii
LIST OF SYMBOLS/ABBREVIATIONS	xiii
1. INTRODUCTION.....	1
2. BACKGROUND and LITERATURE SURVEY	5
2.1. Distributed and Parallel Simulation	5
2.2. Standard Clock Approach to Simulation.....	7
2.3. Simulation of Multiple Experiments Using SC Method	9
2.4. Previous Work on Standard Clock.....	11
2.5. Load Balancing	14
2.6. Web Based Simulation.....	15
2.7. Simulation Optimization	17
3. A WEB-ACCESSIBLE SIMULATION SYSTEM ARCHITECTURE.....	20
3.1. Front-End GUI and SSM	20
3.1.1. Model Development Module	22
3.1.2. Design of Experiments Module (DOE).....	23
3.2. Simulation Engine.....	25
3.2.1. The Client Process.....	25
3.2.2. The Server Process.....	26
3.3. Communication Issues	27
3.3.1. Protocol	27
3.3.2. Event Dispatching	27
3.3.3. Data Conversion.....	27
3.4. SC Simulation Design Issues	28
3.4.1. The Event Generation	28
3.4.2. Model Covered by the Engine.....	28
3.4.3. Thinning and State Update.....	29

3.4.4. Components of the Model	30
4. LOAD BALANCING	31
4.1. Total Time Requirement of Simulation	31
4.2. Estimating the Load of Variants	33
4.3. Static Load Balancing	34
4.3.1. Case 1. Load Balancing with Communication Overhead Overlapped	34
4.3.2. Computation of Acceptance Rates	36
4.3.3. Case 2. Load Balancing with Communication Overhead Non-Overlapped	37
4.3.4. Heuristic Algorithm for Load Balancing	39
4.3.5. Expected Speedup and Efficiency	40
4.4. Dynamic Load Balancing	41
4.5. Experimental Network Environment	45
4.6. Experiments by Varying Number of Workstation	46
4.6.1. Static Load Balancing	46
4.6.2. Comparison with other Static Allocation Strategies	50
4.6.3. Effect of Communication Overlapping	51
4.6.4. Dynamic Load Balancing	53
4.7. Experiments by Varying Parameters	58
5. MODEL DEVELOPMENT MODULE OF THE USER INTERFACE	60
5.1. General Overview of Model Development	60
5.2. Illustration of a Typical Session	61
5.2.1. Data Entry	62
5.2.2. Single Experiment	64
5.2.3. Simulation Execution	65
5.2.4. Presentation of Simulation Results	67
6. Application of Design of Experiments for Simulation Optimization	69
6.1. Two - Node Jackson Network	69
6.1.1. 2^k Factorial Design	70
6.2. Manufacturing System Example	76
7. CONCLUSIONS	81
7.1. Contributions of the Study	83
APPENDIX A: VERIFICATION OF THE SC ALGORITHM	84
A.1. M/M/1/ ∞ System	84

A.2. M/M/2/ ∞ System	84
A.3. M/M/1/8 System	85
A.4. M/G/1/ ∞ System	85
A.5. M/M/1/ ∞ with 2 Distinct Class of Entities.....	86
A.5.1. FIFO Queue Discipline with Regular Service.....	86
A.5.2. Priority Queue Discipline with Regular Service	87
A.5.3. Priority Queue Discipline with Preemptive Service	88
A.6. Hospital Emergency Room Problem: 3 Different Classes of Patients	89
A.6.1. One Doctor with Preemptive Service.....	89
A.6.2. Two Doctors with Preemptive Service.....	90
A.6.3. One Doctor with Nonpreemptive Service	90
A.6.4. Two Doctors with Nonpreemptive Service	91
A.6.5. One Doctor with FIFO Queue Discipline	92
A.6.6. Two Doctors with FIFO Queue Discipline	93
A.7. G/M/1/ ∞ with Two Distinct Classes of Entities.....	94
A.8. G/G/1/ ∞ with Two Distinct Classes of Entities	95
A.9. Network of 4 M/M/1/ ∞ Systems	96
A.10. Markovian System with Bulk Arrival ($M^B / M / 1$)	98
A.11. Markovian System with Bulk Service ($M / M^B / 1$)	99
APPENDIX B: FORMAT OF THE CD CONTAINING SOURCE CODE OF THE SIMULATION ENGINE AND FRONT-END GUI IMPLEMENTATIONS.....	100
REFERENCES.....	101

LIST OF FIGURES

Figure 2.1.	Event scheduling simulation	7
Figure 2.2.	Schematic view of Standard Clock Simulation.....	8
Figure 2.3.	Schematic view of single clock multiple system simulation	10
Figure 3.1.	The architecture of the web-accessible distributed simulation system	21
Figure 3.2.	Event data set	28
Figure 4.1.	Timing diagram of the simulation execution	32
Figure 4.2.	Timing diagram with communication overlapped	32
Figure 4.3.	Static load balancing algorithm.....	39
Figure 4.4.	Basic steps of the dynamic load balancing algorithm.....	44
Figure 4.5.	Two-node Jackson network.....	46
Figure 4.6.	4-node and 7-arc network with two classes of entities	47
Figure 4.7.	The speedup curve for the M/M/1 system with static load balancing.....	48
Figure 4.8.	Speedup curve vs processing power for different number of variants.....	52
Figure 4.9.	Computation and communication levels.....	52

Figure 4.10.	Speedup curve for the M/M/1 system with dynamic load balancing.....	54
Figure 4.11.	Progress of the simulation (workstation-4 and workstation-8)	56
Figure 4.12.	First 10 seconds of the simulation progress	57
Figure 4.13.	Speedup curves obtained for various acceptance rates	59
Figure 5.1.	Model development window.....	61
Figure 5.2.	Screenshot of the GUI with the model created while entering the arrival rate value $\lambda=0.5$ in the "Data Input" window	63
Figure 5.3.	Screenshot of the GUI while entering parameters for the service node 0 through "Node Information", "Update Info" and "Output" windows	63
Figure 5.4.	Experiment #60 with its parameters.....	65
Figure 5.5.	The dialog window "Run Information" used to supply run parameters.....	66
Figure 5.6.	Windows used to select computers and view the progress of the simulation run.....	66
Figure 5.7.	Output window depicting the result for experiments 50 through 60 in text form.	68
Figure 5.8.	Screenshot of graphical result for the example problem.....	68
Figure 6.1.	Design space used for the two-node Jackson Network.....	70

Figure 6.2.	Response surface of the two-node Jackson Network for $C_s=(1,1)$, $C_w=(1,1)$	71
Figure 6.3.	A snapshot of the "Design Phase" along with all the windows	72
Figure 6.4.	The ANOVA Table of 2^2 factorial design.....	72
Figure 6.5.	A snapshot of the "Model Fitting Phase" along with the "Regression Results"	74
Figure 6.6.	Proposed experiments on the steepest descent direction for the Jackson Network.....	74
Figure 6.7.	Steepest descent experiments proposed and results obtained for second first order model.....	75
Figure 6.8.	Steepest descent paths of two first order models for $C_s=(1,2)$, $C_w=(3,5)$	76
Figure 6.9.	Manufacturing system with four workstations.....	77
Figure 6.10.	The set of windows used to define the design parameters of the manufacturing system example.....	78
Figure 6.11.	The ANOVA table of 2^3 factorial design.....	79
Figure 6.12.	Model selection and ANOVA table of the first order model	80
Figure 6.13.	Proposed experiments for the line search.....	80

LIST OF TABLES

Table 3.1. Model development constructs.....	23
Table 3.2. Usage of random numbers.....	29
Table 4.1. Characteristics of workstations	45
Table 4.2. Results obtained for four models experimented.....	47
Table 4.3. Experimental results of the M/M/1 system for “Communication Included” case	49
Table 4.4. Results of different allocation strategies for the M/M/1 system	50
Table 4.5. Experimental results obtained with dynamic load balancing.....	53
Table 4.6. Number of migrations occurred	55
Table 4.7. Parameter setting for different experiments of the M/M/1 system with $\mu=10$..	58
Table 6.1. ANOVA table for the second first order model	75
Table A.1. Theoretical waiting time values	89

LIST OF SYMBOLS/ABBREVIATIONS

A	Average computational load of thinning operation
B	Average computational load of state update operation
CC	Average communication time of a batch between client and server
CR	Cumulative remaining workload
c	Coefficient of variation
E	Number of variants
e_i^k	Mean number of visit to node i in variant k
H _w	Workload assigned to the workstation w
K	Number of event types
L _w	Load allocated to the workstation w
M _w	Number of variants allocated to the workstation w
N(μ, σ)	Normal distribution with mean μ and standard deviation σ
NW _w	New workload of the workstation w
P	Number of processing nodes (workstations, servers)
P _w	The processing power of the workstation w
PC	Average execution time of a batch for one variant
PP	The total processing power of the computing environment
R _k	Total number of state update per unit time in variant k
RL _w	Remaining workload of the workstation w
r_i^k	Acceptance rate of event i in the variant k
S _w	Set of variants assigned to the workstation w
S _{pp}	Speedup with processing power pp
T _A	Setup time
T _{Ci}	Communication time for the transfer of frame i between two processes
T _{Ei}	Processing time of event frame i
T _F	Communication time for collecting results
T _k	The load of variant k

T_s	Simulation execution time
T_w	Finish time of workstation w
Δt	Time interval
TB	Total number of batches
TL	Total load of variants
TST	Total simulation time
u_i	Uniform random number
$U(a,b)$	Uniform distribution between a and b
V_w	Speed of the workstation w
X_w	Number of variants migrated to the workstation w
Y_w^j	Batch j processed in workstation w
β_j	Regression coefficient
ε	Error term
$\lambda, \lambda_{\max}$	Arrival rate, maximum arrival rate
λ_i	Rate of event i in SC simulation
Λ	Total event rate
$\mu, \mu_{\max}$	Departure /service rate, maximum departure / service rate
ρ	Traffic capacity
σ^2	Variance of the distribution
A1, A2,...	Allocation strategies
ANOVA	Analysis of Variance
CGI	Common Gate Interface
DEDS	Discrete Event Dynamic Systems
DOE	Design of Experiments
FD	Factorial Design
FIFO	First In First Out
G/G/1	Single server queueing model with general / arbitrary distributions
GUI	Graphic User Interface
HTML	Hypertext Markup Language

LPT	Longest (Largest) Processing Time
M/M/1	Single server queueing model with Poisson arrival, exponential service time
P/DES	Parallel / Discrete Event Simulation
PIRS	Parallel Independent Replicated Simulation
RSM	Response Surface Methodology
SC	Standard Clock
SIMD	Single Instruction Multiple Data
SSM	Simulation and Storage Manager

1. INTRODUCTION

A system is a collection of entities that act and interact toward the accomplishment of a logical end. In order to study the system, it is modeled in the form of logical and mathematical relationships. For simple models, the performance of the underlying system can be evaluated analytically. However, the real-world systems could not be solved by simple analytical methods because of the presence of stochastic elements and complex interactions between the system entities. In these cases, the model can be studied by using discrete event simulation. Most of the real discrete event systems are complex and technological developments bring about new level of complexity into these systems. Analytical solutions are available for only a limited class of discrete event dynamic systems (DEDS) which makes simulation the primary method for analyzing their performance and control.

Simulation of DEDS may have different objectives. When the simulation is performed to understand the behavior of a single, large and complex DEDS, a single simulation experiment is decomposed into tasks that can be run on separate computers. Once the model is ready, the simulation approach to problem solving can be used to guide the selection of parameters and operating strategies of that model. For this purpose, the simulation of a number of parametric variants of the same model has to be experimented. It typically involves a series of trials in which changes are made to the input variables in order to observe the resulting changes in the output variables. The ultimate goal of the simulation study is to determine the best designs. Although in many cases, simulation can provide a good estimate of system performance, it is also a time consuming and therefore a costly technique. However, the evolution of the computer technology in the area of distributed and parallel computing systems lead to the development of parallel and distributed approaches for discrete event simulation.

Standard Clock (SC) method introduced by Vakili, is a time synchronous approach that is convenient for parallel simulation of parametrically different and structurally similar variants of a single model (Vakili, 1992). The SC method is mainly experimented on massively parallel computers for different purposes (Vakili *et al.*, 1992; Strickland and

Phelan, 1995; Hu, 1995). On the other end, networks of computers present effective and economical platforms for high performance parallel computing. Obviously, an important factor affecting the performance of the simulation is the allocation of variants to workstations since an inefficient assignment may lead to long run times. The objective in making an assignment is to minimize the completion time of the entire workload. Hence, the design of a mapping strategy, known as load balancing, becomes important for the efficient utilization of the workstations employed in the simulation investigation.

The implementation of SC method on networks of computers is discussed in two recent papers (Bhatti and Vakili, 1997; Mollamustafaoğlu, 1998). Both implementations follow the client server model. The first one uses broadcast protocol on subnetworks of homogeneous workstations and the load balancing is achieved by balancing the consumption and production rates of events. However, some drawbacks are noted with such implementations. First, such a test bed may not be available for the simulation analyst. In addition, it may not be reasonable to reserve one workstation for the clock mechanism for a large number of variants. Therefore, it is preferable to carry out the experiments on a network of heterogeneous workstations. The second implementation of the SC method is on a network of heterogeneous Macintosh computers communicating via proprietary Apple Events. In this work, the load is balanced by distributing variants to computers in proportion to their static normalized performance index computed by a benchmark. The load balancing used is simple in the sense that it does not include different processing time requirement of each variant.

This study has three purposes. The primary objective is to examine the scalability of the distributed implementation of the SC method on a network of UNIX workstations and study the effect of load balancing. Queueing network models, which are widely employed in industrial production systems as well as computer network systems, are selected as the application area for this dissertation. In order to meet this aim, a simulation software prototype is developed which is used for the optimization of queueing network systems via simulation. However, it is not intended to develop a commercial simulation package and the necessary codes are written just to carry out the desired experiments. Secondly, to support web access, we intend to integrate a web based user interface to this system to provide an easy to use and practical experimentation platform. Finally, we proceed to

extend this study with the power of experimental design methodology and response surface based optimization technique to provide a way of planning which variants to simulate.

In this study, we first develop a simulation engine. Our primary focus is on studying possible balancing strategies and investigating how the loading of workstations effects the total processing time of the entire simulation for different simulation models. For that purpose, we describe two load-balancing techniques and experiment these techniques by using simple queueing models in order to analyze their performance in terms of efficiency and scalability criteria. We present numerical results obtained from real-time simulations on a network of up to seven workstations.

Previous simulation implementations based on SC method are coded using high-level programming languages such as C or C++. Consequently, none of these implementations is able to run through the web. To the extent of our knowledge, text-based interface is used for creating and feeding input to simulation models in these systems. With the advancements in Internet and web technology, web-based simulation environment provides a dynamic web interface for a standalone simulation engine. We believe that a web-based graphical user interface system will facilitate the development of simulation models and the analysis of results. Furthermore, it will make these existing platforms more usable, by allowing those who do not have access to these computing resources, and those who have no knowledge on standard clock algorithm.

As a second phase of this study, the simulation engine developed is then integrated with a web based access mechanism. The complete system mainly consists of a web based graphical user interface that communicates with a powerful server. The user interface facilitates the model construction and the analysis of simulation results in a platform independent way, while engine executes long running and expensive simulation experiments on the network.

Even though the SC method is an efficient method, the distributed simulation of all variants may be time consuming and expensive in terms of computation if the search space of the problem under study is large. Since the simulation studies are conducted within a time and budget limit, the requirement of the study can be reduced by carefully choosing

the simulation runs that will be carried out. This can be accomplished by applying the design of experiment techniques that provide a way of planning which variants to simulate. Based on the results of the current simulation runs, the optimization process may be carried out using the well-known response surface methodology to determine the new set of variants.

This study is further extended in such ways that we develop a prototype that integrates the power of experimental design methodology and response surface based optimization techniques to a simulation environment using the SC method.

Besides the introductory chapter, this thesis contains five more chapters. In the next chapter, some brief background in SC simulation and its application on a parallel computing platform are provided. Load balancing, web-based simulation, and simulation optimization are briefly explained in this part to form a basis. This is followed by the description of the web-enabled simulation system architecture based on the distributed SC method implementation in a network of workstations with emphasis to its components and their capabilities. In Chapter 4, the application of the static and dynamic load balancing techniques to SC method is discussed respectively, and the experimental results are presented. In Chapter 5, we illustrate the web-based interface developed as an aid for model development in detail, through an example queueing network problem, and discuss the integration of experimental design and response surface based optimization in Chapter 6. Finally, the last chapter concludes the thesis and gives general remarks for future improvements.

This thesis contains two appendices following the conclusion part; in Appendix A, the simulation results obtained using the SC method are verified using SLAM for various models. Appendix B gives brief information on the content of the CD.

2. BACKGROUND AND LITERATURE SURVEY

This chapter reviews the literature by referring to some original studies related to our topics. We start by introducing the distributed simulation and discuss in depth the Standard Clock (SC) approach to simulation to help the reader to get a theoretical insight as well as a better understanding of our study. Next, a survey of load balancing methods with focus on the basic concepts and different approaches are presented. This is followed by a quick survey of possible approaches to web based simulation. In the last section of this chapter, design of experiment and application of response surface methodology to simulation optimization are discussed.

2.1. Distributed and Parallel Simulation

In Discrete Event Simulation (DES) (Law and Kelton, 2000), the essential features of a physical system are described by a collection of state variables. Changes to the state variables are assumed to occur instantaneously and only at specific discrete points in time. The changes to the system state are specified through the occurrence of an event. For instance, a typical event is an arrival of a new customer in a queueing system with n customers waiting at a certain time t , hence increasing n by one. A customer leaving the queue, therefore decreasing n by one, would be the occurrence of another type of event.

Simulation of complex DES often requires large amounts of resources in terms of CPU time and memory. One possible solution to this problem is to use parallel or distributed simulation techniques. Nowadays, in many computer environments a large number of processors are available as either a cluster of workstations or multiprocessor machine architecture. The parallel and distribution simulation techniques use such a multiprocessor environment to increase the simulation efficiency. Two potential benefits of the techniques are: (1) the execution time of a simulation run could be reduced by execution in parallel, and (2) more memory could be used by distributing the execution over several nodes or workstations.

All approaches for exploiting parallelism in discrete event simulation are based on splitting up the total simulation process. There are four main ways of parallelism in the literature as follows;

The first two approaches are *dedicated execution* and *hierarchical decomposition*. The first one uses dedicated functional units to implement specific simulation functions (e.g. random number generation, event-list manipulation, or statistics collection). In the second approach, the simulation model is decomposed in a hierarchical fashion so as to allow an event consisting of several sub-events to be processed concurrently (Fujimoto, 1990).

The well-known approach is the Parallel (and Distributed) Discrete Event Simulation (PDES) which is the execution of a simulation model on a parallel computer or as distributed over a network. This is accomplished by decomposing the simulation application into a set of concurrently executing processes (Misra 1986). In general, it is not simple to identify submodels that can be run in parallel as separate processes. In DES, all events have an associated time-stamp indicating at what simulated time the event should occur. In a sequential simulation, the events are processed in time stamp order to ensure that an event cannot affect its history (known as a causality error). On the other hand, in PDES, all processes run independently but the problem is that they are not independent. They have to maintain certain *sequencing constraints* in order for the overall simulation process to be correct. Hence, the processes have to be synchronized at interaction points to avoid causality errors. Mechanisms proposed to handle this problem fall roughly into two categories: *conservative* and *optimistic* synchronisation schemes (Fujimoto, 1990). Conservative schemes ensure that only *safe* events are processed so that causality errors never occur, while optimistic schemes use a detect and recovery mechanism in the case of a causality error occurring.

Considering the stochastic simulation models, each simulation output represents only one possible outcome of a statistical experiment. Hence, it is necessary to perform a number of replications of each simulation run and use statistical analysis to draw conclusions about the system. A straightforward and often efficient way to exploit parallelism is referred as the Parallel Independent Replicated Simulation (PIRS), in which

several replications of the sequential simulation process are executed independently on separate processors (Heidelberger 1988). PIRS is a framework to get more observations in shorter time simply by distribution several replications of the sequential simulation process on processors. One drawback with this approach is that one workstation might not have sufficient physical memory for a single replication. This approach is useful if the simulation is largely stochastic, that it need a number of independent observations to obtain a certain confidence of the estimates or if one attempting to simulate a specific simulation problem across a large number of parameter settings.

Related with the PIRS, another set of research involves the task of simulating different variants of the same system for finding the best system among the alternatives. The Standard Clock method addresses this approach by running separate experiments on different workstations.

2.2. Standard Clock Approach to Simulation

Simulation process involves two primary components: event generation and state updating. These components are coupled in event-scheduling mechanism; the feedback of the state information is used by the event generator (Clock Mechanism) to determine the next event time and type. The event is used by the state update component, which leads to a new feedback that controls the scheduling and generation of events. The simulation progresses in this manner as seen in Figure 2.1.

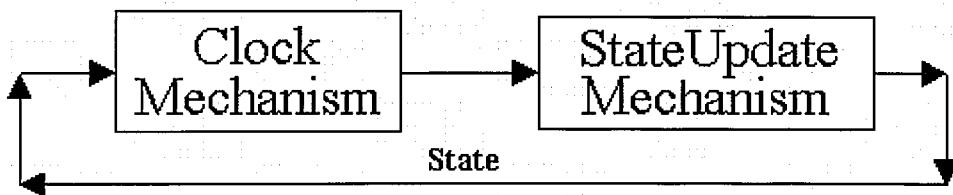


Figure 2.1. Event scheduling simulation

The SC approach decomposes the event generation from state update. In the method, generic events are generated at the "maximal" rate, which can be calculated from the individual rates of actual events. The clock that ticks at the maximal rate is named as the

Standard Clock. Each generic event is converted to an actual event according to the probability of occurrences of each actual event through *thinning* process. The state update mechanism is represented as an autonomous process. It uses the events generated in this manner to update the state as depicted in Figure 2.2.

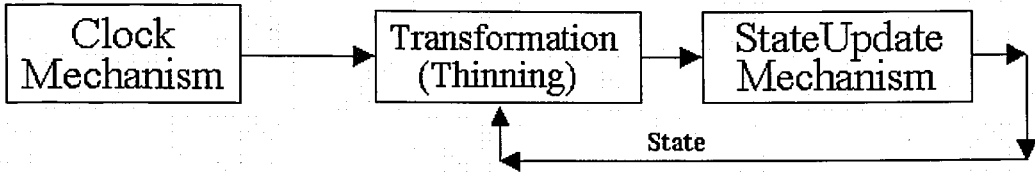


Figure 2.2. Schematic view of Standard Clock Simulation

For the sake of illustration, considering the M/M/1 system with arrivals rate λ and exponential service with mean $1/\mu$, generic events will be generated at the rate $\Lambda=\lambda+\mu$ instead of generating two different types of events separately. Generic events are marked as

- an arrival event if $u_i < \lambda/\Lambda$ or
- a departure event if $u_i \geq \lambda/\Lambda$

where $u_i, i=1,2,\dots$, is a sequence of uniform random variables on $[0,1)$. By the virtue of the Markov property, the arrival and departure events generated in this manner are statistically indistinguishable from that which are generated directly as two separate sequences of arrival and departure events with rate λ and μ respectively.

This method is mainly applicable to Markovian queues since it relies on the memorylessness property of the exponential distribution in order to be able to use the concept of thinning. However, nonexponential distributions can also be incorporated using shifted exponential and hyperexponential approximations (Chen and Ho, 1995). The basic idea is to match the first two moments of the distribution. To summarize, the general distribution with mean μ and variance σ^2 , two cases must be considered depending on whether the coefficient of variation, $c=\sigma/\mu$, is less than one or greater than one.

When $c < 1$, a shifted exponential distribution with rate $\lambda = \sigma^{-1}$ and shift factor $d = \mu - \lambda^{-1}$ is used and this is accomplished by thinning successive events occurring before a specified time d . As to the case when $c > 1$, an hyperexponential distribution with balanced means is used. An exponential distribution with rate λ is converted to an hyperexponential distribution with rates a_1 and a_2 , and the probability p , where $a_1 = 2p\lambda$ and $a_2 = 2(1-p)\lambda$. Two thinning routines are applied to each event; one to decide on the actual rate of the event and the other to reduce the rate λ to a_1 or a_2 depending on the selection of the first thinning.

Compared to classical simulation, SC method does not need an explicit event list such as *Future Event List*. The main advantage of the method is the separation of two components that allows the independent development of each, thereby simplifies the simulation model development. Its benefit becomes important when more than one experiment are to be run. By using the SC method, part of the simulation common to all experiments (e.g. generation of events) is done once. Only the state update mechanism needs to be implemented separately for each experiment. Thus, this decreases the simulation work and the synchronization among experiments is achieved by sharing of the events.

2.3. Simulation of Multiple Experiments Using SC Method

The SC method is originally designed to perform several experiments simultaneously on a parallel computer. Simulation of multiple experiments consists of running N variants of the model, each variant differs in one or more parameters. When simulating a set of parametric variants of a model, it is obvious that event processes should be similar. Hence, the common sequence of events generated by the SC method can be used to drive all experiments as depicted in Figure 2.3. Although events can be generated in many different ways, in an efficient one, the method generates events at the fastest possible rate and marks them according to the maximal occurrence probabilities of each event type. Event types are determined according to the outcome of a random number via the alias method (Ho and Coa, 1991). Through this method, the determination of the event types is independent of the number of different event types, an indicator of the model complexity.

As seen in Figure 2.3, each experiment consists of a transformation process and a state update process. The transformation process does thinning that rejects some events based on local parameter and state information. *Rate thinning* recovers the local rate of each event from its SC rate and *state thinning* reduces the rate of the invalid events to zero, hence, only the accepted events cause state updates in each experiment.

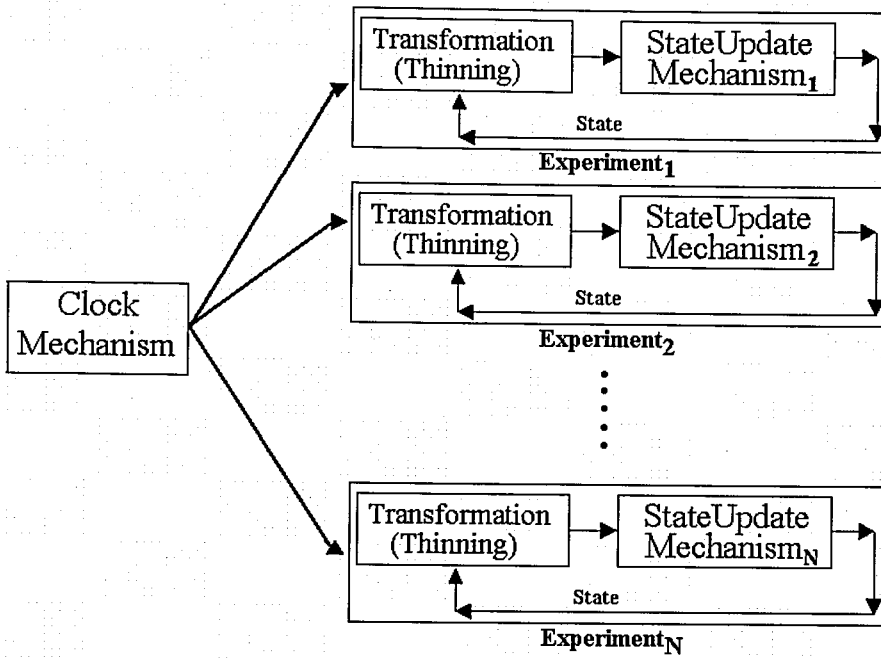


Figure 2.3. Schematic view of single clock multiple system simulation

For the sake of illustration, the simulation analysis of a single server, Poisson arrival, exponential service time queueing model designated as M/M/1 is experimented. These queues are characterized by arrival rates λ_i and service rates μ_i for $i=1\dots N$, where each experiment corresponds to a *variant* i of the model, generic events are generated at the rate

$$\Lambda = \lambda_{max} + \mu_{max} \text{ where } \lambda_{max} = \max_i \{\lambda_i\} \text{ and } \mu_{max} = \max_i \{\mu_i\} \quad (2.1)$$

The events are classified as arrival events with probability λ_{max}/Λ and as departure events with μ_{max}/Λ .

The experiment, say i , thins the arrival events with probability λ_i/λ_{max} , similarly the departure events with probability μ_i/μ_{max} . Similar to event type determination, the thinning

is performed by means of random numbers, i.e., an event is accepted if the random number is less than or equal to the above ratio. In addition, through state thinning, the events corresponding to a departure from an empty queue are discarded.

Application of the SC method to the advanced structures requires the development of necessary transformation routines. In (Mollamustafaoğlu, 1992), the author has extended the SC method for simulating general queueing network systems with complex structures such as multiple entity classes, parallel identical servers and finite buffers. Similarly, the idea is to generate events at a rate of a maximal system, which is defined as the sum of maximum rates of all event types. Each event is also associated with a set of six random numbers that are used for rate reduction. Two of which are used to determine the event type and the inter-event time in the clock mechanism while the remaining four are used for transformation and routing in the experiments. The maximal rate of an event is transformed to the corresponding rate of each experiment by means of a series of thinning filters. Hence, the transformation component is modeled as a collection of filters. "Node thinning filters" perform rate thinning discussed above. Validity of the event for the current state of the experiment is checked with a "Feasibility filter". The model uses a single event type for departures from any class, therefore an additional "Class thinning filter" is applied to the departure events to reduce the maximum departure rate of the node to the departure rate of a particular entity class. Multiple parallel servers at any node leads to an increase in the service rate, which is reduced to the correct service rate by a "state dependent filter".

2.4. Previous Work on Standard Clock

First papers on the SC method discuss its implementation on massively parallel computers. The models, which are investigated in these simulation studies, are mainly queueing theoretic models dealing with analysis of service systems. Most of the analysis on these studies mainly focus on the efficiency of the method in terms of the simulated event rate.

In Vakili's earlier work, (Vakili, 1992) the SC method is first introduced as a new approach to parallel and distributed simulation of DEDS. A detailed description of its implementation on the Single Instruction Multiple Data (SIMD) computer is discussed.

The simulation runs under the direction of a serial computer, called the front end, that broadcasts events to each processor of the SIMD computer which in turn execute state update corresponding to the event on the assigned experiment. The speedup and the efficiency of the SC method are described as a function of the utilization of events. It is argued that the rate of rejected events is directly translated to an inefficiency in the SC method of the same magnitude since the processor which rejects an event remains idle until all processors finish executing the current event.

The general queueing network simulation program named Parsim (Vakili *et al.*, 1992) is another implementation of the SC method on SIMD computer. Experimental results show that a speedup is achieved for a large number of experiments. However, two drawbacks of SIMD computers in the context of simulation of multiple experiments are noted. Firstly, simulating only a small number of experiments causes under-utilization of the computer. Secondly, simulating variants of large systems is not possible due to the limited memory of each processor. In addition, a set of variants of a queueing network with varying feedback probabilities is simulated in parallel in order to determine the effect of the parameter on the performance. The result shows that execution time increases as the number of entities in the system increases.

Another study focuses on the efficiency of the parallel simulation on the SIMD computers using the SC method in terms of number of events simulated is in (Strickland and Phelan, 1995). In this investigation, event efficiency is defined as the fraction of "front-end" events that are simulated at the "back-end". This efficiency depends on two types of losses: the thinning losses for which it is trivial to compute for most of the classical family of systems and the losses due to invalidity of an event. However, the latter is not readily calculable for more general systems such as the network of queues. The event efficiency of the SC method is compared with the so-called "event synchronous" method.

The study of Wieselthier (Wieselthier, J. *et al.*, 1995) computes the simulation time of the SC method on a sequential machine for a finite queue capacity system designated as M/M/1/K for the case of traffic capacity $\rho=1$. As expected, results show that efficiency of the simulation decreases as the queue capacity K increases. It is also pointed that the SC simulation of network of queues does not differ much from that of a single queue apart

from the number of event types. SC method is applied to an admission control problem on communication network. A simulation model for wireless integrated network, which includes extensions of SC model to include M/D/1 systems, is developed.

The implementation of SC method on networks of computers is discussed in two more recent papers (Bhatti and Vakili, 1997; Mollamustafaoğlu, 1998). Both implementations follow the client server model.

In the work of Bhatti and Vakili, the performance of the distributed implementation of standard clock method is described in the context of simulation of wireless communication networks. The model is implemented on a subnetwork of homogeneous workstations and one-way communication based on the UDP/IP broadcast protocol is used between client and server processes. The client process is running the clock mechanism on one of the workstations and broadcasting the events, and each remaining workstation runs a server process simulating certain number of update mechanisms. The implementation disregards the packet loss due to unreliability of the UDP Protocol, but the buffer overflow is reduced by balancing the load in each workstation. Load balancing is achieved by a simple experimentation which involves increasing the number update mechanisms run on each server in order to determine the number of experiments to be run so that the consumption and production rates of events are close. However, not only the applicability of broadcast protocol is limited to subnetworks but also the subnetworks of homogeneous workstations might be unreachable. It may be unreasonable to reserve one workstation for the clock mechanism for a large number experiments. Furthermore, the state update mechanism can be more complex than the one described in that work.

An object-oriented framework for distributed discrete-event simulation is described in (Mollamustafaoğlu, 1998). In this research, the distributed version of the algorithm in (Mollamustafaoğlu, 1992) is implemented on a network of heterogeneous Macintosh computers using the proprietary protocol named Apple Events. Three different queueing models - M/M/1/K, three-node Tandem and a multiclass M/M/1 are simulated to investigate the reduction in the total simulation time as more computers are added to the run. The load is balanced by distributing variants to computers in proportion to their static normalized performance index computed by a benchmark. Results show that a sublinear

speedup is obtained with an efficiency value greater than 80 per cent for both M/M/1 and three-node Tandem model for which it increases above 90 per cent. The load balancing used is simple in the sense that it does not include different processing time requirement of each variant.

2.5. Load Balancing

Full potential of any distributed processing system can be realized by evenly/proportionately sharing the computational and communication load among all processors. The even distribution of the load among several processing nodes is necessary to achieve some performance goal. This problem is referred to as *Load Balancing*. Load balancing is realized by distributing/redistributing loads to processing elements in such a way that the performance measure is optimized. Depending upon the problem on hand, commonly used examples of optimization measure are to minimize the total completion time or the average response time, to maximize the system throughput, or to maximize the system's processing elements utilization. Load Balancing techniques can be classified into two major categories; static techniques or dynamic techniques.

The static load balancing technique is based on the prior knowledge of the workload in the application as well as the processing capacity of each processing node. The workload is then balanced before the execution using this prior knowledge. The static load balancing techniques are NP-complete since they can easily be mapped to "set-partitioning" problem. The simplest static load balancing technique is to assign workloads to nodes according to either some cycling schedule or given probabilities determined statistically according to factors such as execution rate of each processor (Wang and Morris 1985; Burdorf and Marti 1993; Shrazi *et al*, 1995). Load balancing problem is also solved by the well-known task scheduling heuristics (Selvakumar and Murthy, 1994; Shrazi *et al*, 1995). Some other static balancing strategies that can be applied to parallel simulation are discussed in (Nicole and Fujimoto 1994). Most of them apply to PDES in order to minimize the communication among submodels. Apart from random partitioning, the methods used for load balancing are structural partitioning as used in digital logic simulation, and partitioning according to the critical path or the task graph computed through multiple execution of the simulation.

The dynamic load balancing, on the other hand, attempts to maximize the utilization of the environment at all times (or at some check points) by migrating the workload among processors during the runtime. In essence, dynamic load balancing strategy may be viewed as an ongoing decision process with the goal of improving the performance of the application. The dynamic load balancing requires the estimation and updating of the load information that is typically represented by a *load metric*.

The dynamic load balancing can be classified into three different models depending on how the workload is redistributed. First, dynamic load balancing operations may be centralized in a dedicated processor, which is responsible for maintaining a global state of the system. Based on the gathered information, the central processor reschedules the workload to individual processors (Lu and Tan 1994; Hamdi and Lee, 1997). The centralized model has the advantage of collecting easy and up-to-date information about the execution of jobs which leads to the utilization of more sophisticated load balancing techniques. Second, in a fully distributed model, processors of the system build their own views of the global state and make autonomous decisions based on the information exchanged with other processors (Cybenko 1989; Franklin and Govidan 1996; Kohring, 1995). Third, a semi-distributed model combines the advantages of both of the above-discussed models, i.e., a centralized node per cluster and distributed cooperation between neighbored clusters (Calinescu and Evans 1994; Evans and Butt 1994).

The literature is rich in terms of dynamic load balancing algorithms used in simulation field. However all of them apply again to PDES for conservative and optimistic synchronization schemes. (Burdorf and Marti 1993; Glazer and Tropper, 1993; Schlagenhaft *et al.*, 1995; Carothers and Fujimoto, 1996; Khalil and Tropper, 1999)

2.6. Web Based Simulation

The web-based simulation is first discussed as a new direction for simulation research and development in various papers in a special session on web-based simulation at the 1996 Winter Simulation Conference (Buss and Stork, 1996; Fishwick, 1996; Nair *et al* 1996). The key enabler of the web-based simulation is the Java language. It provides new possibilities for simulation languages and enables simulation models to be distributed

over the web in a platform-independent manner. Executable content such as Java applets facilitates this type of client side execution. Basically, there are three main approaches to web-based simulation.

The first approach is the simplest way to allow simulation models to be executed over the web. It consists of the simulation model components and user interfaces that are downloaded on the browser from Internet and executed upon the remote user's computer. Typically, this is accomplished by Java applets that are embedded on web home pages. However, the performance of this type of simulation is limited by the client computer performance capability. In addition to the ones at various sites, a list of applets for different types of simulation studies can be found in (Page, 1997).

The second approach that is closely related with this study, executes the simulation at the server. The remote user is typically able to specify certain parameters governing the simulation and view the results of the experiments. In the early stage, Hypertext Mark up Language (HTML) pages and Common Gateway Interface (CGI) scripts are used for parameter entry and model invocation. This can also be done by having a thin client running on a browser communicate with a server where the models run. The simulation program is executed on powerful servers. It might have been written in Java (eg. as a Java application or a Java servlet) or in another programming language such as C++ or Fortran. The advantage with this approach is that larger simulations can run on powerful, high-end computers, and it enables the reuse of existing tools (Kaylan *et al*, 1997; Kaylan *et al*, 1998).

The third approach allows the remote viewer to construct and execute the simulation model using the services of the web. Most of the current research in the web-based simulation involves the development of Java based discrete simulation systems and environments. They are mainly a collection of packages that permit the construction of simulation programs of the last approach. These packages fall into two categories: a) new simulation languages implemented in Java such as Simkit (Buss and Stork, 1996), JSIM (Miller *et al*, 2000), Silk (Healy and Kilgore, 1997; Healy and Kilgore, 1998; Kilgore *et al*, 1998), Netsim (Veith, 1999), b) Java implementation of existing simulation packages, such as JavaGPSS (Klein, 1998) and SimJava (Howell, 1999).

2.7. Simulation Optimization

A simulation optimization problem could simply be defined as an optimization problem where the objective function consists of a set of responses that can be evaluated by computer simulation. For a simulation model, with N input variables ($X_1, X_2, \dots, X_N$) and M output variables ($Y_1, Y_2, \dots, Y_M$), the objective of the simulation optimization is to find the optimum values ($X_1^*, X_2^* \dots X_N^*$) that minimizes or maximizes the output variable(s). In order to solve the simulation optimization problem, both the simulation model implemented on the computer and an optimization procedure are needed. The optimization procedure uses the outputs from the simulation model that evaluates the outcomes of the inputs that are fed into the model. Therefore, the process of a simulation optimization is to orchestrate the simulation of a sequence of system configurations so that a system configuration is eventually obtained that provides an optimal or near optimal solution. Based on the past evaluations and current simulation outputs, the optimization procedure decides on the new set of input values until an appropriate termination criterion is satisfied. The optimization procedures reported in the literature could be classified basically into four categories: Gradient based search, Stochastic Approximations, Heuristic Search Methods and the Response Surface Methodology (RSM) that we will focus on in this thesis.

RSM is mainly a collection of statistical and mathematical techniques useful for optimizing functions (Meyers and Montgomery, 1995). The methodology attempts to fit a regression model of appropriate degree to the response of the system. The application of the RSM to simulation optimization falls into two categories: (1) metamodels and (2) sequential procedures. In the former method, simulation is used to fit a global response curve, which is then treated as a deterministic curve and optimized using deterministic procedures. However, this method requires the largest amount of simulation effort in order to characterize the response curve of the entire domain. In the literature, simulation optimization using RSM refers to the second category in which small subregions are explored in succession (Kleijnen, 1998; Neddermeijer *et al* 2000).

The RSM procedure applies a more “blackbox” approach to the optimization, so it is unclear a priori the importance of input variables. A carefully planned design of the

experiments can provide us with valuable data from which to extract the relations. The main goal of DOE applied to simulation is to estimate how changes in input variables affect the output variables of a simulation experiment (Kelton, 2000). It provides a way of planning which configurations to simulate so that the desired information can be obtained with the less computational burden. In the DOE terminology, the input variables are called *factors* and the performance measures or outputs are called *responses*. The “experiment” is the execution of a simulation model with the factors fixed at certain values. Each factor takes at least two values or *levels*, for the designed experiments.

Factorial designs are the most commonly used class of designs because they are easy to visualize, and they permit the estimation of cross-product terms. The example of this class is the 2^K full-factorial design, in which for 2 levels and K factors, 2^K different combinations of the factors, each defining one experiment of the model, are obtained. The levels are usually set to be “opposite” in nature and referred as the low and the high levels. The responses of the experiments can be used to estimate the main effect of each factor and interaction effect between the factors. Depending on the complexity of the model, available time, and computational resources, acceptable designs may be 2^K or 3^K factorial designs.

Most experimental designs are based on a regression-model assumption about the way the factors affect the response. If there are two factors, say X_1 and X_2 , that are thought to affect the objective function Y of the simulation model, one might approximate this relationship by the regression model,

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_{12} X_1 X_2 + \beta_{11} X_1^2 + \beta_{22} X_2^2 + \varepsilon \quad (2.2)$$

where the β_j coefficients are the unknown regression coefficient to be estimated, and ε is a random error term representing whatever inaccuracy such a model might have in approximating Y . The regression models typically fall into one of the following three classes: (i) a first-order polynomial, (ii) a first-order polynomial augmented with interactions between pairs of factors (two-factor interactions); and (iii) a second-order polynomial, which also includes purely quadratic terms.

The optimization process is usually carried out by a two-stage version of the sequential RSM, which can be outlined as follows. The process usually referred as the first phase, starts on a small subregion of the domain, described by a current point of interest and the step size. Simulation runs are carried out at design of experiment points on this subregion. A first-order regression is performed to characterize the response surface around the current point. A line search is carried out in the direction of steepest descent/ascent to determine the next region of interest. This process is repeated until the first-order fit is inadequate which is indicated by a significant lack-of-fit. In the second phase, the objective function is approximated by a second order polynomial, and the optimum is determined analytically from this fit.

3. A WEB-ACCESSIBLE SIMULATION SYSTEM ARCHITECTURE

The architecture of our web-accessible simulation system developed and used in this study consists of three principal software components; the Simulation and Storage Manager (*SSM*), the Front-End Graphical User Interface (*GUI*) and the *Simulation Engine* as shown in Figure 3.1. The SSM is hosted by the web server, interfaces with a *web client* and drives the simulation engine. The simulation engine itself comprises two interrelated applications: the *simulation dispatcher* application that runs on the web server, and the server application. A copy of latter is run by each server participating in the simulation study. Thus, this whole system consists of two interconnected subsystems, each based on client/server architecture interacting with each other.

The SSM component orchestrates basically the simulation experiments that are performed on the simulation engine after the simulation parameters have been received from the web client. When a simulation experiment is finished, SSM stores results in a database and transmits them to the web client.

3.1. Front-End GUI and SSM

The graphical user interface in the Web client, *Front-end GUI*, is provided by a Java applet downloaded into a browser. The user accesses the simulation environment remotely through this applet running in a Java-enabled platform. It consists of a single applet that resides on the web server that is loaded and executed on demand of the user's client computer. The Front-end applet only communicates with the **SSM** component in the Web server for diverse purposes. The operations that can be initiated by utilizing of the Front-end GUI are the followings:

- loading and modifying already existing models,
- requesting the desired simulation experiments, and viewing the progress of the execution
- viewing a particular simulation experiment with its parameter settings,
- viewing the simulation results and carrying out simple statistical data analysis,

- saving and retrieving simulation models,
- assigning the servers that will participate in the execution of simulation experiments.

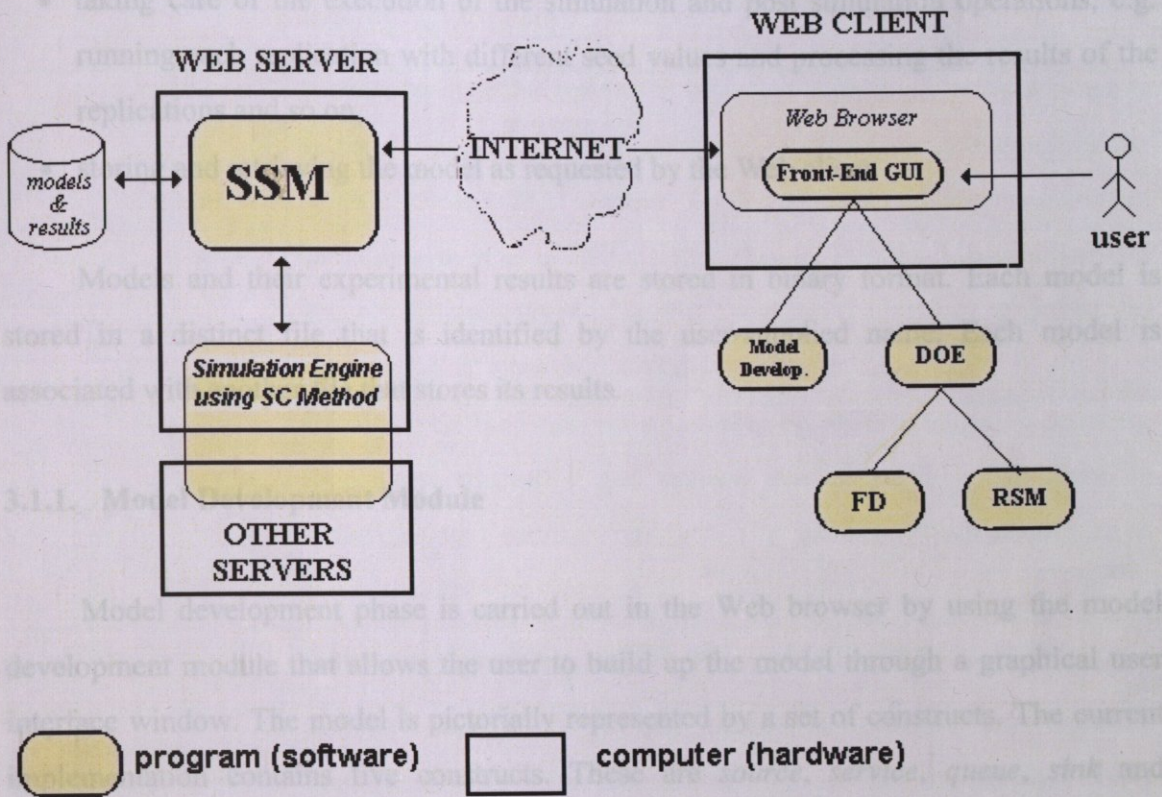


Figure 3.1. The architecture of the web-accessible distributed simulation system

The purpose of the SSM is twofold; it controls the entire simulation process and manages the storage and retrieval operations. It directly communicates with the simulation client and the Web client. It accepts requests from the Web client and mainly performs the following operations:

- conveys the information regarding the model, the run parameters and the results between the simulation client and the Web client,
- prepares a list of available servers by checking the status of each server dynamically and computes workload prior to the assignment of experiments,
- activates servers that take part in the simulation execution and launches the simulation client with all necessary information; e.g. the model, list of servers with their URL and processing speed,

- periodically notifies the Web client about the current status of each server during the execution as feedback from the simulation client in form of list of events processed, and the final status when execution terminates,
- taking care of the execution of the simulation and post simulation operations, e.g. running each replication with different seed values and processing the results of the replications and so on,
- storing and retrieving the model as requested by the Web client.

Models and their experimental results are stored in binary format. Each model is stored in a distinct file that is identified by the user-supplied name. Each model is associated with another file that stores its results.

3.1.1. Model Development Module

Model development phase is carried out in the Web browser by using the model development module that allows the user to build up the model through a graphical user interface window. The model is pictorially represented by a set of constructs. The current implementation contains five constructs. These are *source*, *service*, *queue*, *sink* and *connector* as described in Table 3.1.

Model constructs are diagrammed as simple graphical shapes, and each construct is labeled with a sequence number. The user builds up the model by inserting these constructs into draw area and supplies parameters through a dialogue window appropriate for the current construct. The user may also specify the ranges over which the factors will be varied by some specific amount for an exhaustive search of the entire experimental region.

Table 3.1. Model development constructs

Type	Icon	Description
Source		It is the calling population which generates entities into the model from the outer world.
Service		It characterizes the service mechanism with a predefined number of parallel servers and collects statistics. The user also specifies the service types. The current implementation supports complex structures such as multiple entity classes, parallel identical servers and six different types of service: regular, preemptive, preemptive-resume, batch, assemble, and disassemble.
Queue		It holds a defined maximum number of waiting entities according to a given discipline and collects statistics. They can only be connected to a service construct. The current system allows up to three queues to be connected.
Sink		It consumes departing entities and records statistics about them.
Connector		It keeps track of the routing information between two logically appropriate constructs respectively, i.e. Source to Queue, Service to Service, Service to Queue, and Service to Sink.

3.1.2. Design of Experiments Module (DOE)

The experimental design and optimization process are realized in the DOE module, which consists of Factorial Design (FD) and RSM modules. The experimental design phase is carried out through the FD module. This module lets the user define various technical parameters related with the design, and analyze the results. All steps of this phase are realized through a sequence of user-friendly interface windows. They are employed for

- entering the design parameters (i.e. type of the design, number of factors, number of center points and number of replications),

- selecting the factor from the model parameters such as service or arrival rate, queue length or number of servers. They can be chosen among the input parameters that are specified in the development phase,
- setting the levels of the selected factors if default values will not be used,
- setting the objective function (e.g. cost function), the objective function may be composed of one or more performance measures as well as the input parameters,
- analyzing the effects of factors and their interaction after the experiments are executed.

Current status of the FD module is capable of handling all full and fractional factorial designs up to five factors. The FD module determines the input factors and/or their interactions, which have significant effect on the objective function. These factors or interactions are then considered as input of the regression analysis in conjunction with the Response Surface Methodology to fit a functional relationship between the response and its corresponding factors.

The optimization phase is carried out through the RSM module through a sequence of user-friendly interface windows. This module lets the user to choose the type of the regression model that will be used for the predicted response values. The current prototype of the module supports two different types of regression models; they are linear, pure quadratic. For all models, the user has the opportunity to select the factors for which the coefficient of the main and the interaction effects are estimated and their statistical significances are checked.

The results are given in an Analysis of Variance (ANOVA) table that is especially designed for RSM. For the selected regression model and the lack-of-fit of the interaction effects dropped from the model, it displays their sum of squares, degree of freedom, Mean Square, F-ratio, and critical F-ratio. In addition, the best model fit that describes the behavior of the objective function is given below the table to help the analyst.

By using the accepted model, optimization is carried out. The current prototype implements the steepest descent/ascent algorithm to provide a set of runs for a first-order model or solves the quadratic equation for the second order model.

3.2. Simulation Engine

The simulation engine uses the distributed implementation of the Standard Clock approach on networks of heterogeneous UNIX workstations. The well-known client-server model is used. The program consists of two different types of processes; the client process and a collection of server processes. The clock mechanism is carried out in the client process, which acts as the controller of the entire simulation. Hence, it is the dispatcher of simulation runs as well as the compiler and analyzer of the output data. The server process, on the other hand, simulates a set of individual variants. There exist as many server processes as the number of workstations. These processes operate simultaneously and independently, and they communicate with the client process for the transfer of events and with each other for the migration of variant data. It should be noted that the client-server concept here might be contradictory to normal practice, in which a powerful machine is used as the server computer. The pictorial view of the simulation on three workstations is illustrated in Figure 3.2.

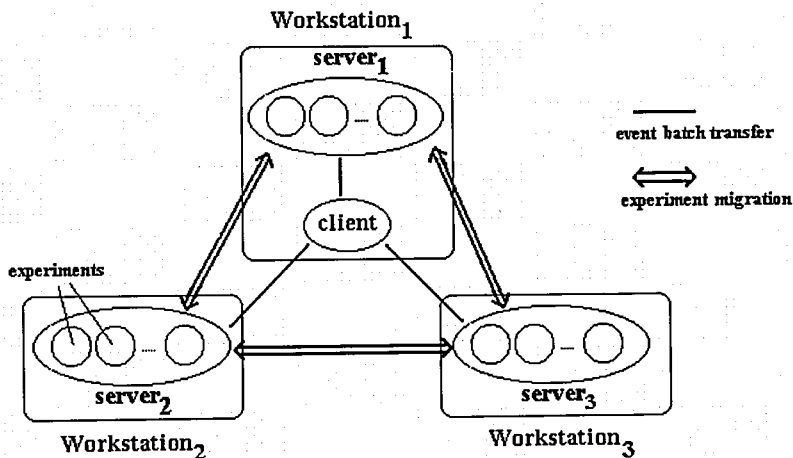


Figure 3.2. Simulation using three workstations

3.2.1. The Client Process

The administrative work is done by the client process in a three-phased algorithm, where each phase is executed sequentially. The working sequence of each phase is given below:

- The setup phase, in which the model and the update information as well as the system parameters are read, and the necessary computations for the clock mechanism are performed. The static load balancing decisions are made. The decision consists of the number as well as the list of variants to be run on a particular workstation. After having sent the model and the corresponding updates to each server, the simulation is started.
- It keeps on generating events, and dispatching them to servers iteratively in accordance with the server requests. If a dynamic load balancing is used, in addition to feeding servers with events, the client process behaves like a regulator in this step. By periodically monitoring the simulation progress, it decides when to perform variant migrations as well as the number of variants for each server involved in the reassignment of variants and sends the necessary information to these servers instead of the next batch of events.
- It collects the simulation results when all servers terminate their simulation variants.

3.2.2. The Server Process

The server process is mainly responsible for the processing of events of the assigned variants. It consists of three phases as described below:

- It creates model variants based on the information received from the client.
- During the simulation phase, each server processes sequentially the events (the event set) transferred from the client for each variant that it simulates. Processing involves thinning the events and updating the state. It keeps on requesting new events and processing them until the predetermined number of events. After having received a message for migration, the server communicates with another server to fulfill data migration.
- Finally, simulation results are sent to the client.

3.3. Communication Issues

3.3.1. Protocol

In the computer network, communication between the servers and the client is carried out as message passing which is accomplished by the socket system calls. We use the UDP/IP protocol i.e., connectionless datagram service because it is less expensive than the TCP/IP protocol. Considering the fact that UDP is an inherently unreliable protocol, stop-and-wait flow control - acknowledgement and error checking - is used to ensure the correct transfer of data to/from the client that concurrently communicates with all servers. As to the GUI, the applet communicates with the SSM over the network through TCP/IP sockets with predefined ports.

3.3.2. Event Dispatching

Load balancing tries to achieve a balanced partition of the variants into servers so that each progresses at the same rate. However, the error made in estimating the speed of each workstation may cause an imbalance among the partitions and the events requested by each server is likely to differ at a certain time of the simulation execution. Consequently, the client has to trace the progress of the simulation in each server. Besides, since the messages may be lost because of the traffic load congestion, the client keeps all the events for a possible retransmission. This is accomplished by using a sliding window control. To avoid the communication overhead, a batch of events is sent in each message. The appropriate batch size is determined via initial experimentation so that no data loss is encountered because of fragmentation.

3.3.3. Data Conversion

The number of data to be sent depends on the model and the number of update information. Thus, the experiment data and results are sent in the stream format. Acknowledgement messages are sent in server's binary format. The client performs all conversions between physical representations. This approach keeps the server independent of the processor type. Any message sent to client carries a field identity from which type of

the processor can be determined. The information in the message is converted into the server's format before handling the request. Events are also sent in binary format. The main reason for this approach apart from the conversion overhead is that it is possible to send more events in one message. Server process maintains one additional window of batches for each type of processor whose data representation is different from its own format. Converted copies of each new event are stored in these windows. Events from the appropriate window are sent to the client.

3.4. SC Simulation Design Issues

3.4.1. The Event Generation

During the simulation study, the client process generates event data. A single event data set consists of the *event type*, the *node number* on which event occurs, the *class* for arrival events, and four uniform random numbers as shown in Figure 3.2. This is accomplished by generating events at maximal possible rate. Each event is classified by using event alias method which keeps a distinct entry for each individual arrival event and a single entry for a departure event to each node. Two random numbers are used to generate events; one for the event time and one to determine the entry in the table.

Event Type	Node Number	Class	R1	R2	R3	R4
------------	-------------	-------	----	----	----	----

Figure 3.2. Event data set

3.4.2. Model Covered by the Engine

Each variant in this study, is a network of queues and nodes connected with links. Queueing models covered in the system are

- Markovian Models with
 - multiple entity classes
 - multiple parallel servers
 - service with interruption
 - assembling and disassembling services

- Advanced Markovian models with bulk arrivals and bulk service,
- Queues with FIFO and priority discipline
- Non-Markovian Models G/G/1, M/G/1 and G/M/1

3.4.3. Thinning and State Update

In SC approach to simulation, each variant consists of a thinning process, feasibility check and a state update process. The thinning process can reduce the SC rate to the local rate. The reduction is based on the information related to (1) local rate, *rate thinning*, and (2) current state of the experiment, *state-dependent thinning*. The rate thinning simply thins the SC rate to local rate while state-dependent thinning considers the current state of the experiment such as the state of the servers and the queue in corresponding variant. Four random numbers associated with each event are used for thinning as well as the other purposes as given in Table 3.2.

Table 3.2. Usage of random numbers

Random Number	Arrival Event	Departure Event
1	Thinning (Exponential)	
2	Thinning(if Hyperexponential)	
3	Selection of Parallel Server	
4	Bulk Arrivals	Routing

The rate thinning mechanism is implemented as simple filters, *rate filters*, connected to each node for each event type. Each node and queue structure is associated with a set of *feasibility filters* for state dependent thinning. They are applied in different order to an arrival event than to a departure event.

An arrival event goes first through the rate thinning process in which it is either rejected or the number of arriving entities is determined. Since the maximal of the arrival rate is used for the event, this event has to be thinned. In addition for bulk arrivals, the rate filter also determines the number of incoming entities from the batch size probabilities

using the fourth random number. The feasibility of an arrival event with respect to current state of the queue is then checked during the state-dependent thinning process. An arrival event is feasible provided that the queue has enough space to accommodate the incoming entities.

For a departure event, the feasibility filter determines the server on which the departure will occur. A departure event is filtered out if the corresponding server is either empty or already blocked with an entity in the previous departure process. The third random number is used to determine by the feasibility filter of the node with multiple parallel servers to determine on which the departure will occur. The class of the event is determined as the class of the entity in service. The current event is either rejected after the feasibility check or goes through rate thinning. The accepted events cause state update in the corresponding node and each node is responsible for updating its state for accepted events.

3.4.4. Components of the Model

Each node consists of rate and state thinning components and state update mechanisms. A basic structure of the node holds common properties such as number of servers, filters, state transition and routing information. Based on the object oriented design, new node classes are derived from the base node for more sophisticated services, such as preemptive service, batch, assemble or disassemble. The new derived class adds the attributes and redefines methods, which are necessary for that specific state update.

Queue is represented as a separate object with its properties such as queue length and size and its necessary primitive functions. Priority queue is derived from the basic queue by redefining the way data are stored and retrieved. They are instantiated and linked to a node.

Statistics are collected in separate object which is instantiated and linked to every node or queue. The standard statistics are, server utilization, queue length, waiting time, blocking and balking rates, current and the maximum number of entities, system time, and throughput.

4. LOAD BALANCING

An important factor affecting the performance of a simulation analysis in a distributed environment is load balancing. The objective of load balancing is to minimize the completion time of the entire set of simulation experiments. The simulation completion time, which is considered as the performance measure, depends on the assignment of variants to workstations. This chapter starts with a brief discussion of the time requirements of the distributed simulation using SC method. In the following sections, we present two different load balancing techniques: (1) a static load balancing that uses estimated cost of each variant based on the “acceptance rate”, the number of events accepted and (2) a dynamic load balancing that migrates variants between workstations, based on their estimated performance during the simulation process. The remaining sections discuss numerical results obtained from real-time simulations of simple queueing models on a network of workstations. They are used to investigate the speedup and the efficiency of the load balancing techniques.

4.1. Total Time Requirement of Simulation

The total simulation time, TST , can be broken into three terms:

$$TST = T_A + T_S + T_F \quad (4.1)$$

T_A : setup time composed of data input of model specifications, load balancing computations, and model transfer to the servers

T_S : simulation execution time

T_F : the time spent for collecting results.

The sum of T_A and T_F is negligible with respect to total processing time. Therefore, the total simulation time becomes close to T_S . Besides, as it can be seen in Figure 4.1, T_S is actually composed of two components;

T_{C_i} : the communication time necessary for the transfer of the batch i between client and server processes, and

T_{E_i} : the processing time of the batch i .

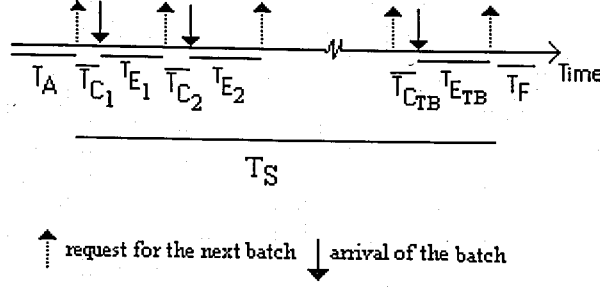


Figure 4.1. Timing diagram of the simulation execution

The total simulation time can then be approximated as

$$TST \approx \sum_{i=1}^{TB} \{T_{C_i} + T_{E_i}\} \quad (4.2)$$

where TB is the number of batches, and each batch corresponds to the processing of one set of events.

Each processor may send the request for the next batch before starting the processing of the current batch. Hence, there is a possible overlapping between the processing time of the current batch and the transmission time of the next batch as seen in Figure 4.2.

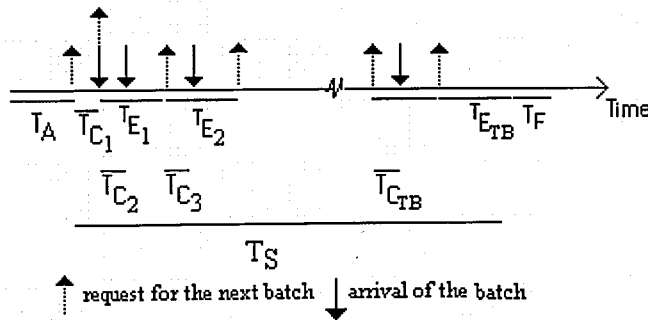


Figure 4.2. Timing diagram with communication overlapped

and the total simulation time then becomes,

$$TST \approx \sum_{i=1}^{TB} \max\{T_{C_{i+1}}, T_{E_i}\}. \quad (4.3)$$

Since the number of variants is much greater than the number of workstations and the network is assumed to be less congested, $T_C \ll T_E$, time spent in a simulation will be dominated by the processing time T_E . The total processing time requirement of the variants assigned to any workstation is proportional to its total load, TL , which will be used to approximate T_E .

4.2. Estimating the Load of Variants

In SC simulation, each event is subject to two distinct sets of operations; thinning (rate thinning and state thinning) and the state update operations. Let A and B be the average computational load due to thinning and state update respectively during the whole simulation process and λ_i be rate for the event i in SC simulation. All events are checked for feasibility but only the surviving ones, whose probability to survive is referred to as the acceptance rate, r_i^k , will cause the state update in the variant k . For K different event types, T_k , the load of variant k , can be expressed in terms of the parameters defined above as

$$T_k = B \sum_{i=1}^K r_i^k + \Lambda A \quad (4.4)$$

where Λ is the total event rate for the system.

The total load for E variants is,

$$TL = \sum_{k=1}^E T_k = B \sum_{k=1}^E \sum_{i=1}^K r_i^k + E\Lambda A \quad (4.5)$$

and the load, L_w , allocated to any workstation w which executes the set S_w of M_w variants assigned will be

$$L_w = B \sum_{k \in S_w} \sum_{i=1}^K r_i^k + M_w \Lambda A. \quad (4.6)$$

4.3. Static Load Balancing

The basic idea in our load balancing strategy is to balance the total load of each workstation so that they spend more or less the same time to complete the simulation runs. For this purpose, we consider two different cases; namely, the case in which

- the communication overhead is overlapped and
- the communication overhead is included in the load balance calculations

4.3.1. Case 1. Load Balancing with Communication Overhead Overlapped

In case 1, load balancing can be achieved in our problem by simply distributing variants into workstations in proportion to their relative processing power, in such a way that each subset will progress at approximately the same rate.

Load balanced distribution aims to balance the number of thinning and state update operations in each of the workstations. The load formula in (4.6) has two components. The ratio of A to B is hard to define, and furthermore, it is implementation dependent. Hence, it is more reasonable to satisfy the proportionality for each component. ie., distribute experiments among processors so that

- the number of experiments, and
- the total number of accepted events in each workstation

are proportional to its speed.

Considering the second component, the number of operations necessary for each variant (Λ events are processed in each variant) is the same for thinning. Therefore, load balancing due to thinning is based on the number of variants allocated to each workstation. The number of variants, M_w , allocated to the workstation w must be proportional to its processing power, P_w

$$M_w = \left\lfloor \frac{P_w E}{PP} \right\rfloor \quad (4.7)$$

where PP is the total processing power of the network of workstations $PP = \sum_{w=1}^P P_w$. The

measurement of P_w is further discussed in Section 4.5. Since it may not be possible to allocate all the variants due to their integrity, the remaining experiments are allocated using the longest processing time (LPT) heuristic (Dobson, 1984) such that each excess variant will be assigned to the processor w with,

$$\min_{w=1 \dots P} \frac{M_w + 1}{P_w}. \quad (4.8)$$

Similarly, to balance the load due to state update, the total number of accepted events of variants allocated to any workstation w must also be proportional to P_w . To simplify, let

$$R_k = \sum_{i=1}^K r_i^k \quad k=1 \dots E \quad (4.9)$$

be the total number of accepted events per unit time in variant k , then the balance formula becomes

$$\frac{\sum_{k \in S_w} R_k}{P_w} \approx \frac{\sum_{k=1}^E R_k}{PP}. \quad (4.10)$$

Each variant k with R_k is to be assigned to one of the workstations. The objective becomes to find a partition of the set of variants such that the maximum ratio of the total number of events accepted to the processing power in each workstation, is minimized. This can be stated more formally as:

Given a set, S , consisting of E variants, each with estimated computational cost R_k and a set of P workstations with powers $P_1, P_2, \dots, P_P$, where $(E > P)$ select a disjoint partition of $S = S_1 \cup S_2 \cup \dots \cup S_P$ such that

$$\min \left\{ \max_{w=1 \dots P} \left\{ \frac{\sum_{k \in S_w} R_k}{P_w} \right\} \right\} \quad \text{s.t. } \|S_w\| = M_w \text{ for } w=1 \dots P. \quad (4.11)$$

The load balancing is reduced to the so-called minimum makespan or multiprocessor scheduling problem in which the number variants to be allocated to each workstation is determined in advance. The problem is known to be NP complete, and the LPT algorithm can be used to solve this problem. The idea in determining the number variants to be allocated to each workstation before partitioning, is to require faster workstations to get any of the remaining variant k since its R_k value is small compared to Λ and ΛA is expected to dominate $B R_k$.

4.3.2. Computation of Acceptance Rates

As stated before, the rate of event i which the global rate is thinned to in variant k , is the acceptance rate r_i^k . Acceptance rate is calculated from the parameters of the distribution assuming that the system is in steady state equilibrium.

For arrival events, the acceptance rate is the rate of the distribution. Therefore, for an exponential distribution

$$r_i^k = \lambda_i^k \quad (4.12)$$

However, the above computations are only valid for arrival events. A departure event is accepted when an entity exists in the node. In order to compute the number of departure events accepted, one has to find the mean number of visits to any node, denoted by e_i^k , to which the acceptance rate of any departure event i is equal. Obviously, for simple systems, the acceptance rate for a departure event of an entity depends on its arrival rate. For the M/M/1 system, $e_i^k = \lambda_i^k$ of the arrival event i . Considering a nonexponential distribution with mean μ and variance σ^2 , the acceptance rate of the distribution is $e_i^k = 1/\mu_i^k$.

As to a queueing network consisting of N nodes, the row vector of acceptance rates for the exogenous arrival events in experiment k is, $r^k = (r_1^k, r_2^k, \dots, r_N^k)$. In order to compute acceptance rates for departure events, the vector of mean number of visits in experiment k , $e^k = (e_1^k, e_2^k, \dots, e_N^k)$ can be represented as solutions of the system $e = r + ep$ where $p^k = [P_{nxm}]$ denotes the transition probability matrix. Individually, each e_i^k is equal to

$$e_i^k = r_i^k + \sum_l^N e_l^k p_{l,i}^k \quad (4.13)$$

Unfortunately, the relation $e = r + ep$ must be separately solved for each class if the network has multiple entity classes.

4.3.3. Case 2. Load Balancing with Communication Overhead Non-Overlapped

This case differs from the previous one in the way the processing power values of workstations are computed as follows;

Let PC_l and CC_l be the average execution time of a batch for one variant in the first server and the required communication time for transferring one batch between the client and server. The total processing requirement of the simulation can be written in terms of the above parameters as

$$TST \approx TB * (CC_1 + E * PC_1). \quad (4.14)$$

On the other hand, the average computation times for other servers can be computed from the processing power ratios such that

$$PC_w = PC_1 \frac{P_w}{P_1}. \quad (4.15)$$

Let F_w be the fraction of set of variants assigned to server w , that can complete the execution of one batch, then the equations that govern the relations among various variables and parameters in the system are the following

$$T_w = CC_w + F_w * E * PC_w \quad w=1...P \quad (4.16)$$

and the fraction of number of variants should sum to 1. Since we want to minimize the maximum completion time, this becomes

$$\min \max_{w=1..P} (T_w). \quad (4.17)$$

The completion time of the simulation is the time when last processor finishes processing and the minimum finish time is achieved when all processors stop at the same time that is when:

$$T_1 = T_2 = \dots = T_P. \quad (4.18)$$

The optimal values for F can be calculated by solving the following set of equations;

$$T_w = T_{w+1} \quad \text{for } w=1...P-1. \quad (4.19)$$

Finally, the new processing powers for the non-overlapped simulation becomes $P_w = F_w$. These new processing power values are used in the heuristic algorithm that follows to distribute variants.

4.3.4. Heuristic Algorithm for Load Balancing

The load balancing algorithm used for distribution of E experiments has three steps as presented in Figure 4.3.

Step 1: Compute the maximum number of experiments, M_w , to be allocated to processor w

$$\text{Compute } M_w = \left\lfloor \frac{E P_w}{PP} \right\rfloor \text{ for each } w=1 \dots P \quad \text{Let } Z = E - \sum_{w=1}^P M_w$$

If $Z > 0$ increment M_w by 1 for $\min_{w=1 \dots P} \left\{ \frac{M_w+1}{P_w} \right\}$ for $i=1 \dots Z$

Step 2: Compute R_k the cost of an experiment k . Distribute the experiments using LPT scheduling algorithm

Sort E experiments in a nondecreasing order of R , such that $R_1 > R_2 > \dots > R_E$

Let $S_w = \emptyset$, $H_w = 0$ (workload assigned to processor w), and the number of experiment currently assigned to processor w , $N_w = 0$ for $w=1 \dots P$

Repeat the following for $k=1 \dots E$

Find the processor w for which $\min_{w=1 \dots P} \left\{ \frac{H_w + R_k}{P_w} \right\}$ and $N_w < M_w$

set $S_w = S_w \cup \{k\}$, $N_w = N_w + 1$, and $H_w = H_w + R_k$

Step 3: This step does workload tuning. The basic idea in this step is to reduce the maximum workload by a series of *pairwise* exchange of experiments between the most heavily loaded processor and the other processors.

Repeat the following steps until no exchange is possible

Let $D_w = H_w - \frac{P_w}{PP} TL$ for $w=1 \dots P$

Sort processor loads in a decreasing order of D .

Let m be the processor with $D_m = \max_{w=1 \dots P} (D_w)$ to which the set S_m be assigned.

Repeat the following steps for each processor $w=1 \dots m-1$,

Let $a = (D_m - D_w) / 2$

If there exist two experiments x and y such that $x \in S_m$ and $y \in S_j$

then set $d = \max_{x \in S_m, y \in S_j} \{R_x - R_y\}$ for $0 \leq d \leq a$

$$S_m = S_m \cup \{y\} - \{x\}, S_j = S_j \cup \{x\} - \{y\}, H_m = H_m + d, H_j = H_j + d$$

Figure 4.3. Static load balancing algorithm

4.3.5. Expected Speedup and Efficiency

Performance evaluation of parallel and distributed algorithms is given in terms of the speedup. The speedup, S_{PP} , is defined as

$$S_{PP} = \frac{T(1)}{T(PP)} \quad (4.20)$$

where $T(1)$ is the execution time on a *specified* workstation for which the processing power is assumed to be 1. All other processing powers are normalized relative to that workstation, and $T(PP)$ is the simulation time on P workstations with a total processing power PP .

With the ideal distribution, the base workload L_S assigned to the specific workstation s can be computed as

$$L_S = \frac{TL}{PP} \quad (4.21)$$

The load balancing algorithm produces a distribution with maximum computational load on P processors is designated as L_{max} . L_{max} is the base workload multiplied by β , $\beta > 1$, the load imbalance factor: ie., β is the ratio of the maximum load to the base one,

$$L_{max} = \max_{w=1 \dots P} \left\{ \frac{L_w}{P_w} \right\} = \beta \cdot L_S \quad (4.22)$$

Therefore, by rewriting L_S , we get,

$$L_{max} = \frac{\beta \cdot TL}{PP} \quad (4.23)$$

and the expected speedup becomes

$$S_{PP} = \frac{1}{\beta} PP \quad (4.24)$$

It has basically a linear form $c.PP$ for some constant $c > 1$.

Another measure, the efficiency which defines how well the particular algorithm has been paralleled is given by

$$Efficiency_{PP} = \frac{T(1)}{T(PP) * PP} \quad (4.25)$$

4.4. Dynamic Load Balancing

Dynamic load balancing attempts to maximize utilization of the environment at all times (or at some checkpoints) by migrating the tasks among processors during the runtime. One of the policies used in dynamic load balancing is the centralized one, in which a dedicated processor is responsible for maintaining a global state of the system. This policy can easily be applied to standard clock simulation without any additional overhead due to information collection since the client has information about the current progress made by each workstation as well as the number of variants assigned. The client computes periodically the performance of each workstation from this information. It estimates the completion time of the entire simulation process. When a load imbalance is detected, it decides whether the migration of a certain number of variants is profitable to reduce the total simulation time. Hence, the objective of our dynamic load balancing is to coordinate all workstations on the unprocessed work by redistributing it among the workstations proportionally to their present performance so that they all spend more or less the same time to complete the simulation.

Dynamic load balancing requires both a metric to determine the system load as well as a mechanism for controlling process migration. The simulation using the SC method is mainly processing events for each variant created by a central controller, i.e. the client process. Batch processing rate of any particular workstation can be implicitly used a metric to determine its load. A discrepancy in the respective completion times of any two workstations will result in an increase in the number of batches stored by the client process. Hence, the difference between the highest and the lowest numbered batches

processed will reflect how well the loads are balanced and we try to limit the difference in between furthest ahead workstation and the furthest behind workstation processing. However, load balancing based on only the difference of batch numbers may result in very frequent load balancing computations or migrations due to fluctuations in the processing power of workstations. To remedy this problem, a checkpoint concept is introduced in the load balancing computations. After the migration has completed, the controller waits until the next checkpoint before it examines the load. Thus, the dynamic load balancer lets the system regain its equilibrium before deciding to migrate another set of variants. In addition, we define an initial start of the load check.

Let Δt be the time spent in the interval between two consecutive migration points during which each workstation w has processed the set of batches $[Y_w^{start}, Y_w^{end}]$. X_s ($X_s > 0$) is number of variant set received from another workstation s at the beginning of this interval. Note that Y_s^{start} is different for each set and the receiving workstation is forced to process these variants. Then, the workload assigned, L_w , in Δt can be defined as the product of the number of variant in each particular set and the number of batches corresponding to that set:

$$L_w = M_w * (Y_w^{end} - Y_w^{start}) + \sum_{s \neq w} X_s * (Y_w^{end} - Y_s^{start}) \quad \text{for } w=1 \dots P, s=1 \dots P. \quad (4.26)$$

After having processed the batch Y_w^{end} , the remaining workload on workstation w , RL_w , becomes

$$RL_w = (M_w + \sum_{s \neq w} X_s) * (TB - Y_w^{end}) \quad (4.27)$$

and its performance can be characterized in terms of the ratio

$$V_w = \frac{L_w - RL_w}{\Delta t_w} \quad \text{for } w=1 \dots P. \quad (4.28)$$

We then estimate the expected finish time, T_w , for each workstation as

$$T_w = \frac{RL_w}{V_w} \quad \text{for } w=1 \dots P. \quad (4.29)$$

If the difference between the expected completion time of the furthest ahead and furthest behind workstations exceeds some predefined threshold value,

$$\max_{w=1 \dots P} (T_w) - \min_{w=1 \dots P} (T_w) > \text{threshold} \quad (4.30)$$

then the load balancing between any pair of workstations is considered, and we compute the ideal amount of workload that must be assigned to any particular workstation using the parameters measured below.

The cumulative remaining workload, CR , current processing power of the computing environment, PP , are given by

$$CR = \sum_w RL_w \quad \text{and} \quad PP = \sum_w V_w \quad \text{for } w=1 \dots P. \quad (4.31)$$

In order to balance the remaining workload among workstations, the appropriate portion of the remaining work, say NW_w , which should be assigned to the corresponding workstation w , can be calculated as

$$NW_w = \frac{CR * V_w}{PP} \quad \text{..for } w=1 \dots P \quad (4.32)$$

and the number of variants that must be migrated to or from workstation w is determined as

$$X_w = \left\lceil \frac{NW_w - RL_w}{TB - Y_w^{end}} \right\rceil \quad \text{for } w=1 \dots P \quad (4.33)$$

Each workstation with $X_w > 0$ is overloaded, hence, becomes a candidate sender, similarly, more variants can be run on the workstation with $X_w < 0$. Ideally, we would like X_w for most of the workstations to be zero, as this will indicate these workstations are being utilized to finish the simulation process on time. Candidate matching mechanism is simple; the best-fit heuristic is used to find the number of variants to be migrated from senders to receivers. Migration between candidates is allowed if it fulfills the following inequalities;

- i. the sender is behind of the receiver in terms of simulated events

$$Y_S < Y_R \text{ and} \quad (4.34)$$

- ii. communication cost between workstations S and R does make the load balancing profitable, i.e.

$$\frac{(M_S - X_S) * (TB - Y_S^{end})}{V_S} > \frac{(M_R * (TB - Y_R^{end}) + X_S * (TB - Y_S^{end}))}{V_R} + T_{S,R} * X_S \quad (4.35)$$

Based on the definitions given above, summary of the dynamic load balancing algorithm steps is given in Figure 4.4.

```

if( First batch number - last batch number > Distance
   AND migration point is reached)
{
  Compute the expected finish time for each workstation;
  if (Tmax - Tmin > threshold)
  {
    Compute total remaining work;
    Compute repartition;
    Create the list of possible senders and receivers;
    Do matching by best fit heuristic and create a migration matrix
    by checking feasibility of each possible match;
    Set the new migration point;
    Adjust batch information for the receivers;
  }
}

```

Figure 4.4. Basic steps of the dynamic load balancing algorithm

4.5. Experimental Network Environment

Runs have been carried out using eight workstations in the Campus network. The heterogeneous simulation testbed consists of two IBM, three Silicon Graphics, one HP and two DEC machines. Workstations are connected to three different local subnetworks in which they are all loosely coupled via Ethernet. The client and one of the servers are concurrently run on the first workstation. The timing results can be affected by the load on the workstations. This effect is minimized by running simulation experiments when the workstations are idle. In order to compute the processing power of each workstation a "benchmark" simulation is concurrently run. The program simulates N replicas of an M/M/1 system with $\rho=0.5$ and 10 initial entities. The processing power, P_w , for each workstation is defined as;

$$P_w = \frac{N}{ExecutionTime_w} \quad (4.36)$$

where $ExecutionTime_w$ corresponds to T_S of workstation w . Characteristics of each workstation used in the runs, is given in Table 4.1. N is specified as 50 for this investigation.

Table 4.1. Characteristics of workstations

Workstation #	Measured P_w	Type /OS	Com. Cost (msec)
1	57.36	IBM AIX 4.01	2.0
2	47.72	SG Irix 5.3	10.0
3	48.64	SG Irix 6.2	10.0
4	28.17	HP 715 Hp-UX 9.05	10.0
5	60.43	IBM AIX 4.01	9.0
6	22.02	DEC5500 Ultrix V 4.4	11.0
7	15.87	DEC Ultrix V 4.2A	15.5
8	127.74	SG Irix 6.2	15.0

4.6. Experiments by Varying Number of Workstation

In this section, we study the performance of the distributed simulation as the number of workstations varies. Total simulation time (TST), the time elapsed between the start and the termination of the client, is used in all calculations and showed in the tables.

4.6.1. Static Load Balancing

Four different models are simulated on first seven workstations whose parameters are given in Table 4.1 to observe the speedup. The total simulation time on single workstation is chosen to be 10 minutes. A total of 120,000 (TB=3000) events are simulated in all experiments for each model, except G/G/1 for which 200,000 events are simulated. We believe that this length is enough to demonstrate the effect of the distributed simulation. We simulated roughly 600 variants for three different Markovian models and 500 variants of the G/G/1 system. Their characteristics are as follows:

- i. The M/M/1 system with $\lambda = [1.00 , 6.99]$ and $\mu = [2.00 , 7.99]$. Both rates are incremented by 0.01.
- ii. The two-node Jackson network shown in Figure 4.5 in which $\lambda=0.5$, $\mu_1 + \mu_2 = 4$, $\mu_1 = [0.505 , 3.495]$, and $\mu_2 = [3.495 , 0.505]$. μ_1 is incremented by 0.005 and μ_2 is decremented by 0.005.

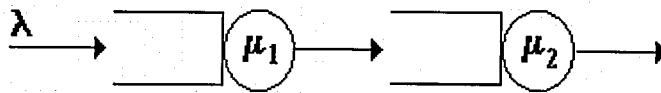


Figure 4.5. Two-node Jackson network

- iii. A 4-node and 7-arc network with exponential arrival and services. There are two classes of entities with arrival rates $\lambda_1 = [1.00 , 3.995]$ and $\lambda_2 = [1.00 , 1.994]$. The service rates μ_1 , through μ_2 are as depicted in Figure 4.6 (The second class parameters are shown below the corresponding component and in *italic*).

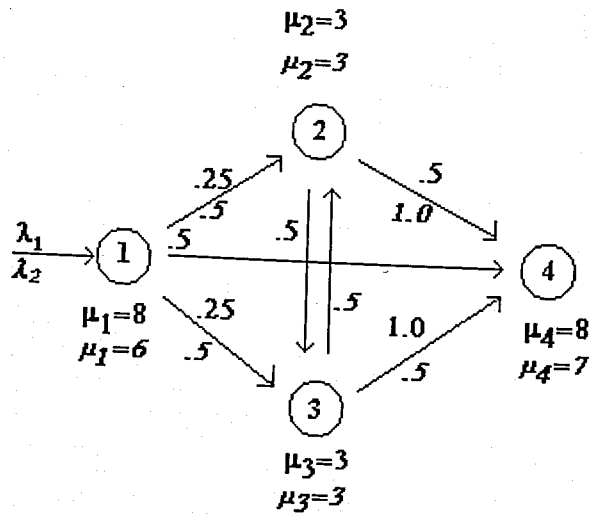


Figure 4.6. 4-node and 7-arc network with two classes of entities

- iv. G/G/1 system, for which the parameters of the arrival distribution is $\mu = [4.00, 8.99]$ and $\sigma^2 = 1$ and the service distribution with $\mu = [2.00, 7.988]$ and $\sigma^2 = 1$. For each of the variants, the mean of the arrival distribution is incremented by 0.01 and that of the service distribution is incremented by 0.012.

The simulation of four different models produced TST results reported in Table 4.2

Table 4.2. Results obtained for four models experimented

PP	Model 1		Model 2		Model 3		Model 4	
	Time(sec)	Efficiency	Time(sec)	Efficiency	Time(sec)	Efficiency	Time(sec)	Efficiency
1.000	543.7		444.6		620.7		357.6	
1.832	309.5	95.9%	254.4	95.4%	342.3	98.9%	204.4	95.5%
2.689	210.3	96.1%	176.0	93.9%	241.6	95.8%	142.1	93.6%
3.171	183.6	93.4%	151.9	92.3%	206.8	94.7%	125.7	89.7%
4.225	139.3	92.4%	114.5	91.9%	158.7	92.6%	100.0	84.6%
4.608	130.2	90.6%	107.6	89.7%	149.7	90.0%	95.1	81.6%
4.885	124.1	89.7%	105.2	86.5%	144.4	88.0%	93.5	78.3%

The total simulation time gets smaller as the number of workstation (total processing power) increases. Figure 4.7 presents the speedup curve where each point corresponds to one additional workstation. A sublinear speedup is observed for the M/M/1 system. Similar kind of behavior is also obtained with other three models tested. This is expected since the SC simulation of a network of queues proceeds in the same manner as for a single queue.

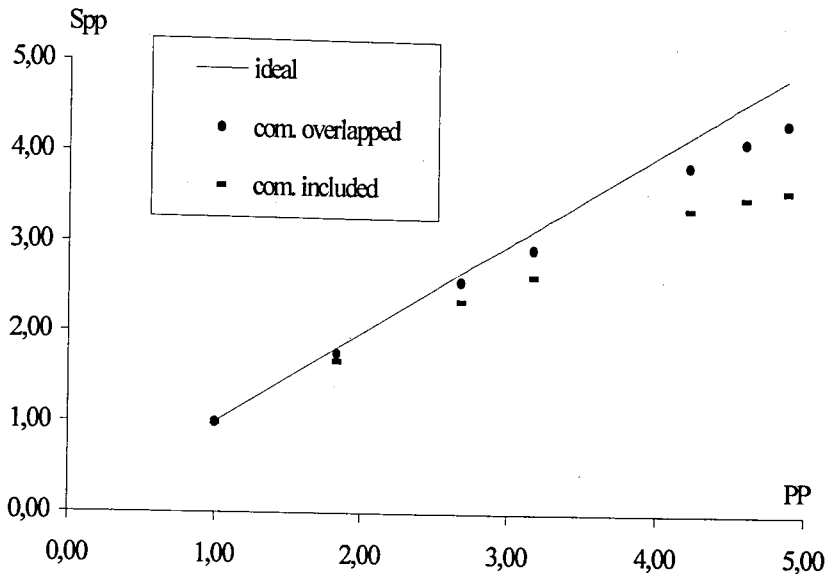


Figure 4.7. The speedup curve for the M/M/1 system with static load balancing

There is a deviation from the ideal as the number of workstations increase. There are two possible reasons for this: First the error made in estimating the processing speed of each workstation, naturally, the relative accuracy of estimation decreases as more workstation are included in the run. Second, an increase in the number of workstation (dispersion in the processing powers) leads to a slightly faulty load distribution and the error becomes more significant since the simulation time gets smaller. The faster drop in the efficiencies for Model 2 and Model 4 is because of the small acceptance rates and note that the simulation time obtained on one workstation is less than the other two models. The efficiency that is greater than 86 per cent is observed for all three Markovian models and 78 per cent for the G/G/1 model on seven workstations.

Although communication in a network environment is expensive, our runs (for up to seven workstations) indicate that the communication has not been a dominating component

when the communication is overlapped. This is normal because the number of variants is high and the transmission time of next frame is overlapped with the computation time of the previous one. The communications requirement of the simulation depends only on the number of events to be simulated. The time spent for communication (requesting and reading batches-no batch is lost during variants) did not significantly change with the increase in the number of workstations (approximately 4 seconds for all runs, with highest value 4.38 and lowest value 3.87). Although the ratio of communication cost to computation cost increases as more workstations are added, its highest value is less than 4 per cent for the M/M/1 system and 9 per cent for the G/G/1 system on seven workstations.

As it is clear from Figure 4.7, the speedup lines separates from the linear speedup line (ideal and as well as the com. overlapped figures) as the communication cost is not overlapped. This is because each additional workstation increases total workload by a constant load (10 msec for each batch) due to communication although the computation cost decreases. Therefore, the ratio of communication to computation increases. Note that the separation becomes evident as more workstations are added. The efficiency of the M/M/1 system simulation reduces to 76.8 per cent from 89.7 per cent that is obtained for "com. overlapped" case as given in Table 4.3.

Table 4.3. Experimental results of the M/M/1 system for "Communication Included" case

PP	Time (sec.)	Efficiency
1.000	543.7	
1.832	318.2	93.3%
2.689	231.9	87.5%
3.171	200.6	85.5%
4.225	157.5	81.7%
4.608	149.9	78.7%
4.885	144.9	76.8%

As to the time spent for generation and dispatching of events in the server process, it is proportional to the number of events generated as well as the number of workstations used in the run. During the simulation of the M/M/1 system in a single workstation, 120000 events are created and sent in 3.5 seconds while this value rises to 9.3 seconds when the simulation is done on seven workstations (similarly from 5.6 to 15.8 for the G/G/1 system for 200000 events).

Finally, two negligible figures are the overhead for collecting simulation results which is less than 2 seconds and the time spent in setup phase is less than 1 second for all variants.

4.6.2. Comparison with other Static Allocation Strategies

In this section, we compare our load balancing results with several different allocation strategies. They are defined as:

- A1 : proposed load balancing allocation strategy
- A2 : **random splitting** (assignment) in which each workstation w is allocated a variant with the probability P_w / PP . This is the simplest way to conduct simulation experiments and has been widely used.
- A3 : **linear forward assignment** from workstation 1 through 7. First M_1 variants are allocated to workstation 1, next M_2 variants to workstation 2 and so on.
- A4 : **linear reverse assignment**, similar to A3, but variants are assigned in the reverse order, i.e., from workstation 7 through 1.

Table 4.4. Results of different allocation strategies for the M/M/1 system

Number of Workstations.	PP	A1	A2	A3	A4
2	1.832	303.35	317.77	423.76	355.71
3	2.680	206.28	215.31	281.89	258.89
4	3.171	179.52	193.26	262.42	226.00
5	4.225	134.19	145.26	184.24	178.20
6	4.608	125.53	132.20	174.76	164.00
7	4.885	117.91	127.58	166.20	158.60

The results for the M/M/1 system, in Table 4.4 show the longest execution time (T_s) obtained for A1 and three different allocation strategies as well as the percentage of deterioration in between. (For A2, the smallest of five runs is reported as the typical value) A1 outperforms all other strategies. Therefore, the acceptance rates plays an important role in distributing the load of our M/M/1 model (on the average 53 per cent of the events are simulated). As it can be seen, A1 is marginally better than A2, the percentage of

deterioration of A1 over A2 varies only from 1 per cent to 8 per cent. The A2 algorithm produces a distribution in which one or two workstations are slightly overloaded in terms of the number of variants and events to be simulated.

The last two allocation strategies A3 and A4 are tested to examine the worst case behavior. They are significantly worse than any one of the previous strategies with a percentage of deterioration varying from 17 per cent up to 46 per cent although the number of variants assigned to each workstation is proportional to its power. The longest execution time is determined by the most heavily loaded workstation which is the first and the last one in A3 and A4 respectively.

4.6.3. Effect of Communication Overlapping

In order to find out the effect of communication overlapping, we run simulations by varying the number of variants and the number of batches. Since the accepted number of events may effect the simulation time, the same M/M/1 system with average acceptance rate of 56 per cent is used in all runs. Note that this observation in this section is valid for our network environment with specification given in Table 4.1.

In this section, we fixed the number of events and varied the number of variants. We simulated five set of M/M/1 system, (multiple of 25 variants in each set) on up to five workstations. With the first set containing 25 variants, the simulation time did change by additional workstations, resulting in a constant speedup value that is approximately equal to 1.30. By increasing the number of variants to 50, we observed a decrease in the simulation time (up to 3 workstations) then it remained constant. The reason for a constant speedup value is that the communication time is high relative to the computation time that is further reduced due to additional workstations. For the set containing more than 75 variants, we see a better increase in the speedup for four workstations ($PP=3.171$) as depicted in Figure 4.8.

The change in the simulation time can be seen in Figure 4.9. The simulation execution time becomes more “computation intensive” after approximately 60 variants.

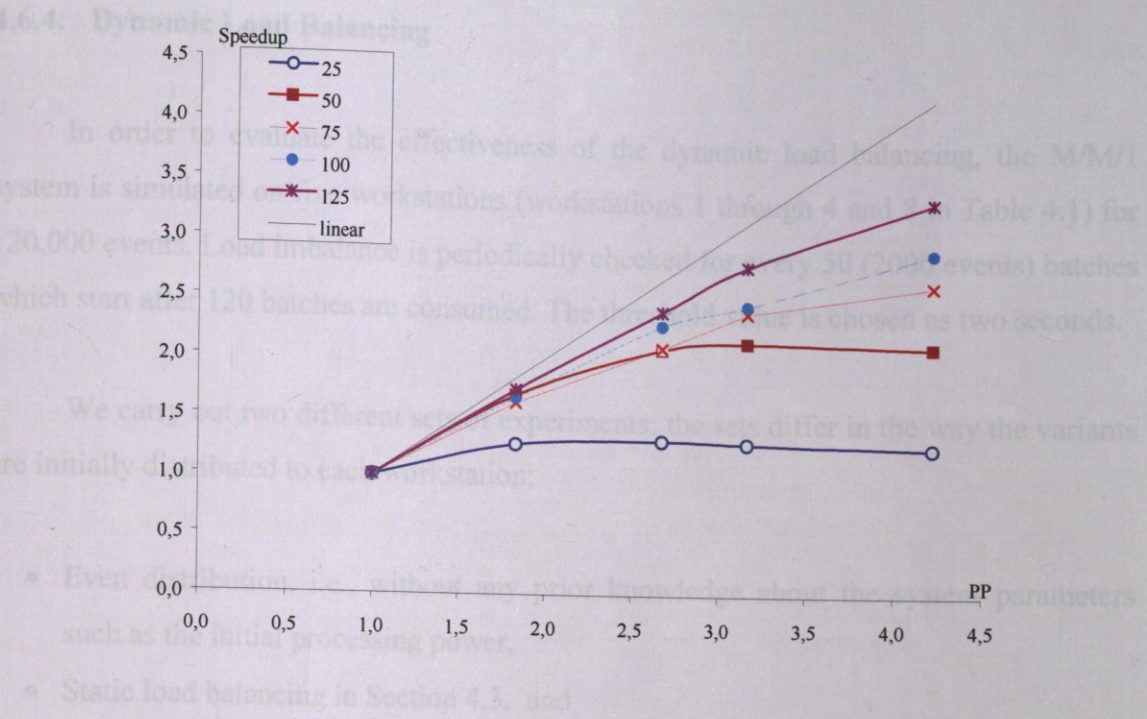


Figure 4.8. Speedup curve vs processing power for different number of variants

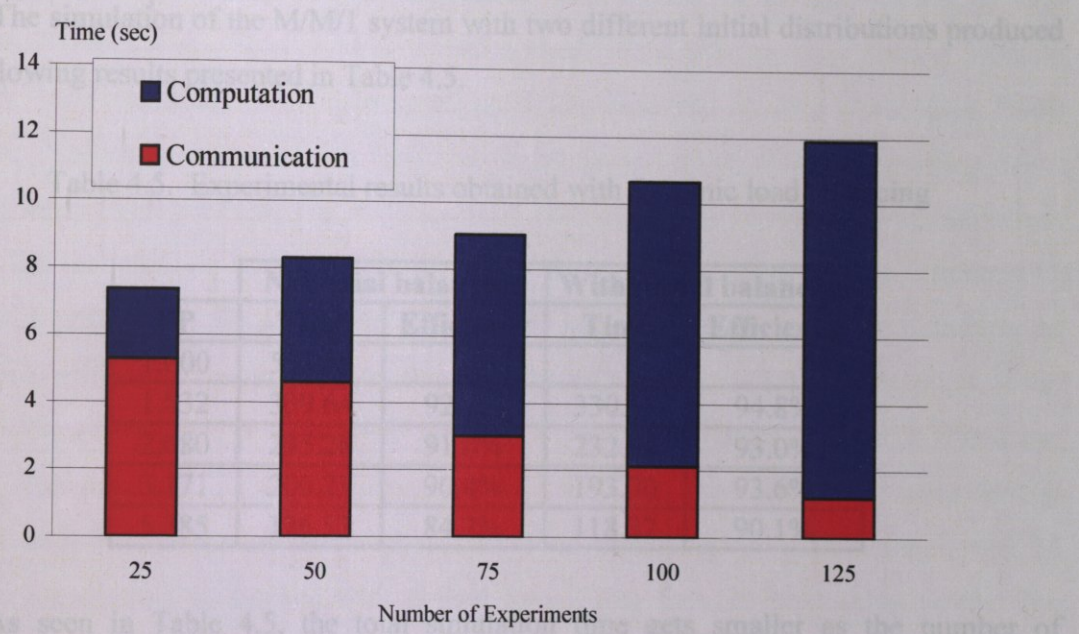


Figure 4.9. Computation and communication levels

4.6.4. Dynamic Load Balancing

In order to evaluate the effectiveness of the dynamic load balancing, the M/M/1 system is simulated on five workstations (workstations 1 through 4 and 8 in Table 4.1) for 120,000 events. Load imbalance is periodically checked for every 50 (2000 events) batches which start after 120 batches are consumed. The threshold value is chosen as two seconds.

We carry out two different sets of experiments; the sets differ in the way the variants are initially distributed to each workstation;

- Even distribution, i.e., without any prior knowledge about the system parameters such as the initial processing power,
- Static load balancing in Section 4.3. and

the distribution is then balanced through migration during the execution as necessary.

The simulation of the M/M/1 system with two different initial distributions produced the following results presented in Table 4.5.

Table 4.5. Experimental results obtained with dynamic load balancing

PP	No initial balancing		With initial balancing	
	Time	Efficiency	Time	Efficiency
1.000	573.88			
1.832	339.64	92.2%	330.53	94.8%
2.680	235.28	91.0%	232.22	93.0%
3.171	200.23	90.4%	193.30	93.6%
5.385	126.52	84.2%	118.32	90.1%

As seen in Table 4.5, the total simulation time gets smaller as the number of workstations (i.e., total processing power) increases. Initial distribution of variants according to a performance rating of workstations results in a slight improvement about 2 per cent to 6 per cent.

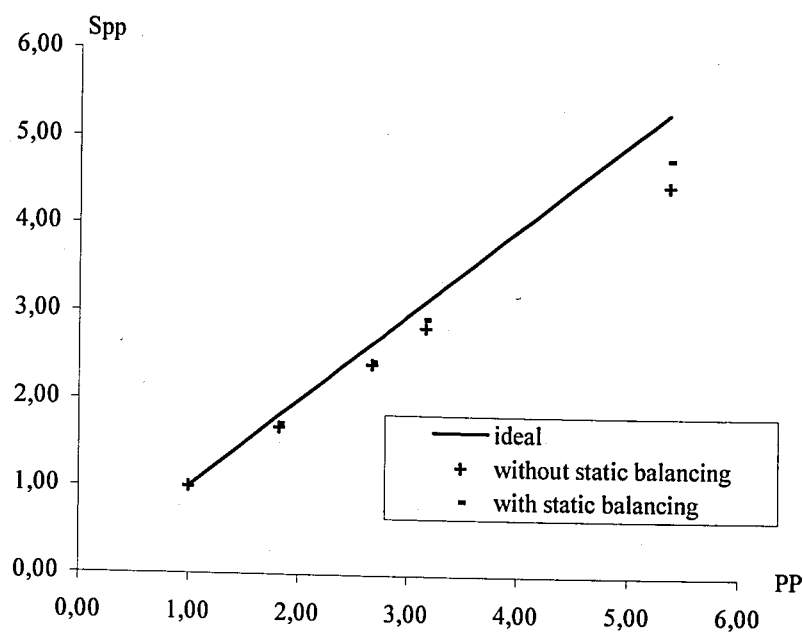


Figure 4.10. Speedup curve for the M/M/1 system with dynamic load balancing

Figure 4.10 presents the speedup curve where each point corresponds to one additional workstation. Similar to the results obtained in the previous section without dynamic load balancing, a sublinear speed up is also observed on four workstations with efficiency values over 90 per cent. However, an additional much faster workstation drops the efficiency to 84 per cent. The deviation from the ideal as the number of workstations is due to the fact that the re-mapping is based on the estimated speed values. Figure 4.11 shows the progress of the simulation with respect to real time for the experimentation on five workstations with no initial balancing. For the sake of clarity only the progress of the slowest (workstation-4, bottom line) and the fastest (workstation-8) workstations are shown. The duration of the simulation lasts 123 sec. The expected finish times for the slowest and fastest workstations without any load balancing are 238 seconds and 54 seconds respectively as marked with dashed lines. Note that the processing powers are benchmarked as 1.00 to 4.53. (From Table 4.1. 28.17 and 127.74).

Figure 4.12 depicts the magnification of the beginning (the first 10 sec.) of Figure 4.11 where most of the load migrations took place. After receiving some new variants, workstation-8 slowed down. The fast increase in the curve for workstation-8 following the

migration is due to the simulation of unprocessed events on migrated variants. On the contrary, workstation-4 accelerates after the migration. The discontinuity in any progress lines shows the time needed for the migration.

As mentioned before, the migration only entails sending the data segment. The necessary communication of the current data set has yielded on the average an additional delay of 20 msec per variant. Table 4.6 shows values for the total number of migrations attempted and the total number of variants migrated as additional workstations included to the simulation run. It is clear that these values get substantially higher if workloads are not initially balanced. Note that the workstation # 8 is the significantly faster than the others, which causes relatively more variants to be migrated.

Table 4.6. Number of migrations occurred

Number of Workstations (Workstation #)	No initial balancing		With initial balancing	
	Number of migrations	Number of variants	Number of migrations	Number of variants
2 (1,2)	8	63	4	18
3 (1,2,3)	10	70	5	22
4 (1,2,3,4)	9	124	4	12
5 (1,2,3,4,8)	16	224	3	18

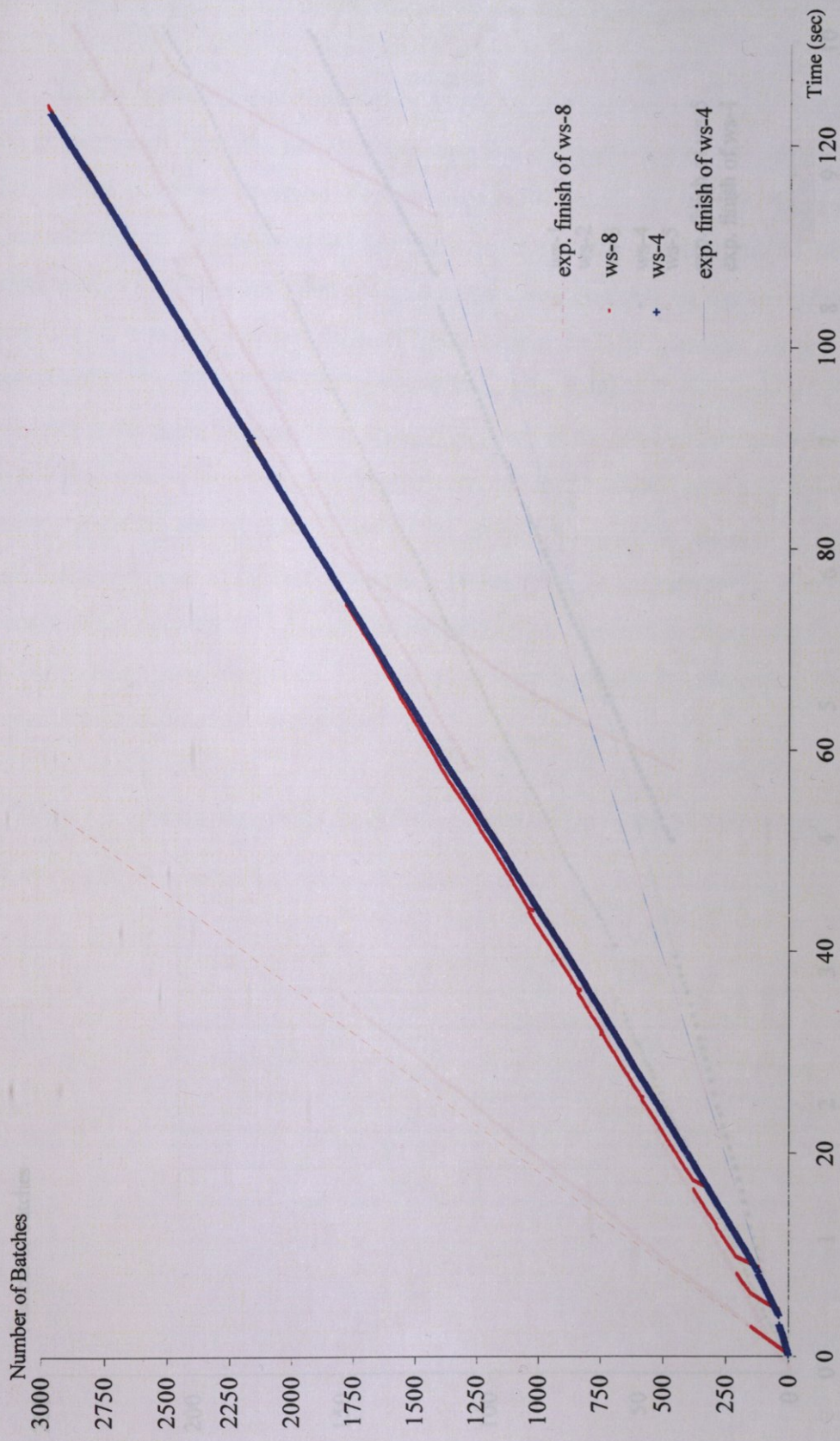


Figure 4.11. Progress of the simulation (workstation-4 and workstation-8)

4.7. Experiments by Varying Parameters

In this section we are interested in determining how parameters of the system effect the performance with the aim of obtaining the relation between simulation time and the number of events accepted. For this purpose, a set of runs is carried out such that the total number of events accepted has increased in each run. The number of events processed is kept constant since the acceptance rule depends on the arrival rate, each run consists of a set of variants with different arrival rates but similar service rates. More specifically, for the $M/M/1$ system shown in Table 4.7, the range of arrival rate of each set is designed in such a way that 2.5 percent of the total events generated is accepted in the first run, 12.5 per cent of them is accepted in the second run, and so on. In order to make the global rate of events constant for all runs, the service rate μ is 19 for all variants and the arrival rate of the last variant is 19 such that the global rate $\lambda = 20$. The results of the simulation runs show that the simulation time increases by increasing the number of event accepted for each specific run.

(Table 4.7). Parameter setting for different experiments of $M/M/1$ system with $\mu = 19$

Run	λ Range	Events accepted
1	[0.0, 0.49]	2.5%
2	[0.0, 2.49]	12.5%
3	[0.0, 4.98]	25%
4	[0.0, 7]	37.5%
5	[0.0, 9.98]	50%
6	[5.0, 7.49]	62.5%
7	[5.0, 9.98]	75.0%
8	[7.5, 9.99]	87.5%

Number of Batches

250

200

150

100

50

0

0

1

2

3

4

5

6

7

8

9

10

Time (sec)

- ws-1
- ws-2
- ws-3
- ws-4
- ws-5
- exp. finish of ws-5
- exp. finish of ws-4

Figure 4.12. First 10 seconds of the simulation progress

4.7. Experiments by Varying Parameters

In this section, we are interested in determining how parameters of the system effect the performance with the aim of examining the relation between the simulation time and the number of events accepted. For this purpose, a set of runs is carried out such that the total number of events accepted has increased for each run while the number of events processed is kept constant. Since the acceptance rate depends on the arrival rate, each run consists of a set of variants with different arrival rates but similar service rates. More specifically, for the set of M/M/1 system shown in Table 4.7, the range of arrival rate of each set is designed in such a way that 2.5 per cent of the total events generated is accepted in the first run, 12.5 per cent of them is accepted in the second run, and so on. In order to make the global rate of events constant for all runs, the service rate μ is 10 for all variants and the arrival rate of the last variant λ is 10 such that global rate $\Lambda=20$. Each set is run on network that consists of 3, 5 and 7 workstations. As depicted in Figure 4.13, the results of the simulation runs show that the simulation time increases by increasing the number of event accepted for each specific run.

Table 4.7. Parameter setting for different experiments of the M/M/1 system with $\mu=10$

Run	λ Range	% of Events Accepted
1	[0.0 , 0.49]	2.5%
2	[0.0 , 2.49]	12.5%
3	[0.0 , 4.98]	25.0%
4	[0.0 , 7.47]	37.5%
5	[0.0 , 9.96]	50.0%
6	[5.0 , 7.49]	62.5%
7	[5.0 , 9.98]	75.0%
8	[7.5 , 9.99]	87.5%

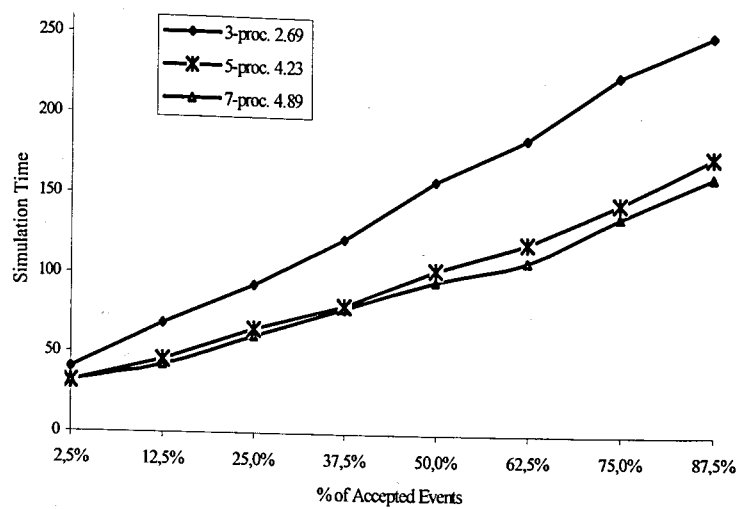


Figure 4.13. Speedup curves obtained for various acceptance rates

5. MODEL DEVELOPMENT MODULE OF THE USER INTERFACE

The complete system consists of a web based graphical user interface that communicates with a powerful server. The user interface facilitates the model construction and the analysis of simulation results in a platform independent way. The system described below enables the user to access the simulation environment over the internet, create simulation models of network of queues, update information for the variants, carry out the desired experiments and view the results. The simulation model is constructed by the model development component as explained in the next section.

5.1. General Overview of Model Development

The user is faced with a user interface window when the web page with the Front-end GUI applet, is downloaded by entering its URL. The Model Development is carried out on the window that is the main control window as seen in Figure 5.1. It consists of three parts. The central panel is the *model design area*. The design area is where the user provides a description of the model in terms of predefined constructs such as service and queue. The left side is the *construct panel*. The upper part of this panel comprises of radio buttons that enable the user to select the construct type. The **Info** button opens up a new template window to enter the construct's parameters. Pressing the **Clear** button from this panel can wipe out current model while the **Delete** button removes a single construct, e.g. the active one. Finally, the bottom part is the *model action* panel that consists of buttons for the operations related with the simulation model, these are **run**, **load**, and **save** actions. In addition, it is possible to view a certain variant of the model (a particular simulation experiment with its parameters) by typing the experiment number into the field next to the **Experiment** button in the bottom middle area.

The user builds up the model by selecting the appropriate radio button for the desired construct and adds them into model by clicking the mouse on any position in the design area. Every time the mouse is clicked, an action will be performed at its location in the design area; i.e. a new construct will be drawn at the location. The currently inserted construct remains active until a new construct is added. To distinguish from the others, the

active construct is colored in red. An existing construct may become active by clicking on its shape. Before inserting a connector, the user must select two constructs that can logically be connected. Then, a connector link will be drawn between these two constructs; otherwise, no action will be taken. An active construct can be edited, moved around the design without losing their connections, and deleted with its incoming and outgoing connections.

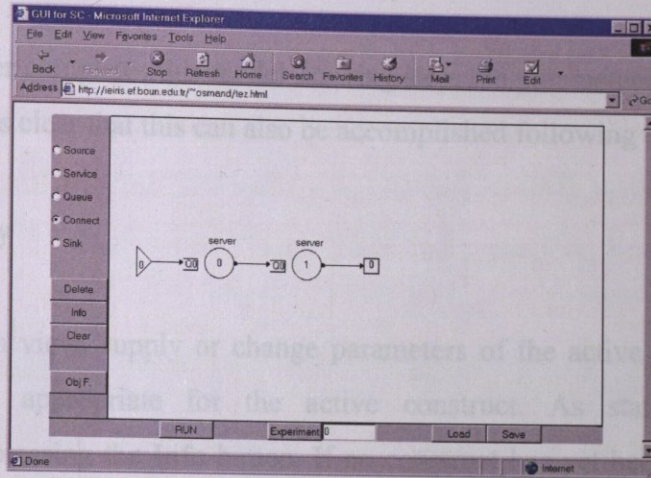


Figure 5.1. Model development window

5.2. Illustration of a Typical Session

For the sake of illustration, we choose an example of a two-node Jackson network. Figure 5.1 shows this interface with this network whose structure consists of one source two service stations that are connected in tandem and a sink. The customers arrive at a rate $\lambda=0.5$ and upon the completion of service at station 1, the customers continue to station 2. The objective in this problem setting is to identify the design that minimizes the mean sojourn time;

$$\begin{aligned}
 & \min_{\mu_1, \mu_2} E[ST] \\
 & \text{st. } \mu_1 + \mu_2 = 4 \\
 & \mu_1 > 0.5, \mu_2 > 0.5
 \end{aligned} \tag{5.1}$$

We define the experimental region as the following: the arrival rate is kept constant at $\lambda=0.5$, the service times are distributed with rate $\mu_{1j} = 0.525+0.025j$ and $\mu_{2j} = 3.475-0.025j$ for the j^{th} alternative design, simply referred to as experiment $j = 0,1,...,118$.

This two-node Jackson network model is composed of six constructs. It can be created by inserting one source, two services, two queues and one sink in our user interface as described in the section above. They are then linked with the connectors accordingly.

Once the user completes this model drawing step, the parameters must be entered for each construct. It is clear that this can also be accomplished following each insertion.

5.2.1. Data Entry

The user can view, supply or change parameters of the active construct through a dialogue window appropriate for the active construct. As stated before, this is accomplished by pressing the **Info** button. If no construct has yet been made active, then the **Info** button is disabled.

Clicking the "Info" button opens the "Source Information" dialogue window for the user to enter the rate of arrival for the active source construct. Figure 5.2 shows this window along with the arrival rate value entered as 0.5. Since one type of entity is used in the example, the fields for one class appear in the window, denoted as C1.

More information concerning the service construct should be supplied. The "label" field is used to describe the name of the service station with its default as "server". The second parameter is the service type that can be selected from the pull-down list. Finally, typing their values into the corresponding fields sets the number of servers and the service rate for each class. As depicted in Figure 5.3, the first service node is labeled as "station 1" with its single server having a service rate of 0.525. This rate is used when no update info is defined for the experiment.

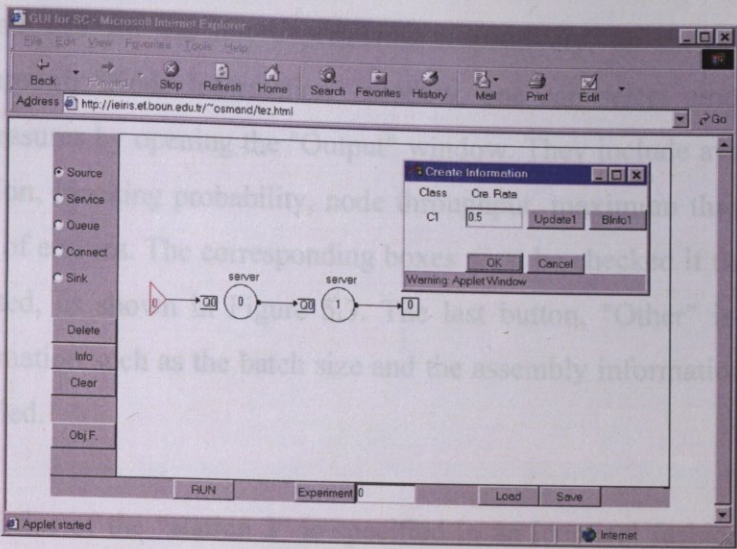


Figure 5.2. Screenshot of the GUI with the model created while entering the arrival rate value $\lambda=0.5$ in the "Data Input" window

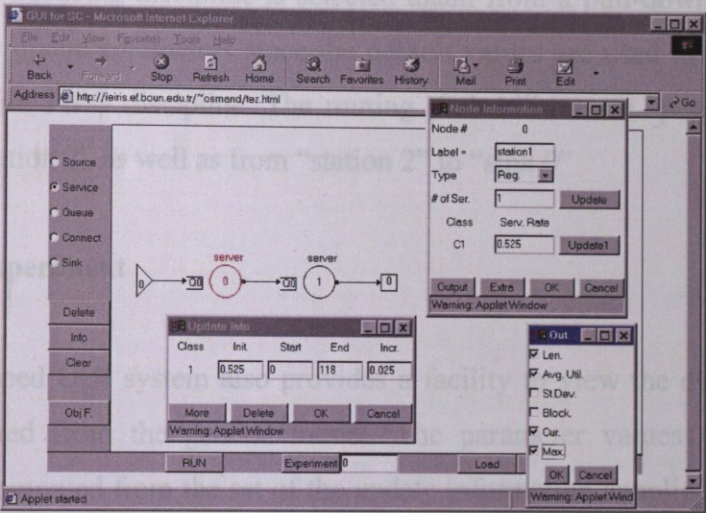


Figure 5.3. Screenshot of the GUI while entering parameters for the service node 0 through "Node Information", "Update Info" and "Output" windows

Model developed in the design area can be referred to as the general or the basic model. As with the Source construct, "Update Info" window is brought up by clicking on the Update1 button next to the "Rate" field. This window is used to define the deviation from the general model. The information is supplied in the form of increment/decrement value with the starting and ending experiment number pair. In our experimentation region for the example problem, the initial service rate of the server in "station 1" is 1.0. It is incremented by 0.025 in the 119 successive experiments. This information is entered as 0

for the start and 118 for the last experiments respectively, as shown in Figure 5.3. Furthermore, through "Node Information" window, one can select among the possible performance measures by opening the "Output" window. They include average utilization, standard deviation, blocking probability, node throughput, maximum throughput, and the current number of entities. The corresponding boxes must be checked if the user wants the measures reported, as shown in Figure 5.3. The last button, "Other" is used to supply additional information such as the batch size and the assembly information if such type of service is specified.

Characteristics of the "station 2" is specified in an identical fashion as for the first one. The initial value for the rate is 3.475 and the increment parameter is -0.025 to denote a decrement. As to the queues associated with the service nodes, queue length can be specified for the basic model and varied for the experiments in a similar way done for the service rate. The queueing discipline is selected again from a pull-down list. No entry is necessary for the queue parameters since the default values are used in this example, i.e. infinite queue with FIFO discipline. The routing probabilities are specified as 1.0 from "station 1" to "station 2" as well as from "station 2" to "sink 0"

5.2.2. Single Experiment

The developed GUI system also provides a facility to view the description of each experiment derived from the general model. The parameter values of this particular experiment are computed from the set of the update information supplied. Each individual experiment with its own parameter values can be examined in a separate window that is brought up by means of the **Experiment** button. The user has to enter the experiment number in the field at the mid-area of the action panel. Figure 5.4 shows the experiment #60 of our current design, where each construct is associated with its parameters, e.g. station 1 has a single server with $\mu_1=2.0$ displayed as 1/2.0 below the node shape, routing probabilities are not displayed if equal to 1.

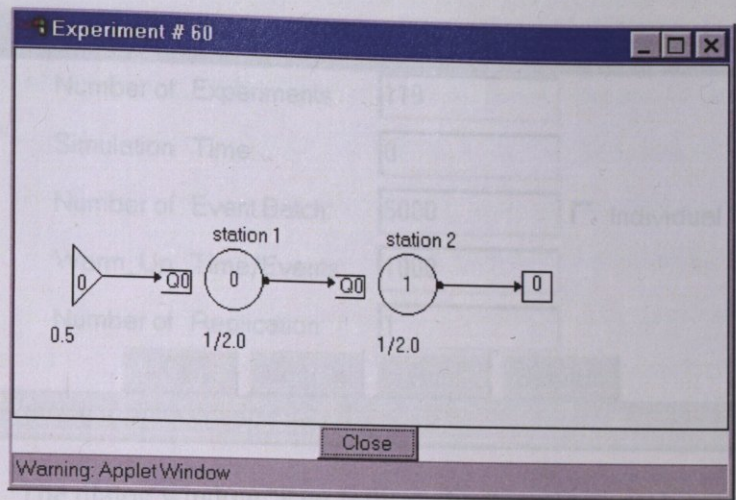


Figure 5.4. Experiment #60 with its parameters

5.2.3. Simulation Execution

After the model has been developed, the user may click on the **Run** button in the action panel to bring up the "Run Information" dialogue window. The run parameters of the simulation study such as the number of experiments, and stopping criteria are defined through this window.

As shown in Figure 5.5, the first line of the window is used for defining the number of model variants. Its default value is displayed as computed from the maximum number entered for the last experiment value in the update information declaration (=119 specified in the update window for the service rate as 0 through 118). The user can choose among three different ways to terminate the simulation by filling out the corresponding field, i.e. the *Simulation Time* field or the *Size of Event Batch* field. In the displayed case, the size of event batch field is specified as 5000 with an additional 1000 event batch for the warm up period, each batch consists of 40 events. As a third type of stopping criteria, the simulation may terminate when certain number of entities from a given class leaves the system. In this case, then the *Individual* checkbox has to be selected and the relevant information needs to be entered through the "Sink Info" window.

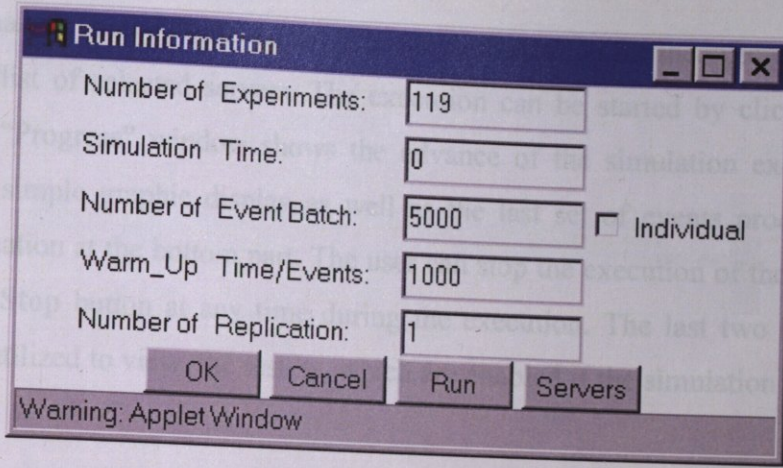


Figure 5.5. The dialog window "Run Information" used to supply run parameters

Furthermore, the user can select the available servers that will take part in the simulation execution. The simulation workload may be distributed on a network of up to seven computers designated as server 1 through server 7. As stated before, the simulation manager determines the servers that are available for the simulation execution. The list of available servers is displayed in the left window shown in Figure 5.6, in which the user has selected the first three servers for the execution.

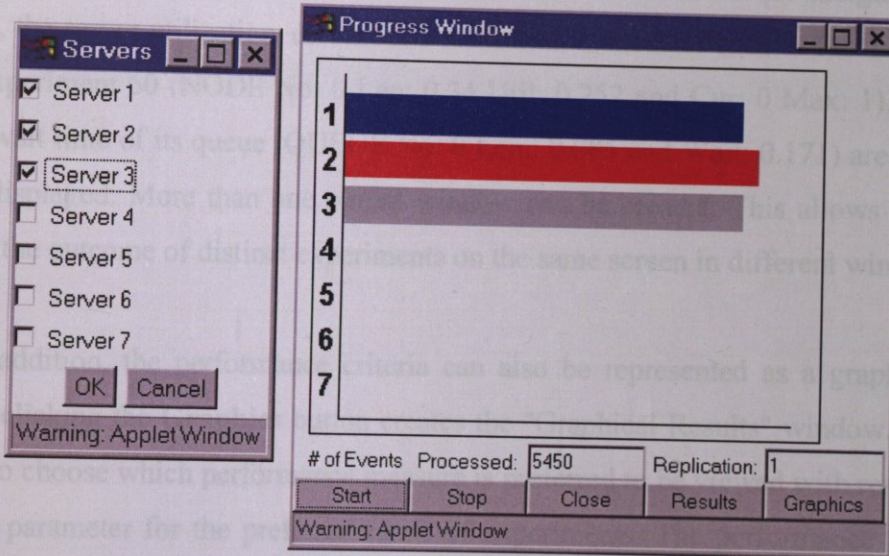


Figure 5.6. Windows used to select computers and view the progress of the simulation run

The **Run** button in Figure 5.5 becomes enabled upon completing the run parameter specification. Clicking on this button opens the "Progress" window displayed in the right part of Figure 5.6. Meanwhile, the corresponding Applet communicates with the

simulation manager and submits all the necessary information for the execution such as the model data, list of selected servers. The execution can be started by clicking the **Start** button. The "Progress" window shows the advance of the simulation execution on the servers in a simple graphic display as well as the last set of events processed and the current replication at the bottom part. The user can stop the execution of the simulation by pressing the **Stop** button at any time during the execution. The last two buttons in this window are utilized to view the results, which are enabled if the simulation is successfully completed.

5.2.4. Presentation of Simulation Results

There are two ways of viewing the results: text-based and graphical. The text-based results consist of statistical summary report of each experiment. Upon clicking the **Results** button, they are displayed in "Results of the Experiments" window. Initially, the window is opened with an empty list of results. The user has to specify the range of experiments desired to view by typing the first and the last experiments, e.g. from experiment 55 to 65, as observed in Figure 5.7. The information contained in this window is a list of the performance criteria associated with each construct of the model in the design phase. For example, the server utilization, current and maximum number of entities for the first node of the experiment 60 (NODE No: 0 Len: 0.34 Util: 0.252 and Cur: 0 Max: 1), the length and the wait time of its queue (QUEUE No: 0 Len.: 0.085 and Wait: 0.171) are among the criteria displayed. More than one output window can be created. This allows the user to examine the outcome of distinct experiments on the same screen in different windows.

In addition, the performance criteria can also be represented as a graph. For that purpose, clicking the **Graphics** button creates the "Graphical Results" window. It permits the user to choose which performance measure is preferred to be viewed with respect to the specified parameter for the preferred range of experiments. The performance measure is selected from the list boxes provided at the bottom part of the window. The possible variables for the x-axis are the parameters that can be varied in the simulations such as service and arrival rates, the number of servers and the queue size. Similarly, any performance criteria such as throughput, utilization, and length can be selected for the y-axis. As it can be seen in Figure 5.8, we provide a snapshot of the window in which the

mean sojourn time values designated as Sys T are plotted versus the service rate values of the “station 1” (Node 0) for the experiments 51 through 69.

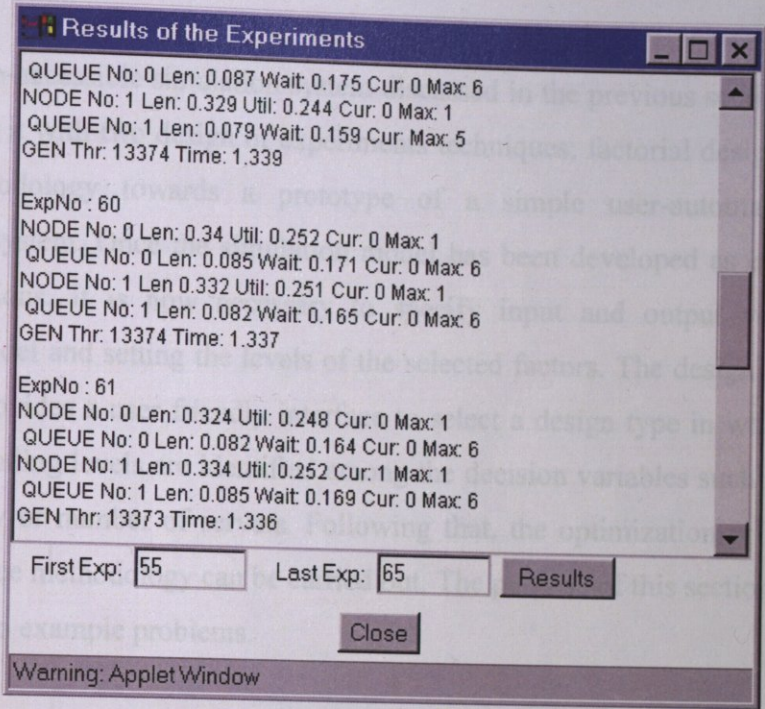


Figure 5.7. Output window depicting the result for experiments 50 through 60 in text form

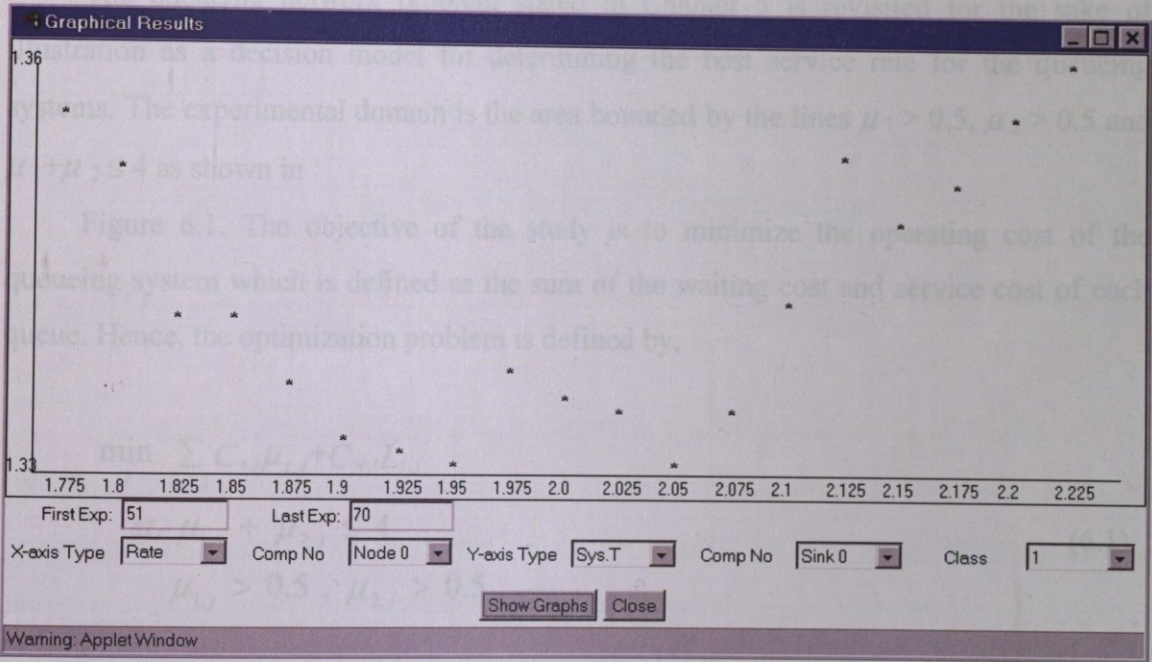


Figure 5.8. Screenshot of graphical result for the example problem

6. APPLICATION OF DESIGN OF EXPERIMENTS FOR SIMULATION OPTIMIZATION

The web-accessible simulation system discussed in the previous sections is extended by integrating it with two design of experiments techniques; factorial design and response surface methodology towards a prototype of a simple user-automated simulation optimization system. Once the simulation model has been developed as explained in the previous sections, it is now necessary to specify input and output variables of the simulation model and setting the levels of the selected factors. The design of experiments component provides a user friendly interface to select a design type in which factors and their corresponding levels are identified among the decision variables such as service rate, queue capacity or number of servers. Following that, the optimization process using the response surface methodology can be carried out. The purpose of this section is to illustrate its use with two example problems.

6.1. Two - Node Jackson Network

The queueing network problem stated in Chapter 5 is revisited for the sake of illustration as a decision model for determining the best service rate for the queueing systems. The experimental domain is the area bounded by the lines $\mu_1 > 0.5$, $\mu_2 > 0.5$ and $\mu_1 + \mu_2 \leq 4$ as shown in

Figure 6.1. The objective of the study is to minimize the operating cost of the queueing system which is defined as the sum of the waiting cost and service cost of each queue. Hence, the optimization problem is defined by,

$$\begin{aligned}
 &\min_j \sum_{i=1,2} C_{s,i} \mu_{i,j} + C_{w,i} L_{i,j} \\
 &\text{st. } \mu_{1,j} + \mu_{2,j} \leq 4 \\
 &\quad \mu_{1,j} > 0.5, \mu_{2,j} > 0.5
 \end{aligned} \tag{6.1}$$

where $C_{s,i}$ is the marginal cost per unit of service per unit time and $C_{w,i}$ is the cost of waiting per unit time per entity in station i .

We define the experimental layout as a grid composed of alternative design cells j with corner points corresponding to service rates $(\mu_{1,k}, \mu_{2,l})$ where $\mu_{1,k} = 0.55 + 0.05k$ for $k=0, \dots, 58$ and $\mu_{2,l} = 0.55 + 0.05l$ for $l=0, \dots, 58$. The arrival rate is kept constant at $\lambda=0.5$.

Full exhaustive search of this grid would require roughly 1770 simulation runs. Even though the SC approach is an efficient method, the distributed simulation of all variants may be time consuming and computationally expensive. We aim to reduce the computational effort of this investigation by applying the design of experiment techniques that provides a way of planning which variants to simulate. In the following section, we discuss the 2^k factorial design that starts with an arbitrary region for exploration with the aim of fitting a first order model and move rapidly to the vicinity of the optimum.

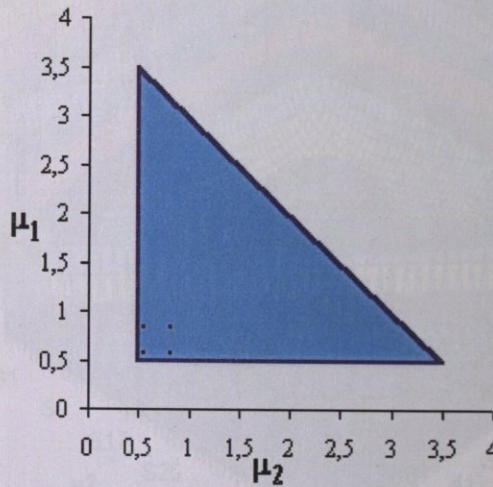


Figure 6.1. Design space used for the two-node Jackson Network

6.1.1. 2^k Factorial Design

The experimental design for this problem is used to check the significance of the effects of service rates, namely μ_1 and μ_2 , on the response variable which is the total operating cost. Since we attempt to fit a first-order model, we select a 2^2 factorial design. Figure 6.3 depicts the set of windows used to enter the design parameters. Factor A and B are defined as the service rate for each of the servers (Rate of Node 0 and Rate of Node 1 in the "Global Panel" window). The region of interest for this design is arbitrarily selected. It is defined by the levels chosen as $[0.55, 0.85]$ for factor A as well as factor B (as

specified the "Factor Panel" window). This design leads to the completion of four experiments, five replications will be conducted for each combination; therefore 20 simulation runs are made. As to the cost function f in (6.1), it is specified by selecting the corresponding parameter and performance measure and its associated component from the list in the "Objective Function" window (*Rate* and *Len* items of Node 0 and Node 1). For this problem, each cost parameter is specified as 1. The response surface for the example problem is depicted in Figure 6.2.

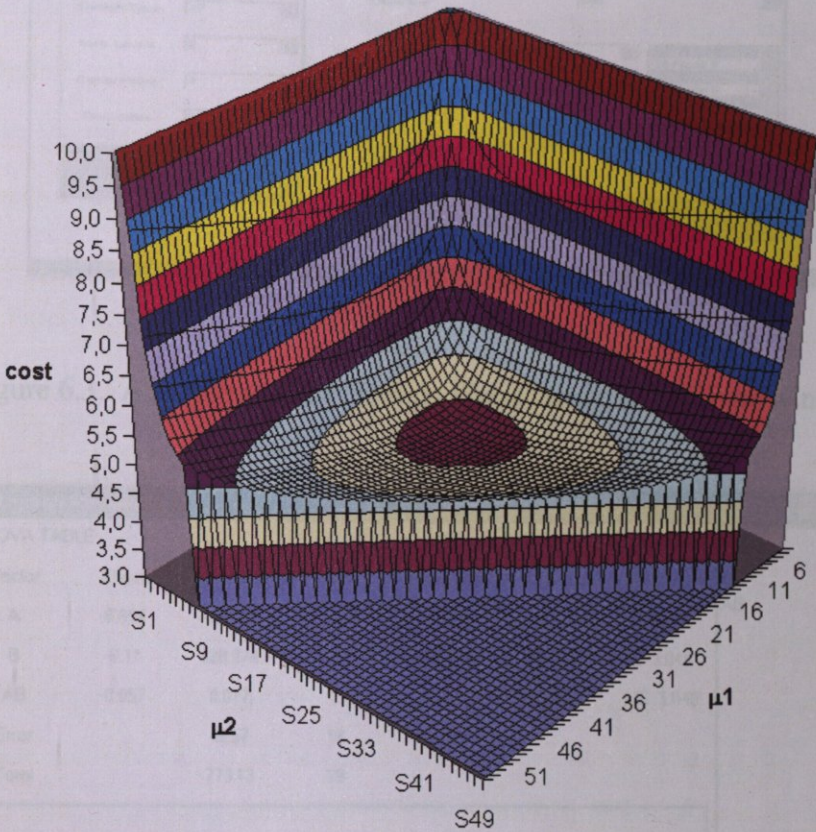


Figure 6.2. Response surface of the two-node Jackson Network for $C_s=(1,1)$, $C_w=(1,1)$

Execution of the experiments according to the experimental design generates the corresponding output data which are then used to compute the objective function value of the experiment. The statistical analysis of the simulated data is carried out through "Design Results" window which provides the effects of the two independent factors on the dependent variable. Figure 6.4 illustrates the ANOVA table that corresponds to the generated data. We see that the main factors *A*, *B* are significant (indicated by symbol 'S' in the last column, $130.3 > F_{0.1, 1, 16} = 3.04$, F-value is computed by approximation after

10) for the cost and the interaction **AB** is non-significant at 0.1 level of significance (Conf: %10).

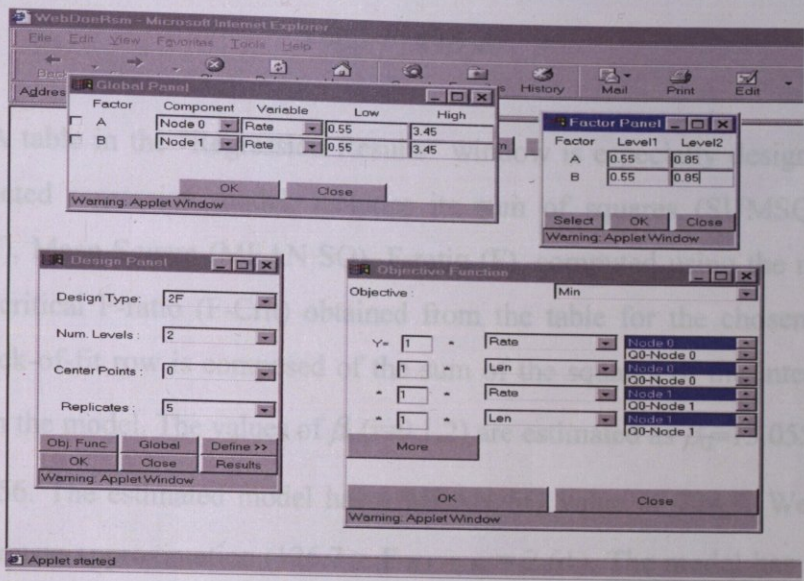


Figure 6.3. A snapshot of the "Design Phase" along with all the windows

The screenshot shows the 'Design Results' window with an ANOVA table for a 2² factorial design. The table is as follows:

Factor	Effect	SUMSQ	DF	Mean SQ	F	F-Crit	A B AB
A	-8.894	395.57	1	395.57	130.31	S 3.048	
B	-8.11	328.974	1	328.974	108.372	S 3.048	
AB	-0.057	0.017	1	0.017	0.0060	NS 3.048	
Error		48.57	16	3.036			
Total		773.13	19				

At the bottom of the window, there are buttons for 'Close' and 'Update', and a 'Conf:' dropdown menu set to '%10'.

Figure 6.4. The ANOVA Table of 2² factorial design

The obtained model includes two main factors **A**, and **B**. In the next subsection, we explore the regression analysis approach to estimate the relation between the **cost** and the factors. This is carried out by the "Model Fit" component of our system that consists of simple statistics software that uses regression analysis. Based on the design described

above, we present a snapshot of model fitting process in Figure 6.5. Through the "Model Fit Panel" we choose the first-order model given by

$$Y = \beta_0 + \beta_1 A + \beta_2 B. \quad (6.2)$$

The ANOVA table in the "Regression Results" window is especially designed for RSM. For the selected regression model, includes its sum of squares (SUMSQ), degree of freedom (DF), Mean Square (MEAN SQ), F-ratio (F), computed using the residual mean square, and critical F-ratio (F-Crit) obtained from the table for the chosen significance level. The lack-of-fit row is composed of the sum of the squares of the interaction effect dropped from the model. The values of β_i ($i=0,1,2$) are estimated as $\beta_0=13.055$, $\beta_1=-4.447$, and $\beta_2=-4.056$. The estimated model has a MEAN SQ value of 724.5. We see that the model is adequate approximation ($126.7 > F_{0.1, 2, 16} = 2.61$). The model has an R-squared value of 93.7 per cent that accounts for its goodness of fit.

From the estimated parameters of the model, "Optimizer" component helps the user to perform a line search from the center point of the current region of interest. Because this problem is defined as a minimization problem, the direction of steepest descent is (4.447, 4.056). The step size is supplied as 0.05. Therefore, the steps along the path are $(\Delta A, \Delta B) = (0.05, 0.45)$. As seen in Figure 6.6, first fifteen experiments are proposed for the steepest descent path as the user chooses "Experiments". To end the line search, the user can determine the number of consecutive observation that is higher than the preceding observations. Since the stopping number of experiments is selected as 0, all the experiment are executed. Note that by the virtue of the SC approach all experiment are simulated in parallel. The new region of interest is centered at the point (1.25, 1.201) which corresponds to run #10. Therefore, another 2^2 factorial design should be performed and a second first order model should be fit in the vicinity of this point and pick a new direction. This process continue until we reach a point where the metamodel appears flat.

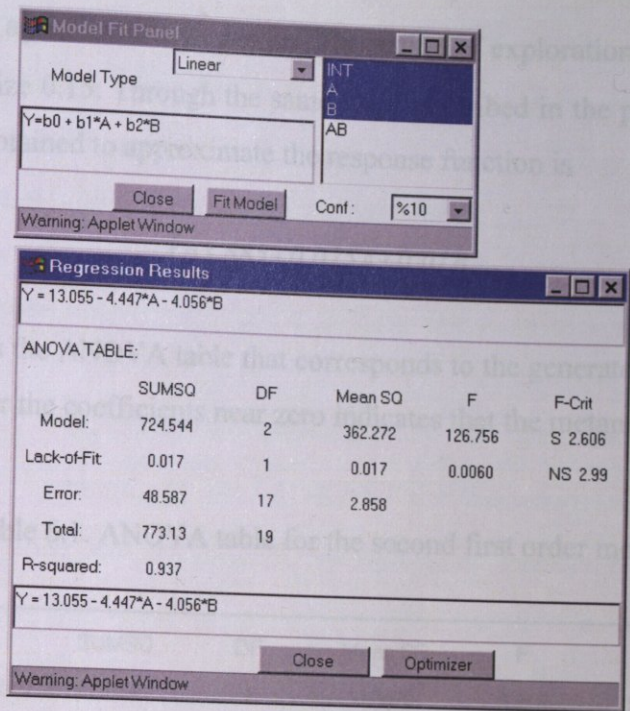


Figure 6.5. A snapshot of the "Model Fitting Phase" along with the "Regression Results"

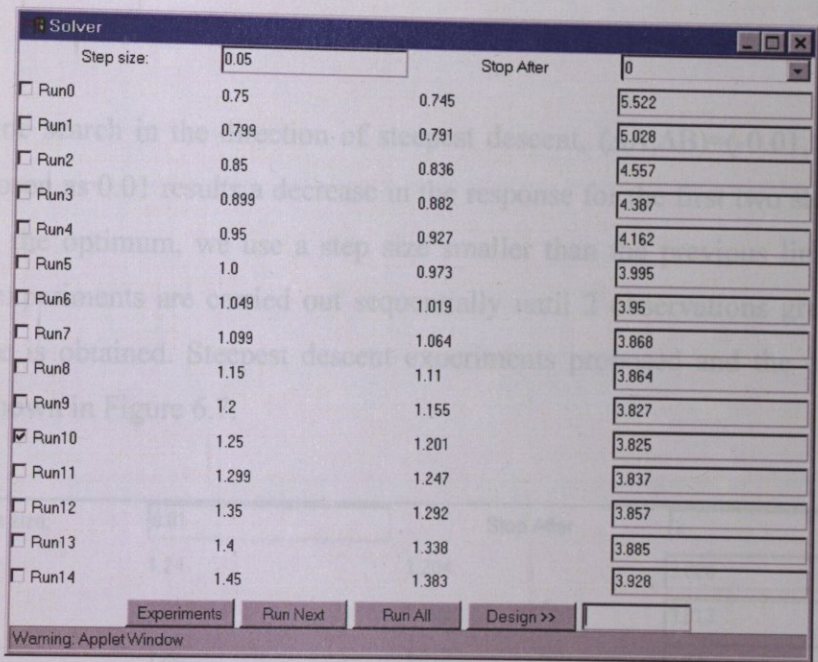


Figure 6.6. Proposed experiments on the steepest descent direction for the Jackson Network

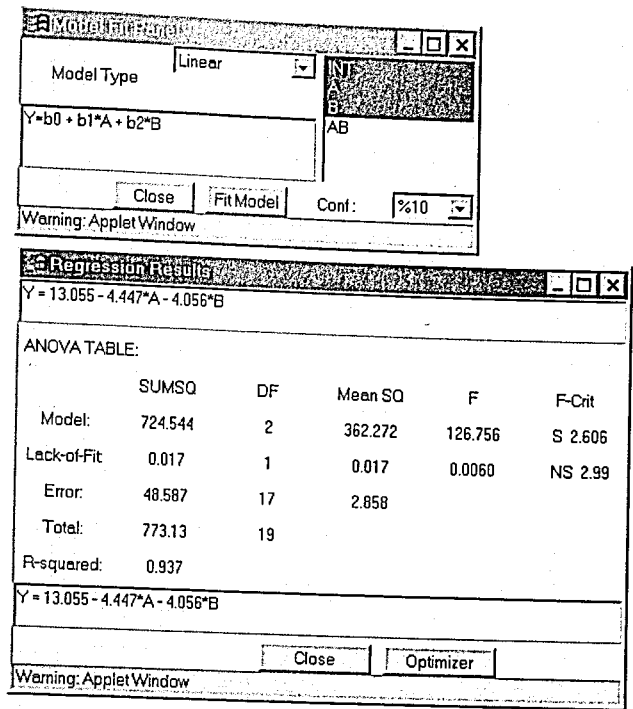


Figure 6.5. A snapshot of the "Model Fitting Phase" along with the "Regression Results"

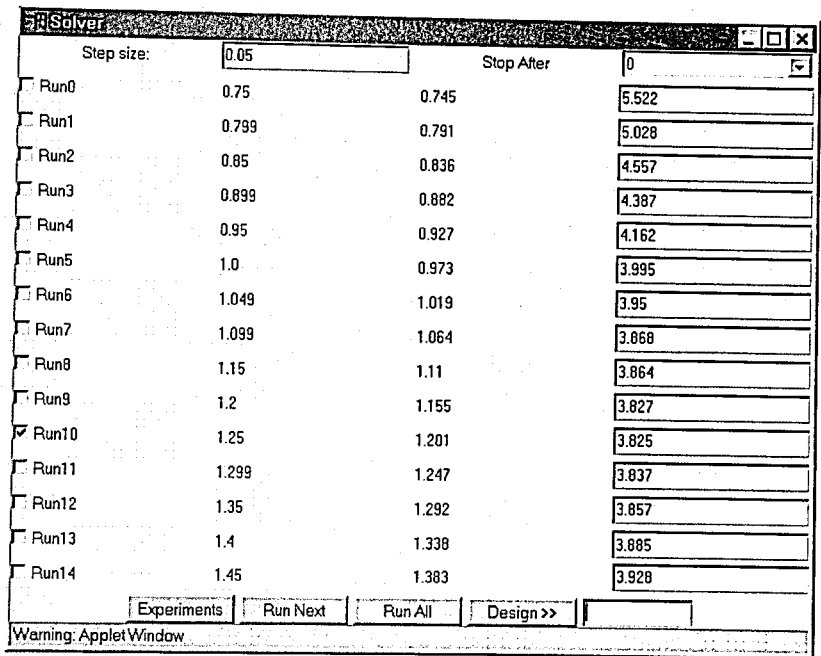


Figure 6.6. Proposed experiments on the steepest descent direction for the Jackson Network

We continue again another 2^2 design. The region of exploration is centered at (1.25, 1,20) with a step size 0.15. Through the same steps described in the previous sections, the first order model obtained to approximate the response function is

$$\hat{Y}=3.885+0.025A+0.01B. \tag{6.3}$$

Table 6.1 illustrates the ANOVA table that corresponds to the generated data. The model is significant, however the coefficients near zero indicates that the metamodel becomes flat.

Table 6.1. ANOVA table for the second first order model

	SUMSQ	DF	Mean SQ	F	F-Crit
Model:	0.014	2	0.0070	8.915	S 2.606
Lack-of-Fit	0.0	1	0.0	0.152	NS 2.99
Error:	0.014	17	0.0010		
Total:	0.028	19			
R-squared:	0.512				

The line search in the direction of steepest descent, $(\Delta A,\Delta B)=(-0.01, 0.0042)$ with step size chosen as 0.01 results a decrease in the response for the first two steps. Since we are close to the optimum, we use a step size smaller than the previous line search, and simulation experiments are carried out sequentially until 2 observations greater than the previous one is obtained. Steepest descent experiments proposed and the corresponding results are shown in Figure 6.7.

Step size:	0.01	Stop After	2
Run0	1.24	1.204	3.826
Run1	1.23	1.208	3.813
Run2	1.22	1.212	3.83
Run3	1.21	1.216	3.833
Run4	1.2	1.22	

Figure 6.7. Steepest descent experiments proposed and results obtained for second first order model

The new center of investigation is the point [1.23, 1.21]. We are close to the optimum point, another 2^2 factorial design seems unnecessary, we suggest a detailed enumeration of all possible system configurations around this new point should be used to find the optimal point which is computed analytically as [1.207, 1.207]. Using the RSM component, we reach the neighbor of the optimal value after performing 116 experiments. Additional experiments would be required to find the optimal response value.

A new set of experiments are performed by changing the cost parameters as $C_s=(1,2)$, $C_w=(3,5)$. The same set of analysis has been carried out, Figure 6.8 shows the steepest descent path for two first order fits as well as its associated experimental design region.

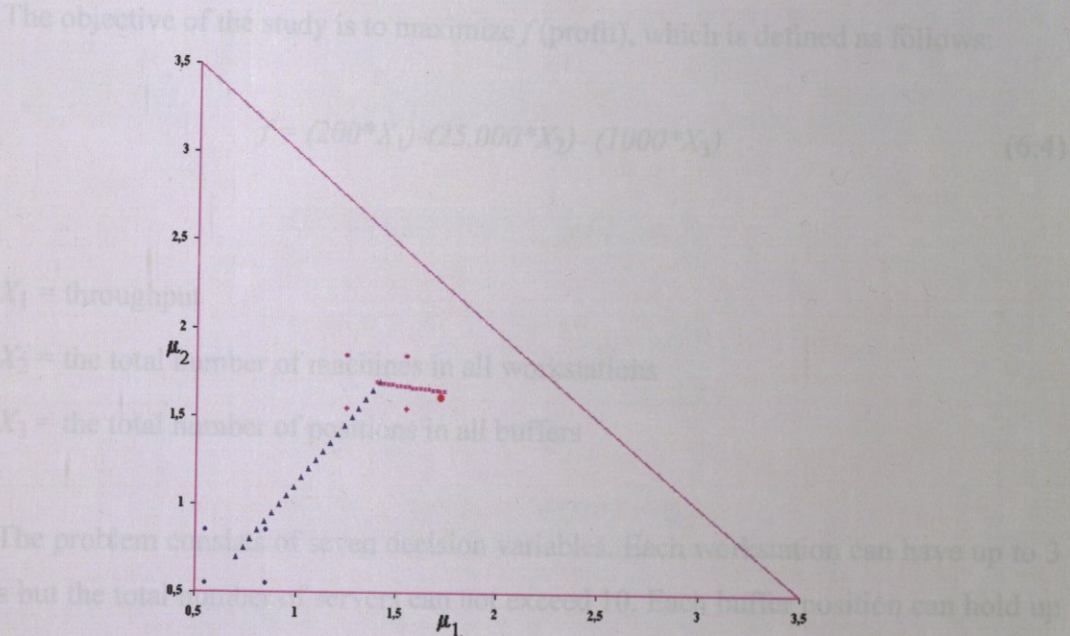


Figure 6.8. Steepest descent paths of two first order models for $C_s=(1,2)$, $C_w=(3,5)$

6.2. Manufacturing System Example

As a second example we consider the manufacturing system which is used as a sample problem in the simulation optimization area (Law and Mc Comas, 2000). The system consists of four workstations and three buffers (queues) as shown in Figure 6.9.

Processing times have exponential distribution with means 0.333, 0.5, 0.2 and 0.25 respectively and the server cannot discharge a part if the succeeding buffer is full.

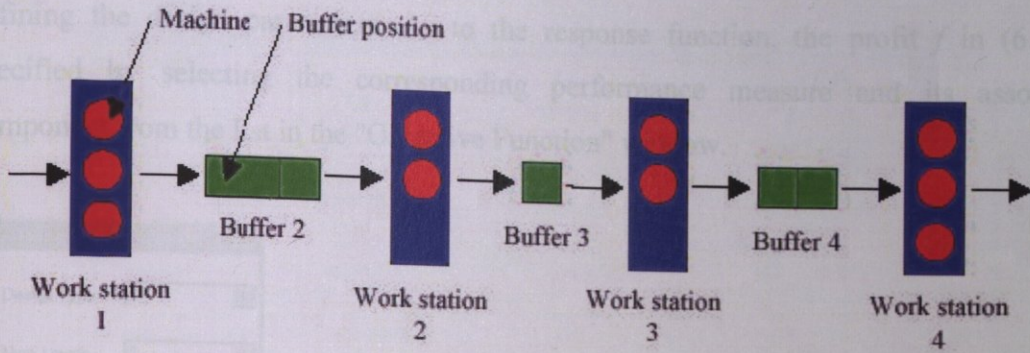


Figure 6.9. Manufacturing system with four workstations

The objective of the study is to maximize f (profit), which is defined as follows:

$$f = (200 * X_1) - (25,000 * X_2) - (1000 * X_3) \quad (6.4)$$

where

X_1 = throughput

X_2 = the total number of machines in all workstations

X_3 = the total number of positions in all buffers

The problem consists of seven decision variables. Each workstation can have up to 3 servers but the total number of servers can not exceed 10. Each buffer position can hold up to 10 parts.

From the load balancing consideration, one can heuristically conclude that 3, 3, 2, and 2 servers at stations 1, 2, 3, and 4, respectively, are optimal. The objective of the problem in this study is reduced to determining the optimal number of buffer positions at each station. We attempt to approximate the objective function by a first-order model. Therefore, we consider three factors at two level each, i.e. a 2^3 factorial experimental design. Factors A, B, and C correspond to the queue size associated with node 1, node 2 and node 3, respectively. The design domain is given the set integer values 1 through 10 for all factors as stated in the original paper. Full exhaustive search of this domain would

require 1000 simulation runs. As for stating the region of interest, we selected for all factors the set of integer values 1 through 3 which is centered at 2 and five replications was made for each system configuration. Figure 6.10 depicts the set of windows used in defining the design parameters. As to the response function, the profit f in (6.4), is specified by selecting the corresponding performance measure and its associated component from the list in the "Objective Function" window.

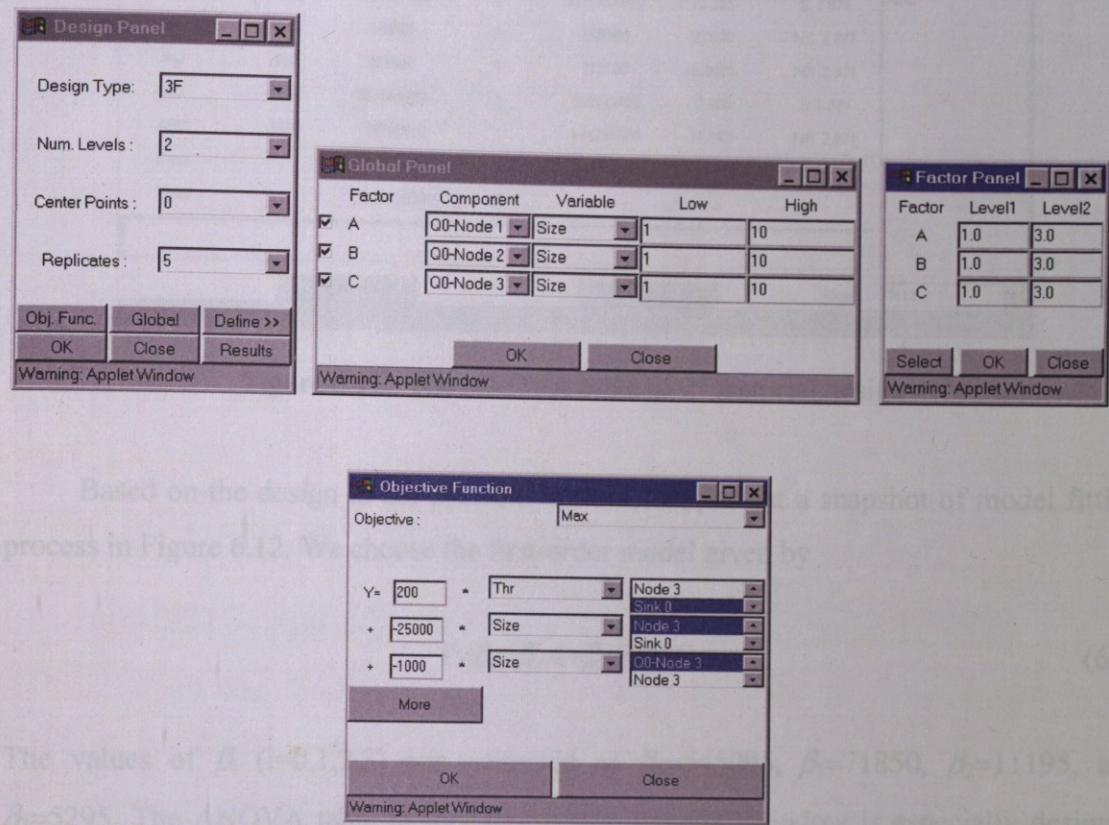


Figure 6.10. The set of windows used to define the design parameters of the manufacturing system example

After the model and the design parameters have been specified, the set of responses are computed by the simulation engine. From the ANOVA table given Figure 6.11, main factors A, B, and C are significant for the dependent variable at 10 per cent level of significance (for A, $22.657 > F_{0.1, 1, 32} = 2.87$). However, table shows that interaction BC is significant while the other interaction effects are not. They can be neglected and added to the error. The significance is based on a user dependent confidence level. The ANOVA table is flexible in the sense that user can modify the confidence level or select the main factors or their interactions from the list that appears on the top-right part of the window. If

the $\alpha=0.5$ level of significance has been selected, ($F_{0.05, 1, 32} = 4.17$) BC would be insignificant.

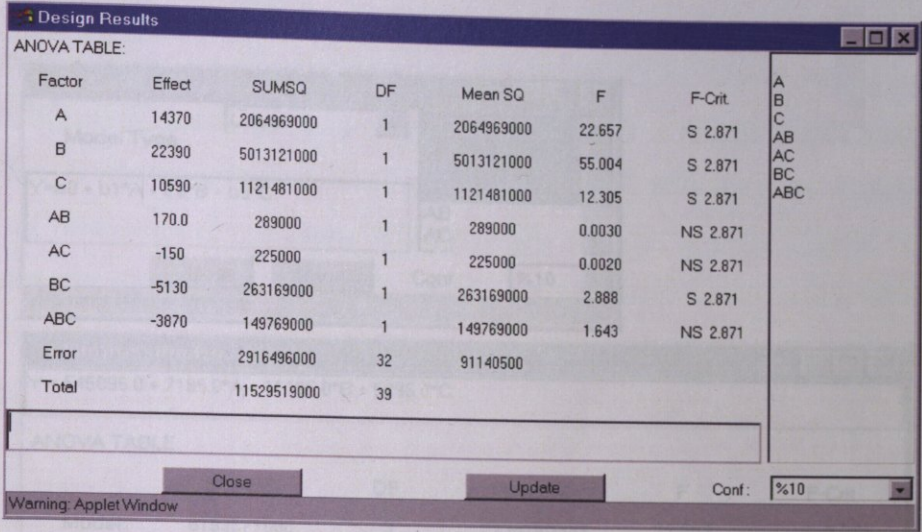


Figure 6.11. The ANOVA table of 2^3 factorial design

Based on the design in the previous section, we present a snapshot of model fitting process in Figure 6.12. We choose the first-order model given by

$$Y=\beta_0+\beta_1A+\beta_2B+\beta_3C. \tag{6.5}$$

The values of β_i ($i=0,1,2,3$) are estimated as $\beta_0=545095$, $\beta_1=71850$, $\beta_2=11195$, and $\beta_3=5295$. The ANOVA table in the "Regression Results" window is especially designed for RSM. The interaction effect results an insignificant lack-of-fit for estimated model ($0.9 > F_{0.1, 4, 32}= 2.1$, indicated by the symbol NS). The best model fit that describes the behavior of the objective function is given below the table.

As seen in Figure 6.13, seven experiments are proposed for the steepest ascent path. Since the stopping number of experiments is selected as 2, execution is stopped after run number 6 and the new region of interest is centered at the point (5,7,4) and with the step size 2. It is described by the set of integer values 3 through 7, 5 through 9 and 2 through 6 for factors A, B, and C respectively.

In the second first order fit, model parameters are estimated as $\beta_1=11000$, $\beta_2=1700$, and $\beta_3=1140$, which leads to a maximum estimated profit of 590,980 at configuration 7,7,4.

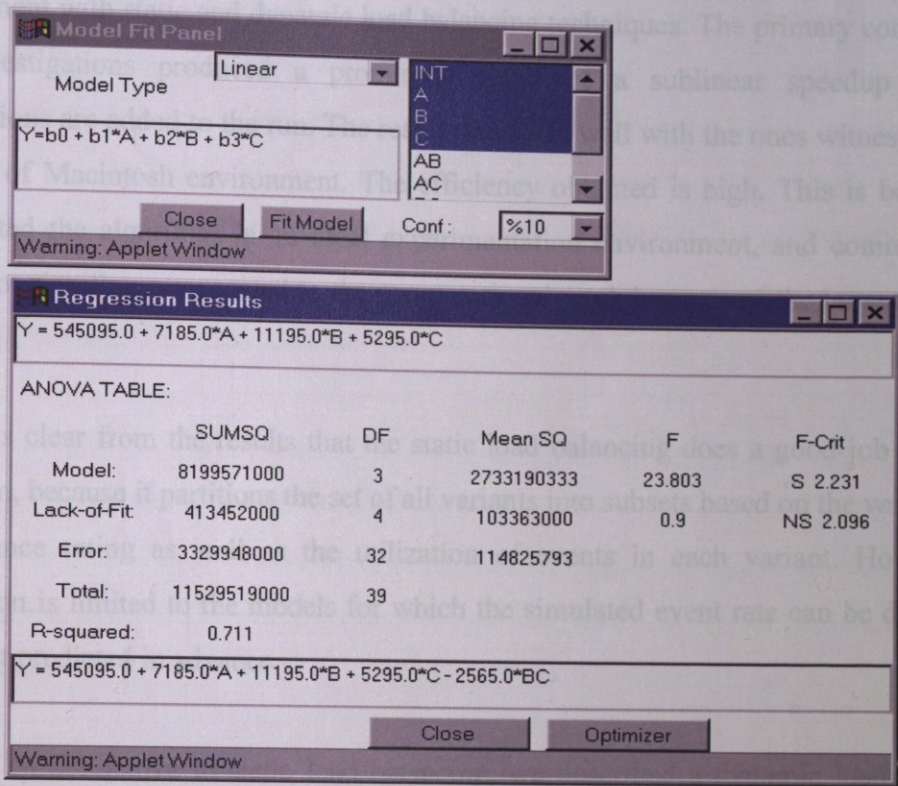


Figure 6.12. Model selection and ANOVA table of the first order model

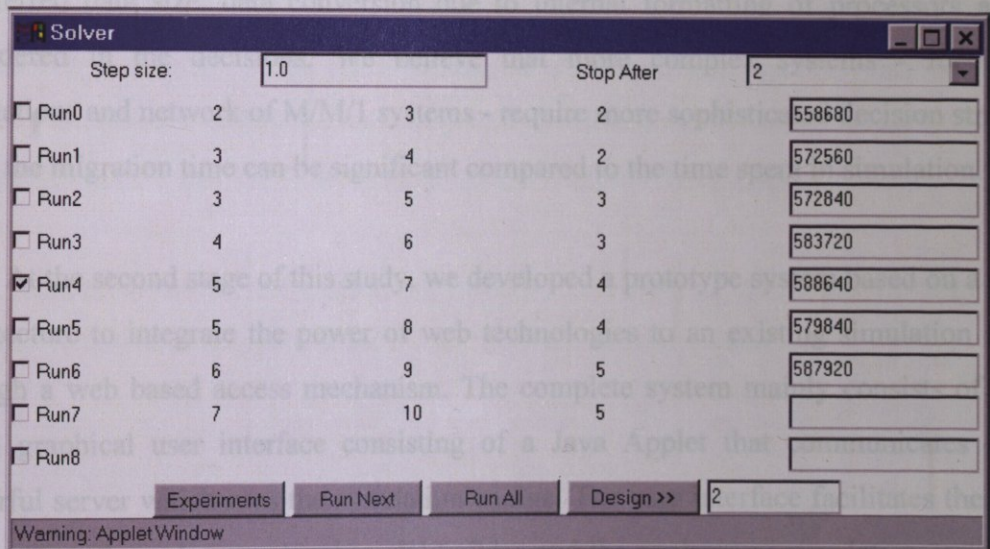


Figure 6.13. Proposed experiments for the line search

7. CONCLUSIONS

In this thesis, we studied the distributed implementation of SC method in the UNIX environment with static and dynamic load balancing techniques. The primary conclusion is that investigations produced a promising result of a sublinear speedup as more workstations are added to the run. The results coincide well with the ones witnessed on the network of Macintosh environment. The efficiency obtained is high. This is because we have tested the algorithm in an ideal experimentation environment, and communication cost is not significant compared to the computational work because of the large number of variants.

It is clear from the results that the static load balancing does a good job of variant allocation, because it partitions the set of all variants into subsets based on the workstations performance rating as well as the utilization of events in each variant. However, its application is limited to the models for which the simulated event rate can be determined or at least predicted in advance.

As an alternative to static load balancing, we described a dynamic load balancing approach. Although the dynamic load balancing described in this study has produced promising results, it is simple in the sense that other factors such as the network topology, transferred data size, data conversion due to internal formatting of processors are not considered in the decisions. We believe that more complex systems - for shorter simulations and network of M/M/1 systems - require more sophisticated decision strategies since the migration time can be significant compared to the time spent in simulation.

At the second stage of this study, we developed a prototype system based on a simple architecture to integrate the power of web technologies to an existing simulation system through a web based access mechanism. The complete system mainly consists of a web based graphical user interface consisting of a Java Applet that communicates with a powerful server which runs the simulation engine. The user interface facilitates the model construction through a graphical model builder and the analysis of simulation results in a platform independent way, while simulation engine executes long running and expensive

simulation experiments on a network of Unix workstations. The system described in the study enables the user to access the simulation environment over the internet, construct simple models and carry out the desired experiments.

As an extension to this study, we build a web based user automated optimization system by including two new components: a design of experiment module and an optimizer module. First, the simulation models are constructed by the model development component and the design of experiment component provides a user friendly interface to select a design type in which factors and their corresponding levels are identified among the decision variables, and a user-automated optimization process based on the response surface methodology is implemented.

The system is essentially a server and a fat client configuration. Although Internet technology is improving to allow more files to run on client browsers using Java, the system could be modified to implement the design and optimizer components on the server site. Hence an automated RSM algorithm with no user intervention can then be included in the system.

The work presented here is limited to simple factorial designs. Designs that are used for fitting second-order models such as central composite design, Box-Behnken designs, should be included. In addition, we do not focus on the selection of the type of the design, but the optimization of the output function through the appropriate factor levels. When the methodology of DOE is applied to determine the significant factors, with the aim of reducing the number of factors (screening design), there are many other issues for selecting the number of runs and the factor settings such as random design, optimal design.

Simulation results and models are currently saved on the server site in binary form. The system can be changed to allow a user to save results into database.

7.1. Contributions of the Study

This thesis aims at presenting the work I have done in the field of Distributed Implementation of Standard Clock during my Ph.D. study at the Department of Industrial Engineering at Boğaziçi University. Most of the results discussed herein has previously been presented at international conferences and published in a journal and other proceedings.

The first part of this study discusses a simulation system based on the distributed implementation of standard clock method on a network of heterogeneous Unix workstations with emphasis to load balancing. The simulation engine and the work on load balancing mechanisms in Chapter 4 were presented in two papers. The description of a static load balancing along with some preliminary performance results were presented in the *10th European Simulation Symposium (ESS'98)* in Nottingham, England. Later, the description of dynamic load balancing is presented in the *Conference on Simulation Methods and Applications: Distributed/Parallel Simulation and Real-Time Applications (CSMA '98)*, in Orlando, USA, 1998. A detailed implementation of the simulation engine and the comparison of the load balancing mechanisms are published as a technical report of the FBE, 1999. A more complete description of load balancing mechanisms developed and the simulation engine are published in *Simulation Practice and Theory*, in September, 2000. The prototype for a web accessible system enabling the user to access the simulation environment over the internet, corresponding to Chapter 5 has been submitted to *Simulation* journal. Finally, the work on Chapter 6, the system coupled with the experimental design methodology namely, factorial design and response surface based optimization techniques is presented in the *International Symposium on Computer and Information Sciences (ISCIS XVII)*, in Orlando, USA, 2002.

APPENDIX A: VERIFICATION OF THE SC ALGORITHM

A.1. M/M/1/ ∞ System

Arrival rate $\lambda=5.0$ and service rate $\mu=6.0$.

THEORETICAL RESULT

$$L=5.0 \quad L_q = 4.164 \quad W=1.0 \quad W_q = 0.834 \quad \rho = 0.834$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.8318 Dev: 0.374 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len: 4.104 Dev: 5.349 Wait: 0.823 Cur: 9 Max: 62

A Events Accepted : 249300
 A Events Rejected : 0
 D Events Accepted : 249290
 D Events Rejected : 50651
 Total Entity created : 249300

PERCENT Accepted : 0.9078

System Throughput : 249290
 System Time Mean: 0.99003 Dev : 1.00550

A.2. M/M/2/ ∞ System

Arrival rate $\lambda=2.0$ and service rate $\mu=1.5$.

THEORETICAL RESULT

$$L=2.4 \quad L_q = 1.07 \quad W=1.2 \quad W_q = 0.533 \quad \rho = 0.667$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 1.3338 Dev: 0.790 Bl:0.0000 Ba: 0 Cur: 1 Max: 2
 QUEUE: Len: 1.085 Dev: 2.125 Wait: 0.540 Cur: 0 Max: 27

A Events Accepted : 100431
 D Events Accepted : 100430
 D Events Rejected : 49791
 Total Entity created : 100431

PERCENT Accepted : 0.8014

System Throughput : 100430
 System Time Mean: 1.20410 Dev : 1.13229

A.3. M/M/1/8 System

Arrival rate $\lambda=1.0$ and service rate $\mu=1.0$.

THEORETICAL RESULT

$$L_q = 3.6 \quad W = 5.0 \quad W_q = 4.0 \quad \rho = 0.9$$

SC RESULT

Simulation Time: 100000.00

Results of Node No: 0

SERVICE: Util: 0.8995 Dev: 0.301 Bl:0.0000 Ba:10125 Cur: 1 Max: 1
 QUEUE: Len.: 3.594 Dev: 2.732 Wait: 4.001 Cur: 8 Max: 8

A Events Accepted : 89812
 A Events Rejected : 10125
 D Events Accepted : 89803
 D Events Rejected : 9979
 Total Entity created : 89812

PERCENT Accepted : 0.8993

System Throughput : 89803

System Time Mean: 5.00308 Dev : 3.43101

A.4. M/G/1/ ∞ System

Arrival rate $\lambda=0.4$ and general service time distribution with $\mu^{-1}=2$, $\sigma^2=0.5$

THEORETICAL RESULT

$$L_q = 1.8 \quad W = 6.5 \quad W_q = 4.5 \quad \rho = 0.8$$

SC RESULT

Simulation Time: 100000.00

Results of Node No: 0

SERVICE: Util: 0.7971 Dev: 0.402 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len.: 1.801 Dev: 2.539 Wait: 4.511 Cur: 0 Max: 21

A Events Accepted : 39923
 A Events Rejected : 0
 D Events Accepted : 39922
 D Events Rejected : 102091
 Total Entity created : 39923

PERCENT Accepted : 0.4389

System Throughput : 39922

System Time Mean: 6.50775 Dev : 5.38286

A.5. M/M/1/ ∞ with 2 Distinct Class of Entities

Arrival rates $\lambda_1=1.0$ $\lambda_2=2.0$ and service rates $\mu_1=3.0$ $\mu_2=4.0$

A.5.1. FIFO Queue Discipline with Regular Service

SLAM RESULT

CURRENT TIME .5000E+05
STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.171E+01	.177E+01	.103E+01	.000E+00	.156E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	QUEUE	4.308	5.693	60	0	1.435

SERVICE ACTIVITY STATISTICS

ACT NUM	ACT LABEL OR START NODE	SER AVERAGE CAP UTIL	STD DEV	CUR AVERAGE UTIL	MAX IDL BLOCK	MAX BSY TME/SER	ENT TME/SER	CNT
1	QUEUE	1 .836	.37	0	.00	2.75	139.62	****

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0
SERVICE: Util: 0.8355 Dev: 0.371 B1:0.0000 Ba: 0 Cur: 1 Max: 1
QUEUE: Len: 4.273 Dev: 5.500 Wait: 1.423 Cur: 27 Max: 55

A Events Accepted : 150142
A Events Rejected : 0
D Events Accepted : 150114
D Events Rejected : 49358
Total Entity created : 150142

PERCENT Accepted : 0.8588

System Throughput : 150114
System Time Mean: 1.70105 Dev : 1.72147

A.5.2. Priority Queue Discipline with Regular Service

SLAM RESULT

CURRENT TIME .5000E+05
 STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.186E+01	.248E+01	.135E+01	.000E+00	.227E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	AWAIT	.357	.744	8	0	.356
2	AWAIT	4.318	6.109	56	0	2.159

REGULAR ACTIVITY STATISTICS

ACTIVITY INDEX/LABEL	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL	ENTITY COUNT
1	.8357	.3705	1	0	150100

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTIL	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL
1	SERVER	1	.84	.371	1	0

RESOURCE NUMBER	RESOURCE LABEL	CURRENT AVAILABLE	AVERAGE AVAILABLE	MINIMUM AVAILABLE	MAXIMUM AVAILABLE
1	SERVER	1	.1643	0	1

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.8372 Dev: 0.369 Bl:0.0000 Ba: 0 Cur: 1 Max: 1

QUEUE:

C: 1 Len: 0.358 Dev: 0.751 Wait: 0.358 Cur: 2 Max: 10

C: 2 Len: 4.429 Dev: 6.243 Wait: 2.210 Cur: 29 Max: 61

GEN: Len: 4.787 Dev: 6.417 Wait: 1.593 Cur: 31 Max: 61

A Events Accepted : 150142

A Events Rejected : 0

D Events Accepted : 150110

D Events Rejected : 49362

Total Entity created : 150142

PERCENT Accepted : 0.8588

System Throughput : 150110

System Time Mean: 1.87205 Dev : 2.55412

A.5.3. Priority Queue Discipline with Preemptive Service

SLAM RESULT

CURRENT TIME .5000E+05
 STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.188E+01	.257E+01	.138E+01	.000E+00	.227E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	PRE PREEMPT	.166	.550	8	0	.166
2	AWAIT	4.569	6.218	57	0	2.285

REGULAR ACTIVITY STATISTICS

ACTIVITY INDEX/LABEL	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL	ENTITY COUNT
1	.8357	.3705	1	0	150100

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTIL	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL
1	SERVER	1	.84	.371	1	0

RESOURCE NUMBER	RESOURCE LABEL	CURRENT AVAILABLE	AVERAGE AVAILABLE	MINIMUM AVAILABLE	MAXIMUM AVAILABLE
1	SERVER	1	.1642	0	1

SC RESULT

RESULTS of EXP #: 1

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.8378 Dev: 0.369 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE:

C: 1 Len: 0.167 Dev: 0.565 WaitT: 0.167 Cur: 2 Max: 10

C: 2 Len: 4.698 Dev: 6.306 WaitT: 2.345 Cur: 27 Max: 60

GEN: Len: 4.868 Dev: 6.410 WaitT: 1.620 Cur: 29 Max: 60

A Events Accepted : 150142

A Events Rejected : 0

D Events Accepted : 150112

D Events Rejected : 49360

Total Entity created : 150142

PERCENT Accepted : 0.8588

System Throughput : 150112

System Time Mean: 1.89953 Dev : 2.62026

A.6. Hospital Emergency Room Problem: 3 Different Classes of Patients

Arrival rates of patients are $\lambda_1=0.2$, $\lambda_2=0.6$, $\lambda_3=1.2$ and service rate of doctors $\mu=3$.

Table A.1. Theoretical waiting time values

	Preemptive Priorities		Nonpreemptive Priorities	
# doctors	1	2	1	2
$W_q(1)$	0.024	0.00037	0.238	0.029
$W_q(2)$	0.154	0.00793	0.325	0.033
$W_q(3)$	1.033	0.06542	0.889	0.048

A.6.1. One Doctor with Preemptive Service

SLAM RESULT

```

CURRENT TIME      .5000E+05
STATISTICAL ARRAYS CLEARED AT TIME  .0000E+00

**STATISTICS FOR VARIABLES BASED ON OBSERVATION**
      MEAN      STANDARD      COEFF. OF      MINIMUM      MAXIMUM      NO.OF
      VALUE      DEVIATION      VARIATION      VALUE      VALUE      OBS
      .102E+01      .000E+00      .000E+00      .000E+00      .170E+02      ****

**FILE STATISTICS**
FILE
NUMBER LABEL/TYPE      AVERAGE      STANDARD      MAXIMUM      CURRENT      AVERAGE
      LENGTH      DEVIATION      LENGTH      LENGTH      WAIT TIME
1 PRE3 PREEMPT      .005      .075      4      0      .024
2 PRE2 PREEMPT      .089      .373      7      0      .150
3 AW AWAIT      1.281      2.193      21      0      1.066

**RESOURCE STATISTICS**
RESOURCE RESOURCE CURRENT AVERAGE STANDARD MAXIMUM CURRENT
NUMBER LABEL CAPACITY UTIL DEVIATION UTIL UTIL
1 DOKTOR 1 .67 .471 1 0

```

SC RESULT

Simulation Time: 50000.00

```

Results of Node No: 0
SERVICE: Util: 0.6629 Dev: 0.473 B1:0.0000 Ba: 0 Cur: 1 Max: 1
QUEUE:
C: 1 Len: 0.005 Dev: 0.074 WaitT: 0.024 Cur: 0 Max: 3
C: 2 Len: 0.094 Dev: 0.389 WaitT: 0.157 Cur: 0 Max: 8
C: 3 Len: 1.215 Dev: 2.098 WaitT: 1.018 Cur: 3 Max: 25
GEN : Len: 1.313 Dev: 2.218 WaitT: 0.660 Cur: 3 Max: 25

```

PERCENT Accepted : 0.7985

System Throughput : 99565
System Time Mean: 0.99246 Dev : 1.30012

A.6.2. Two Doctors with Preemptive Service

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.6651 Dev: 0.745 Bl:0.0000 Ba: 0 Cur: 2 Max: 2

QUEUE:

C: 1 Len: 0.000 Dev: 0.014 Wait: 0.001 Cur: 0 Max: 3
 C: 2 Len: 0.005 Dev: 0.082 Wait: 0.009 Cur: 1 Max: 4
 C: 3 Len: 0.077 Dev: 0.378 Wait: 0.065 Cur: 0 Max: 9
 GEN : Len: 0.083 Dev: 0.400 Wait: 0.042 Cur: 1 Max: 11

A Events Accepted : 99702 A Events Rejected : 0

D Events Accepted : 99699 D Events Rejected : 200847

Total Entity created : 99702

PERCENT Accepted : 0.4982

System Throughput : 99699

System Time Mean: 0.37503 Dev : 0.37610

A.6.3. One Doctor with Nonpreemptive Service

SLAM RESULT

CURRENT TIME .5000E+05

STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.101E+01	.000E+00	.000E+00	.000E+00	.140E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	TYP1 QUEUE	.048	.230	4	0	.238
2	TYP2 QUEUE	.193	.522	10	0	.326
3	TYP3 QUEUE	1.106	2.036	26	0	.926

SERVICE ACTIVITY STATISTICS

ACT NUM	ACT START	LABEL OR NODE	SER CAP	AVERAGE UTIL	STD DEV	CUR UTIL	AVERAGE BLOCK	MAX TME/SER	IDL TME/SER	MAX BSY	ENT CNT
1	SEL	SELECT	1	.664	.47	0	.00	5.55	49.42	****	

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.6629 Dev: 0.473 Bl:0.0000 Ba: 0 Cur: 1 Max: 1

A.6.4. Two Doctors with Nonpreemptive Service

SLAM RESULT

CURRENT TIME .5000E+05
 STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.375E+00	.000E+00	.000E+00	.000E+00	.402E+01	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	TYP1 QUEUE	.006	.078	3	0	.029
2	TYP2 QUEUE	.019	.153	4	0	.033
3	TYP3 QUEUE	.057	.309	8	0	.048

SERVICE ACTIVITY STATISTICS

ACT NUM	ACT START	LABEL OR NODE	SER CAP	AVERAGE UTIL	STD DEV	CUR UTIL	AVERAGE BLOCK	MAX TME/SER	IDL TME/SER	MAX BSY	ENT CNT
1	SEL	SELECT	2	.664	.75	0	.00	2.00	2.00	****	

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.6650 Dev: 0.745 Bl:0.0000 Ba: 0 Cur: 2 Max: 2
 QUEUE:

C: 1 Len: 0.006 Dev: 0.078 Wait: 0.028 Cur: 1 Max: 4
 C: 2 Len: 0.020 Dev: 0.156 Wait: 0.033 Cur: 0 Max: 4
 C: 3 Len: 0.057 Dev: 0.314 Wait: 0.048 Cur: 0 Max: 8
 GEN : Len: 0.083 Dev: 0.400 Wait: 0.042 Cur: 1 Max: 11

A Events Accepted : 99702
 A Events Rejected : 0
 D Events Accepted : 99699
 D Events Rejected : 200847
 Total Entity created : 99702

PERCENT Accepted : 0.4982

System Throughput : 99699
 System Time Mean: 0.37503 Dev : 0.36275

A.6.5. One Doctor with FIFO Queue Discipline

SLAM RESULT

CURRENT TIME .5000E+05
 STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.102E+01	.000E+00	.000E+00	.000E+00	.158E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	AW AWAIT	1.375	2.303	21	0	.688
2	CALENDAR	3.667	.471	5	3	.229

REGULAR ACTIVITY STATISTICS

ACTIVITY INDEX/LABEL	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL	ENTITY COUNT
1	.6670	.4713	1	0	99943

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTIL	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL
1	DOKTOR	1	.67	.471	1	0

RESOURCE NUMBER	RESOURCE LABEL	CURRENT AVAILABLE	AVERAGE AVAILABLE	MINIMUM AVAILABLE	MAXIMUM AVAILABLE
1	DOKTOR	1	.3329	0	1

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.6629 Dev: 0.473 B1:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len.: 1.313 Dev: 2.218 Wait: 0.660 Cur: 3 Max: 25

A Events Accepted : 99569
 A Events Rejected : 0
 D Events Accepted : 99565
 D Events Rejected : 50244
 Total Entity created : 99569

PERCENT Accepted : 0.7985

System Throughput : 99565
 System Time Mean: 0.99246 Dev : 1.00058

A.6.6. Two Doctors with FIFO Queue Discipline

SLAM RESULT

CURRENT TIME .5000E+05
 STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.376E+00	.000E+00	.000E+00	.000E+00	.396E+01	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	AW AWAIT	.084	.395	8	0	.042

REGULAR ACTIVITY STATISTICS

ACTIVITY INDEX/LABEL	AVERAGE UTILIZATION	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL	ENTITY COUNT
1	.6671	.7462	2	0	99943

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTIL	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL
1	DOKTOR	2	.67	.746	2	0

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.6650 Dev: 0.745 Bl:0.0000 Ba: 0 Cur: 2 Max: 2
 QUEUE: Len.: 0.083 Dev: 0.400 Wait: 0.042 Cur: 1 Max: 11

A Events Accepted : 99702
 A Events Rejected : 0
 D Events Accepted : 99699
 D Events Rejected : 200847
 Total Entity created : 99702

PERCENT Accepted : 0.4982

System Throughput : 99699
 System Time Mean: 0.37503 Dev : 0.35949

A.7. G/M/1/ ∞ with Two Distinct Classes of Entities

Arrival-time distributions are $U(4,10)$ and $U(1, 7)$ and service rates $\mu_1=\mu_2=0.5$ for class 1 and class 2 respectively.

SLAM RESULT

CURRENT TIME .5000E+05
STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.579E+01	.000E+00	.000E+00	.000E+00	.507E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	QUEUE	1.493	2.111	18	0	3.791
2	CALENDAR	2.786	.410	4	3	1.244

SERVICE ACTIVITY STATISTICS

ACT NUM	ACT START	LABEL OR NODE	SER CAP	AVERAGE UTIL	STD DEV	CUR UTIL	AVERAGE BLOCK	MAX TME/SER	IDL TME/SER	MAX BSY	ENT CNT
1		QUEUE	1	.786	.41	1	.00	6.79	493.50	****	

SC RESULT

RESULTS of EXP #: 1

Simulation Time: 50000.00

Results of Node No: 0
SERVICE: Util: 0.7871 Dev: 0.409 B1:0.0000 Ba: 0 Cur: 0 Max: 1
QUEUE: Len: 1.493 Dev: 2.092 Wait: 3.808 Cur: 0 Max: 16

A Events Accepted : 19598
A Events Rejected : 38073
D Events Accepted : 19598
D Events Rejected : 5370
Total Entity created : 19598

PERCENT Accepted : 0.4743

System Throughput : 19598
System Time Mean: 5.81590 Dev : 5.49957

A.8. G/G/1/∞ with Two Distinct Classes of Entities

Arrival-time distributions are U(4,10) and U(1,7), service time distribution are U(0,4) and U(0, 2) for class 1 and class 2 entities respectively.

SLAM RESULT

CURRENT TIME .5000E+05
STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION
MEAN STANDARD COEFF. OF MINIMUM MAXIMUM NO.OF
VALUE DEVIATION VARIATION VALUE VALUE OBS
.175E+01 .000E+00 .000E+00 .000E+00 .731E+01 ****

FILE STATISTICS
FILE AVERAGE STANDARD MAXIMUM CURRENT AVERAGE
NUMBER LABEL/TYPE LENGTH DEVIATION LENGTH LENGTH WAIT TIME
1 QUEUE .154 .389 5 0 .391

SERVICE ACTIVITY STATISTICS
ACT ACT LABEL OR SER AVERAGE STD CUR AVERAGE MAX IDL MAX BSY ENT
NUM START NODE CAP UTIL DEV UTIL BLOCK TME/SER TME/SER CNT
1 QUEUE 1 .536 .50 1 .00 6.89 30.04 ****

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0
SERVICE: Util: 0.5366 Dev: 0.499 Bl:0.0000 Ba: 0 Cur: 0 Max: 1
QUEUE: Len: 0.158 Dev: 0.410 Wait: 0.402 Cur: 0 Max: 4

A Events Accepted : 19628
A Events Rejected : 38060
D Events Accepted : 19628
D Events Rejected : 66631
Total Entity created : 19628

PERCENT Accepted : 0.2727

System Throughput : 19628
System Time Mean: 1.76910 Dev : 1.25936

A.9. Network of 4 M/M/1/ ∞ Systems

SLAM RESULT

SLAM II SUMMARY REPORT

SIMULATION PROJECT NETWORK_OF_4_QUEUES

CURRENT TIME .4000E+05

STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.359E+01	.000E+00	.000E+00	.703E-01	.203E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	AW1 AWAIT	2.386	3.409	39	0	.792
2	AW2 AWAIT	.585	1.245	14	0	.447
3	AW3 AWAIT	2.689	3.761	33	0	1.576
4	AW4 AWAIT	2.379	3.507	38	0	.789

RESOURCE STATISTICS

RESOURCE NUMBER	RESOURCE LABEL	CURRENT CAPACITY	AVERAGE UTIL	STANDARD DEVIATION	MAXIMUM UTIL	CURRENT UTIL
1	RES1	1	.76	.429	1	1
2	RES2	1	.52	.500	1	1
3	RES3	1	.77	.422	1	0
4	RES4	1	.75	.432	1	1

Simulation Time: 40000.00

Results of Node No: 0

SERVICE: Util: 0.7566 Dev: 0.429 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len.: 2.365 Dev: 3.495 WaitT: 0.783 Cur: 5 Max: 49

Results of Node No: 1

SERVICE: Util: 0.5188 Dev: 0.500 Bl:0.0000 Ba: 0 Cur: 0 Max: 1
 QUEUE: Len.: 0.573 Dev: 1.221 WaitT: 0.437 Cur: 0 Max: 13

Results of Node No: 2

SERVICE: Util: 0.7726 Dev: 0.419 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len.: 2.739 Dev: 3.800 WaitT: 1.601 Cur: 5 Max: 31

Results of Node No: 3

SERVICE: Util: 0.7553 Dev: 0.430 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len.: 2.309 Dev: 3.306 WaitT: 0.764 Cur: 1 Max: 35

A Events Accepted : 120849

A Events Rejected : 0

D Events Accepted : 362515

D Events Rejected : 197555

Total Entity created : 120849

PERCENT Accepted : 0.7099

System Throughput : 120835

System Time Mean: 3.57124 Dev : 2.29202

SLAM RESULT

SLAM II SUMMARY REPORT

SIMULATION PROJECT NETWORK_OF_4_QUEUES

CURRENT TIME .1000E+06

STATISTICAL ARRAYS CLEARED AT TIME .0000E+00

STATISTICS FOR VARIABLES BASED ON OBSERVATION

MEAN VALUE	STANDARD DEVIATION	COEFF. OF VARIATION	MINIMUM VALUE	MAXIMUM VALUE	NO.OF OBS
.273E+02	.100E+02	.368E+00	.115E+01	.774E+02	****

FILE STATISTICS

FILE NUMBER	LABEL/TYPE	AVERAGE LENGTH	STANDARD DEVIATION	MAXIMUM LENGTH	CURRENT LENGTH	AVERAGE WAIT TIME
1	Q1 QUEUE	10.759	5.446	18	14	11.813
2	Q2 QUEUE	3.939	2.771	8	8	8.687
3	Q3 QUEUE	1.083	1.632	7	0	2.369
4	Q4 QUEUE	5.517	4.909	15	3	6.059

SERVICE ACTIVITY STATISTICS

ACT NUM	ACT START	LABEL OR NODE	SER CAP	AVERAGE UTIL	STD DEV	CUR UTIL	AVERAGE BLOCK	MAX IDL TME/SER	MAX BSY TME/SER	ENT CNT
1	Q1	QUEUE	1	.453	.50	0	.06	19.67	19.74	****
2	Q1	QUEUE	1	.457	.50	1	.01	24.84	19.67	****
3	Q2	QUEUE	1	.912	.28	1	.02	17.32	636.70	****
4	Q3	QUEUE	1	.685	.46	1	.03	24.73	126.58	****
5	Q4	QUEUE	1	.911	.28	1	.00	7.51	861.04	****

RESULTS of EXP #: 0

Simulation Time: 100000.00

Results of Node No: 0

SERVICE: Util: 0.9104 Dev: 0.286 Bl:0.0675 Ba:9050 Cur: 1 Max: 1
 QUEUE: Len.: 10.870 Dev: 5.423 Wait: 11.920 Cur: 8 Max: 18

Results of Node No: 1

SERVICE: Util: 0.9064 Dev: 0.291 Bl:0.0195 Ba:4645 Cur: 1 Max: 1
 QUEUE: Len.: 3.854 Dev: 2.822 Wait: 8.511 Cur: 4 Max: 8

Results of Node No: 2

SERVICE: Util: 0.6877 Dev: 0.463 Bl:0.0217 Ba:747 Cur: 0 Max: 1
 QUEUE: Len.: 1.161 Dev: 1.732 Wait: 2.530 Cur: 0 Max: 7

Results of Node No: 3

SERVICE: Util: 0.9087 Dev: 0.288 Bl:0.0000 Ba:3108 Cur: 1 Max: 1
 QUEUE: Len.: 5.289 Dev: 4.762 Wait: 5.800 Cur: 11 Max: 15

A Events Accepted : 91187

A Events Rejected : 9050

D Events Accepted : 273512

D Events Rejected : 168037

Total Entity created : 91187

PERCENT Accepted : 0.6731

System Throughput : 91161

System Time Mean: 27.08668 Dev : 9.72301

A.10. Markovian System with Bulk Arrival ($M^B/M/1$)

Arrival rate $\lambda=1.0$ and service rate $\mu=6.0$.

THEORETICAL RESULT

$$b=3 \quad \rho=0.5 \quad L=2.0 \quad L_q=1.5 \quad W=0.67 \quad W_q=0.5$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.5028 Dev: 0.500 Bl:0.0000 Ba: 0 Cur: 0 Max: 1
 QUEUE: Len.: 1.498 Dev: 2.619 Wait: 0.497 Cur: 0 Max: 33

A Events Accepted : 50254

D Events Accepted : 150762 D Events Rejected : 149088

Total Entity created : 150762

PERCENT Accepted : 0.5742

System Throughput : 150762

System Time Mean: 0.66356 Dev : 0.60720

THEORETICAL RESULT

$$b=4 \quad \rho=0.67 \quad L=5.0 \quad L_q=4.34 \quad W=1.25 \quad W_q=1.94$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.6703 Dev: 0.470 Bl:0.0000 Ba: 0 Cur: 0 Max: 1
 QUEUE: Len.: 4.359 Dev: 6.081 Wait: 1.084 Cur: 0 Max: 69

A Events Accepted : 50254 D Events Accepted : 201016

D Events Rejected : 98834

Total Entity created : 201016

PERCENT Accepted : 0.7177

System Throughput : 201016

System Time Mean: 1.25095 Dev : 1.16699

THEORETICAL RESULT

$$b=5 \quad \rho=0.83 \quad L=15.0 \quad L_q=14.13 \quad W=3.0 \quad W_q=2.93$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.8378 Dev: 0.369 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len.: 14.622 Dev: 16.539 Wait: 2.910 Cur: 0 Max: 121

A Events Accepted : 50254

D Events Accepted : 251269

D Events Rejected : 48581

Total Entity created : 251270

PERCENT Accepted : 0.8612

System Throughput : 251269

System Time Mean: 3.07631 Dev : 2.88072

A.11. Markovian System with Bulk Service (M / M^B / 1)

Arrival rate $\lambda=20.0$ and service rate $\mu=12.0$.

THEORETICAL RESULT

$$B=2.0 \quad r=0.884 \quad L = \frac{r}{(1-r)} = 7.6 \quad W = \frac{r}{\lambda(1-r)} = 0.38 \quad L_q = r^B L = 5.9 \quad W_q = r^B W = 0.295$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.8842 Dev: 0.320 Bl:0.0000 Ba: 0 Cur: 0 Max: 1
 QUEUE: Len: 5.941 Dev: 7.790 Wait: 0.297 Cur: 0 Max: 88

A Events Accepted : 999675

D Events Accepted : 530578

D Events Rejected : 69219

Total Entity created : 999675

PERCENT Accepted : 0.9567

System Throughput : 999675

System Time Mean: 0.38047 Dev : 0.38032

THEORETICAL RESULT

$$B=3.0 \quad r=0.81 \quad L = \frac{r}{(1-r)} = 4.26 \quad W = \frac{r}{\lambda(1-r)} = 2.13 \quad L_q = r^B L = 2.26 \quad W_q = r^B W = 1.13$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.8137 Dev: 0.389 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len: 2.293 Dev: 4.000 Wait: 1.150 Cur: 1 Max: 44

A Events Accepted : 99701

D Events Accepted : 40334

D Events Rejected : 9321

Total Entity created : 99701

PERCENT Accepted : 0.9376

System Throughput : 99697

System Time Mean: 2.15897 Dev: 2.16192

THEORETICAL RESULT

$$B=4.0 \quad r=0.74 \quad L = \frac{r}{(1-r)} = 2.86 \quad W = \frac{r}{\lambda(1-r)} = 1.43 \quad L_q = r^B L = 0.86 \quad W_q = r^B W = 0.43$$

SC RESULT

Simulation Time: 50000.00

Results of Node No: 0

SERVICE: Util: 0.7433 Dev: 0.437 Bl:0.0000 Ba: 0 Cur: 1 Max: 1
 QUEUE: Len: 0.852 Dev: 2.000 Wait: 0.427 Cur: 0 Max: 32

A Events Accepted : 99701

D Events Accepted : 36793

D Events Rejected : 12862

Total Entity created : 99701

PERCENT Accepted : 0.9139

System Throughput : 99697

System Time Mean: 1.43585 Dev: 1.42532

APPENDIX B: FORMAT OF THE CD CONTAINING SOURCE CODE OF THE SIMULATION ENGINE AND FRONT-END GUI IMPLEMENTATIONS

The CD contains the following material;

Simulation Engine codes in two separate directories: `Static` and `Dynamic`, in each of these directories, there exist two subdirectories named as `Server` and `Client`.

Java Applet front-end GUI named as `mythesis.java` as well as the necessary html file `test.html` are placed in the directory `FrontEnd`.

In addition, `Readme.txt` file contains the information more detailed information about the directory structure of the CD and information on how to install the files and run the programs.

REFERENCES

- Bhatti, G. and P. Vakili, 1997, "Parallel/Distributed Simulation for Parametric Study of Wireless Communication Networks", Computer Science Dept., Boston University, obtained through personal communication, (e-mail: bhatti@bu.edu).
- Burdorf, C. and J. Marti, 1993, "Load Balancing Strategies for Time Warp on Multi-User Workstation", *The Computer Journal*, Vol. 36, pp. 168-176.
- Buss, A. H. and K. A. Stork, 1996, "Discrete Event Simulation on the World Wide Using Java", *Proceedings of 1996 Winter Simulation Conference*, J.M. Charnes, D.J. Morrice, D.T. Brunner, J.J. Swain (Eds.), Coronado, pp. 780-785.
- Calinescu, R. and D. J. Evans, 1994, "A Parallel Simulation Model for Load Balancing in Clustered Distributed Systems", *Parallel Computing*, Vol. 20, pp. 77-91.
- Carothers, C. D. and R. M. Fujimoto, 1996, "Background Execution of Time Warp Programs", *Proc. of the 9th Workshop on Parallel and Distributed Simulation*, pp. 12-19.
- Chen, C. H. and Y. C. Ho., 1995, "An Approximation Approach of the Standard Clock Method for General Discrete-Event Simulation", *IEEE Transactions on Control Systems Technology*, Vol. 3, No. 3, pp. 309-317.
- Cybenko, G., 1989, "Dynamic Load Balancing For Distributed Memory Multiprocessors", *Journal of Parallel and Distributed Computing*, Vol. 7, pp. 279-301.
- Dobson, G., 1984, "Scheduling Independent Tasks on Uniform Processors", *SIAM J. Computing*, Vol. 13, No. 4, pp. 705-716.
- Evans, D. J. and W. U. N. Butt, 1994, "Local Balancing with Network Partitioning Using Host Groups", *Parallel Computing*, Vol. 20, pp. 325-345.
- Fishwick, P., 1996 "Web-based Simulation: Some Personal Observations", *Proceedings of Winter Simulation Conference*, San Diego, pp. 772-779.

- Franklin, M. and V. Govindan, 1996, "A General Matrix Iterative Model for Dynamic Load Balancing", *Parallel Computing*, Vol. 22, pp. 969-989.
- Fujimoto, R. M., 1990, "Parallel Discrete Event Simulation", *Communication of ACM*, Vol. 33, No. 10, pp 31-53.
- Glazer, D. W. and C. Tropper, 1993, "On Process Migration and Load Balancing in Time Warp", *IEEE Trans. Parallel and Distributed Systems*, Vol. 4, No. 3, pp. 318-327.
- Hamdi, M. and C. K. Lee, 1997, "Dynamic Load-Balancing of Image Processing Applications on Clusters Of Workstations", *Parallel Computing*, Vol. 22, pp. 1477-1492.
- Healy, K. J. and R. A. Kilgore, 1997, "Silk™:A Java-Based Process Simulation Language", *Proceedings of Winter Simulation Conference*, S. Andradottir, K.J. Healy, D.H. Withers, B.L. Nelson (Eds), pp. 475-482.
- Healy, K. J. and R. A. Kilgore, 1998, "Introduction to Silk and Java-based Simulation", *Proceedings of the 1998 Winter Simulation Conference*, pp. 1649-1654.
- Ho, Y. C. and X. Cao, 1991, *Perturbation Analysis of Discrete Event Dynamic Systems*, Kluwer Publishing.
- Ho, Y. C., R. S. Sreenivas and P. Vakili, 1992, "Ordinal Optimization of DEDS", *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 2, pp. 61-88.
- Howell, F. W., 1999, The Simjava Home Page, <http://www.dcs.ed.ac.uk/home/hase/simjava/Javasim>
- Hu, J. Q., 1995, "Parallel Simulation of DEDS Via Event Synchronization", *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 5, pp. 167-186.
- Kaylan, A. R., S. Ziya and O. Yilmaz, 1997, "Web-based Architecture for Simulation Modeling and Analysis", *Proceeding of the 11th European Simulation Multiconference*, Lehman & Kaylan, (Eds), pp. 115-119.

- Kaylan, A. R., S. Ziya and E. Kemahlioğlu, 1998, "Web Based Process Management and Simulation Tool", *Proceeding of the 10th European Simulation Symposium*, Andre Bargiela & Eugene Kerckhoffs, (Eds), pp. 433-438.
- Kelton, W. D., 2000, "Experimental Design for Simulation", *Proceedings of the Winter Simulation Conference*, pp. 32-38.
- Khalil, E. and C. Tropper, 1999, "On Metrics for the Dynamic Load Balancing of Optimistic Simulations", *IEEE Proceedings of the 32nd Hawaii International Conference on System Sciences*, pp. 1-13.
- Kilgore, R. A., K. J. Healy and G. B. Kleindorfer, 1998, "The Future of Java-based Simulation", *Proceedings of 1998 Winter Simulation Conference*, D.J. Medeiros, E.F. Watson, J.s. Carson, M. S. Manivannan (Eds), pp. 1707-1712.
- Kleijnen, J. P. C., 1998, "Experimental design for sensitivity analysis, optimization, and validation of simulation models", *Handbook of Simulation*, Wiley, New York, pp. 173-223.
- Klein, U., S. Strassburger and J. Beikirch, 1998, "Distributed Simulation with JavaGPSS based on the High Level Architecture", *Proceedings of the 1998 International Conference on Web-Based Modeling & Simulation*, San Diego, CA.
- Kleinrock, L., 1975, *Queueing Systems*, Wiley, NewYork.
- Kohring, G. A., 1995, "Dynamic Load Balancing for Parallelized Particle Simulations on MIMD Computers", *Parallel Computing*, Vol. 21 pp. 683-693.
- Law, M. A. and W. D. Kelton, 2000, *Simulation Modeling and Analysis*, 3rd Edition. NewYork, McGraw Hill.
- Law, M. A., and M. G. Mc Comas, 2000, "Simulation Based Optimization", *Proceedings of the Winter Simulation Conference*, pp. 46-48.
- Lu, H. and K. Tan, 1994, "Load-Balanced Join Processing in Shared-Nothing Systems", *Journal of Parallel and Distributed Computing*, Vol. 23, pp. 382-398.

- Miller, J. A., A. F. Seila and X. Xiang, 2000, "The JSIM Web-Based Simulation Environment", *Future Generation Computer Systems*, Vol.17, pp. 119-133.
- Misra, J., 1986, "Distributed discrete-event simulation", *Computing Surveys*, Vol 18, pp. 39-65.
- Mollamustafaoğlu, L., 1992, *Standard Clock Method in Parallel Simulation of Several Simulation Experiments*, Research Report, FBE-IE-08/92-10, Boğaziçi University.
- Mollamustafaoğlu, L., 1998, "DISCS: An Object Oriented Framework for Distributed Discrete-Event Simulation with the Standard Clock Method", *Simulation*, Vol.70, No. 2, pp. 75-89.
- Myers, R. H. and D. C. Montgomery, 1995, *Response Surface Methodology: Process and Product Optimization Using Designed Experiment*, J.W. & Sons, New York.
- Nair, R. S., Miller, J. A. and Z. Zang, 1996, "Java-Based Query Driven Simulation Environment", *Proceedings of Winter Simulation Conference*, J.M. Charnes, D.J. Morrice, D.T. Brunner, J.J. Swain (Eds.), Coronado, pp. 786-793.
- Neddermeijer, H. G, Oortmarssen, G. J., Piersma N. and. R. Dekker, 2000, "A Framework For Response Surface Methodology For Simulation Optimization", *Proceedings of the Winter Simulation Conference*, pp. 129-135.
- Nicol, D. M. and R. M. Fujimoto, 1994, "Parallel Simulation Today", in O. Balci (ed.), *Annals of Operations Research*, 53 (J.C. Baltzer AG) pp. 249-285.
- Page, E. H., 1997, *Survey of Web-Based Simulation*, <http://ms.ic.org/websim/survey/>
- Ross, S. M., 1997, *Introduction to Probability Models*, Sixth ed. Academic Press, New York.
- Schlagenhaft, R., M. Runwandl, C. Sporrer and H. Bauer, 1995, "Dynamic Load Balancing of a Multi-Cluster Simulator on a Network of Workstations", *Proceedings of the 9th Workshop on Parallel and Distributed Simulation*, pp. 175-180.

- Selvakumar, S. and C. S. R. Murthy, 1994, "Static Task Allocation of Concurrent Programs for Distributed Computing Systems with Processor and Resource Heterogeneity", *Parallel Computing*, Vol. 20, pp. 835-851.
- Shirazi, B. A., A. R. Hurson, and K. M. Kavi, 1995, "Scheduling and Load Balancing in Parallel and Distributed Systems", *IEEE Computer Society Press*, Silver Spring, MD.
- Strickland, S. G. and R. G. Phelan, 1995, "Massively Parallel SIMD Simulation of Markovian DEDS: Event and Time Synchronous Methods", *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 5, pp. 141-166.
- Vakili, P., 1992, "Massively Parallel and Distributed Simulation of a Class of Discrete Event Systems: A Different Perspective", *ACM Trans. on Modeling and Computer Simulation*, Vol. 2, No. 3, pp. 214-238.
- Vakili, P., L. Mollamustafaoğlu, and Y. C. Ho., 1992, "Massively Parallel Simulation of a Class of Discrete Events", *Proceedings of the 4th Symposium on Frontiers of Massively Parallel Computation*, IEEE Computer Society Press, pp. 412-419.
- Veith, T. L., J. E. Kobza and C. P. Koelling, 1999, "NetSim: Java-based Simulation for the World Wide Web", *Computers & Operations Research*, Vol. 26, pp. 607-621.
- Wang, Y. and R. J. T. Morris, 1985, "Load Sharing in Distributed Systems", *IEEE Transactions in Distributed Systems*, Vol. 34, No.3, pp. 204-217.
- Wieselthier, J. E., C. M. Barnhart and A. Ephremides, 1995, "Standard Clock Simulation and Ordinal Optimization Applied to Admission Control in Integrated Communication Networks", *Discrete Event Dynamic Systems: Theory and Applications*, Vol. 5, pp. 243-280.

