1

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

National Library
of Canada

Bibliothèque nationale
du Canada

Acquisitions and
Bibliographic Services Branch

Direction des acquisitions et
des services bibliographiques

395 Wellington Street
Ottawa, Ontario
K1A 0N4

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

Interval-Based Performance Analysis of Computing Systems

by

Revathy Ramadoss, B.E., M.E.

**A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of the requirements
for the degree of
Master of Engineering**

**Ottawa-Carleton Institute for Electrical Engineering
Faculty of Engineering
Department of Systems and Computer Engineering
Carleton University,
Ottawa, Ontario, Canada**

October 31, 1994

©1994, Revathy Ramadoss

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

THE AUTHOR HAS GRANTED AN
IRREVOCABLE NON-EXCLUSIVE
LICENCE ALLOWING THE NATIONAL
LIBRARY OF CANADA TO
REPRODUCE, LOAN, DISTRIBUTE OR
SELL COPIES OF HIS/HER THESIS BY
ANY MEANS AND IN ANY FORM OR
FORMAT, MAKING THIS THESIS
AVAILABLE TO INTERESTED
PERSONS.

L'AUTEUR A ACCORDE UNE LICENCE
IRREVOCABLE ET NON EXCLUSIVE
PERMETTANT A LA BIBLIOTHEQUE
NATIONALE DU CANADA DE
REPRODUIRE, PRETER, DISTRIBUER
OU VENDRE DES COPIES DE SA
THESE DE QUELQUE MANIERE ET
SOUS QUELQUE FORME QUE CE SOIT
POUR METTRE DES EXEMPLAIRES DE
CETTE THESE A LA DISPOSITION DES
PERSONNE INTERESSEES.

THE AUTHOR RETAINS OWNERSHIP
OF THE COPYRIGHT IN HIS/HER
THESIS. NEITHER THE THESIS NOR
SUBSTANTIAL EXTRACTS FROM IT
MAY BE PRINTED OR OTHERWISE
REPRODUCED WITHOUT HIS/HER
PERMISSION.

L'AUTEUR CONSERVE LA PROPRIETE
DU DROIT D'AUTEUR QUI PROTEGE
SA THESE. NI LA THESE NI DES
EXTRAITS SUBSTANTIELS DE CELLE-
CI NE DOIVENT ETRE IMPRIMES OU
AUTREMENT REPRODUITS SANS SON
AUTORISATION.

ISBN 0-315-98606-9

Canada

Name Interval-Based Performance Analysis of Computing Systems
 Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Performance analysis, Computing Systems
 SUBJECT TERM

0544
 SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
 Art History 0377
 Cinema 0900
 Dance 0378
 Fine Arts 0357
 Information Science 0723
 Journalism 0391
 Library Science 0399
 Mass Communications 0708
 Music 0413
 Speech Communication 0459
 Theater 0465

EDUCATION

General 0515
 Administration 0514
 Adult and Continuing 0516
 Agricultural 0517
 Art 0273
 Bilingual and Multicultural 0282
 Business 0688
 Community College 0275
 Curriculum and Instruction 0727
 Early Childhood 0518
 Elementary 0524
 Finance 0277
 Guidance and Counseling 0519
 Health 0680
 Higher 0745
 History of 0520
 Home Economics 0278
 Industrial 0521
 Language and Literature 0279
 Mathematics 0280
 Music 0522
 Philosophy of 0998
 Physical 0523

Psychology 0525
 Reading 0535
 Religious 0527
 Sciences 0714
 Secondary 0533
 Social Sciences 0534
 Sociology of 0340
 Special 0529
 Teacher Training 0530
 Technology 0710
 Tests and Measurements 0288
 Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
 General 0289
 Ancient 0290
 Linguistics 0291
 Modern 0401
 Literature 0294
 General 0295
 Classical 0297
 Comparative 0298
 Medieval 0316
 Modern 0591
 African 0305
 American 0352
 Asian 0355
 Canadian (English) 0593
 Canadian (French) 0311
 English 0312
 Germanic 0315
 Latin American 0313
 Middle Eastern 0314
 Romance 0370
 Slavic and East European 0372

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
 Religion 0318
 General 0321
 Biblical Studies 0319
 Clergy 0320
 History of 0322
 Philosophy of 0469
 Theology 0323

SOCIAL SCIENCES

American Studies 0324
 Anthropology 0326
 Archaeology 0327
 Cultural 0310
 Physical 0272
 Business Administration 0770
 General 0454
 Accounting 0338
 Banking 0385
 Management 0501
 Marketing 0503
 Canadian Studies 0505
 Economics 0508
 General 0509
 Agricultural 0510
 Commerce-Business 0511
 Finance 0358
 History 0366
 Labor 0351
 Theory 0578
 Folklore 0460
 Geography 0383
 Gerontology 0386
 History 0460
 Toxicology 0383
 Home Economics 0386

Ancient 0579
 Medieval 0581
 Modern 0582
 Black 0328
 African 0331
 Asia, Australia and Oceania 0332
 Canadian 0334
 European 0335
 Latin American 0336
 Middle Eastern 0333
 United States 0337
 History of Science 0585
 Law 0398
 Political Science 0615
 General 0616
 International Law and Relations 0617
 Public Administration 0814
 Recreation 0452
 Social Work 0626
 Sociology 0627
 General 0938
 Criminology and Penology 0631
 Demography 0628
 Ethnic and Racial Studies 0629
 Individual and Family Studies 0630
 Industrial and Labor Relations 0700
 Public and Social Welfare 0344
 Social Structure and Development 0709
 Theory and Methods 0999
 Transportation 0453
 Urban and Regional Planning 0453
 Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
 General 0285
 Agronomy 0475
 Animal Culture and Nutrition 0476
 Animal Pathology 0359
 Food Science and Technology 0478
 Forestry and Wildlife 0479
 Plant Culture 0480
 Plant Pathology 0817
 Plant Physiology 0777
 Range Management 0746
 Wood Technology 0306
 Biology 0287
 General 0308
 Anatomy 0309
 Biostatistics 0379
 Botany 0329
 Cell 0353
 Ecology 0369
 Entomology 0793
 Genetics 0410
 Limnology 0307
 Microbiology 0317
 Molecular 0416
 Neuroscience 0433
 Oceanography 0821
 Physiology 0778
 Radiation 0472
 Veterinary Science 0786
 Zoology 0760
 Biophysics 0425
 General 0996
 Medical 0996

Geodesy 0370
 Geology 0372
 Geophysics 0373
 Hydrology 0388
 Mineralogy 0411
 Paleobotany 0345
 Paleocology 0426
 Paleontology 0418
 Paleozoology 0985
 Palynology 0427
 Physical Geography 0368
 Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
 Health Sciences 0566
 General 0300
 Audiology 0992
 Chemotherapy 0567
 Dentistry 0350
 Education 0769
 Hospital Management 0758
 Human Development 0982
 Immunology 0564
 Medicine and Surgery 0347
 Mental Health 0569
 Nursing 0570
 Nutrition 0380
 Obstetrics and Gynecology 0354
 Occupational Health and Therapy 0381
 Ophthalmology 0571
 Pathology 0419
 Pharmacology 0572
 Pharmacy 0382
 Physical Therapy 0573
 Public Health 0574
 Radiology 0575
 Recreation 0370
 Geology 0372
 Geophysics 0373
 Hydrology 0388
 Mineralogy 0411
 Paleobotany 0345
 Paleocology 0426
 Paleontology 0418
 Paleozoology 0985
 Palynology 0427
 Physical Geography 0368
 Physical Oceanography 0415

Speech Pathology 0460
 Toxicology 0383
 Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences

Chemistry 0485
 General 0749
 Agricultural 0486
 Analytical 0487
 Biochemistry 0488
 Inorganic 0738
 Nuclear 0490
 Organic 0491
 Pharmaceutical 0494
 Physical 0495
 Polymer 0754
 Radiation 0405
 Mathematics 0605
 Physics 0986
 General 0606
 Acoustics 0608
 Astronomy and Astrophysics 0748
 Atmospheric Science 0607
 Atomic 0798
 Electronics and Electrical 0759
 Elementary Particles and High Energy 0609
 Fluid and Plasma 0610
 Molecular 0752
 Nuclear 0756
 Optics 0611
 Radiation 0463
 Solid State 0346
 Statistics 0984

Applied Sciences

Applied Mechanics 0346
 Computer Science 0984

Engineering

General 0537
 Aerospace 0538
 Agricultural 0539
 Automotive 0540
 Biomedical 0541
 Chemical 0542
 Civil 0543
 Electronics and Electrical 0544
 Heat and Thermodynamics 0348
 Hydraulic 0545
 Industrial 0546
 Marine 0547
 Materials Science 0794
 Mechanical 0548
 Metallurgy 0743
 Mining 0551
 Nuclear 0552
 Packaging 0549
 Petroleum 0765
 Sanitary and Municipal 0554
 System Science 0790
 Geotechnology 0428
 Operations Research 0796
 Plastics Technology 0795
 Textile Technology 0994

PSYCHOLOGY

General 0621
 Behavioral 0384
 Clinical 0622
 Developmental 0620
 Experimental 0623
 Industrial 0624
 Personality 0625
 Physiological 0989
 Psychobiology 0349
 Psychometrics 0632
 Social 0451

The undersigned recommend to
the Faculty of Graduate Studies and Research
acceptance of the thesis

“Interval-Based Performance Analysis of Computing Systems”

Submitted by
Revathy Ramadoss, B.E., M.E.

in partial fulfillment of the requirements
for the degree of Master of Engineering

Thesis Supervisor

**Chair, Department of Systems
and Computer Engineering**

**Carleton University,
December 5, 1994**

Abstract

Analytic performance models are often used for predicting the performance of computing systems. Existing models accept single valued parameters as inputs and produce single valued performance measures as outputs. This thesis proposes to associate intervals or ranges of values with performance measures and key system parameters. Conventional arithmetic cannot handle intervals and interval arithmetic-based techniques are required. This thesis reports on the feasibility of application of interval arithmetic in the solution of existing well-known models of computing systems. One of the problems with using interval arithmetic is the potential loosening in the interval for the model output: the computed interval may be wider than the actual interval. Computational methods based on the notion of interval splitting are introduced in this thesis for controlling the looseness in the model outputs. The technique is found to be effective in the context of a number of well-known models such as single and multiclass queueing networks, client-server distributed systems, and more complex models of concurrent systems. Various applications of interval-based performance analysis that include sensitivity analysis of a concurrent application running on a multicomputer and identification of performance bottlenecks are also discussed. A feasibility study of using relational interval arithmetic-based techniques in reverse modeling, wherein a key workload parameter is computed by the model given the performance measures and other workload parameters, is also performed and the results are presented.

Acknowledgments

I would like to express my sincere gratitude to my thesis supervisor Dr. Shikharesh Majumdar for his valuable guidance, advice and encouragement throughout the research.

I would like to thank Alex Hubbard for assisting me in Prolog programming, Erik Dravnieks for explaining some of the concepts of Interval Arithmetic.

My sincere thanks are due to Dr. D.C. Petriu, Dr. C.M. Woodside, Dr. J. Neilson, and Dr. Jerome Rolia for their continued support. The support and assistance from the graduate assistants in the Department of Systems and Computer Engineering are also greatly appreciated.

I would like to thank Carleton University and the Natural Sciences and Engineering Research Council of Canada for their financial assistance.

I would like to thank my parents and brothers for their constant encouragement in my studies. Last but not the least, I would like to sincerely thank my husband Ravi G. Appan for proof reading, for giving critical comments in my thesis, and also for his constant support and encouragement.

To my parents

Table of Contents

Abstract	iii
Acknowledgments	iv
List of Tables	ix
List of Figures	xi
1 Introduction	1
1.1 Performance Analysis of Computing Systems	1
1.1.1 Different Performance Modelling Techniques	2
1.1.2 The Queueing Network Models	4
1.1.3 Markov Chain Models	5
1.2 Motivation for using Interval Arithmetic	6
1.3 Outline of Thesis	10
2 Interval Analysis	11
2.1 Overview	11
2.2 Basics of Interval Arithmetic	11
2.3 Interval Arithmetic in BNR Prolog	14
3 Interval Arithmetic in Performance analysis	21
3.1 Overview	21
3.2 A Problem associated with Interval Arithmetic	21
3.3 The Solution: Interval splitting methods	24

3.3.1	Blind Search Algorithm	27
3.3.2	Extension to Multiple Interval parameters	31
4	Application of Interval Arithmetic to Analytical Modelling	33
4.1	Overview	33
4.2	Interval Arithmetic for Mean Value Analysis.	33
4.2.1	Single Class Queueing Network Model	35
4.2.2	Multi Class Queueing Network Model	40
4.2.3	Approximate Mean Value Analysis	44
4.3	Interval arithmetic for Markov Chain Models	46
4.3.1	Introduction	46
4.3.2	Client-Server Model I	47
4.3.3	Client-Server Model II	50
4.4	Hierarchical Models of Concurrent System	55
4.4.1	Introduction	55
4.4.2	Concurrent Task System Model	56
4.4.3	Sensitivity Analysis for the Concurrent Software System	60
4.5	Larger Models and Scalability of the Algorithm	61
5	Improvements and Other Applications	66
5.1	Chapter Overview	66
5.2	The Selective Search Algorithm	66
5.3	Interval Arithmetic in Bottleneck Analysis	74

5.4 Reverse Modelling	79
6 Conclusions	84
6.1 Research Summary and Contributions	84
6.2 Future Work	86
References	87
Appendix I	
Monotonicity of Throughput with respect to Service time	93
Appendix II	
Multi Class Mean Value Analysis	95
Appendix III	
Schweitzer's Approximate Solution Technique	97
Appendix IV	
Monotonicity of Throughput with respect to Think time	99

List of Tables

Table 3.1	Solutions for the QNM1 Model	23
Table 3.2	Effect of Interval Splitting in QNM1 (ns =15, Tolerance=0.0001)	31
Table 3.3	An example of Multiple interval parameters	32
Table 4.1	Another example of single class MVA (QNM4)	39
Table 4.2	Multi Class Mean Value Analysis	42
Table 4.3	Another example for Multi Class MVA (QNM6)	44
Table 4.4	Approximate Mean Value Analysis for QNM7	45
Table 4.5	Client-Server Model I Results	50
Table 4.6	State Description of Client-Server Model II	54
Table 4.7	Client-Server Model II Results	55
Table 4.8	Service Demands for Processes in Concurrent Application	59
Table 4.9	Effect of changes in Parameters for Concurrent Application	61
Table 5.1	Blind Search Algorithm Results for QNM1	72
Table 5.2	Selective Search Algorithm Results for QNM1	72
Table 5.3	Bottleneck Identification in Multi Class Queueing Networks	79
Table 5.4	Exact Throughputs for given Z	82

Table 5.5	Reverse modelling results	82
Table 5.6	QNM11: Exact throughputs	83

List of Figures

Figure 1.1	System Model	1
Figure 1.2	Example of closed Queueing System	5
Figure 1.3	An example of non-monotonicity	7
Figure 2.1	BNR Prolog Example 1	17
Figure 2.2	BNR Prolog Example 2	18
Figure 2.3	BNR Prolog Example 3	19
Figure 3.1	Single Class Mean Value Analysis	22
Figure 3.2	The Interval Splitting Approach	27
Figure 3.3	The Blind Search Algorithm	29
Figure 3.4	An example of Blind search algorithm	30
Figure 3.5	An example for multiple interval parameters (QNM2)	32
Figure 4.1	Single Class Mean Value Analysis Algorithm	37
Figure 4.2	Single Class Queueing Network Model (QNM3)	38
Figure 4.3	The effect of Blind search Algorithm	38
Figure 4.4	QNM4: Single Class QN	39
Figure 4.5	Multi Class Queueing Network Model (QNM5)	40

Figure 4.6	Non-monotonic behavior of QNM5	41
Figure 4.7	QNM6 - Multi Class QN example	43
Figure 4.8	Client-Server Model I (server with 1 phase)	47
Figure 4.9	Cycle time representation of Client-Server Model I	48
Figure 4.10	Markov Chain Model for Client-Server Model I	48
Figure 4.11	Balance equations for Client-Server Model I	49
Figure 4.12	Client-Server Model II	51
Figure 4.13	Cycle time representation of Client-Server Model II	52
Figure 4.14	Markov Chain Model for Client-Server Model II	53
Figure 4.15	Hardware Platform for Concurrent Tasks	56
Figure 4.16	Precedence graph for the Concurrent Application	57
Figure 4.17	Allocation of Tasks to Computing Systems	57
Figure 4.18	Concurrent Application: Higher level Markov Chain Model	58
Figure 4.19	Variation in CPU time (sec) with Degree of Multiprogramming	63
Figure 4.20	Variation in CPU time (sec) with Number of Devices	64
Figure 4.21	Variation in CPU time (sec) with Number of Subintervals	64
Figure 5.1	An example of Selective search method	67
Figure 5.2	Selective Search Algorithm	70

Figure 5.3	A Comparison of Blind search and Selective search Algorithm	73
Figure 5.4	Comparison of Selective and Blind Search Algorithm	74
Figure 5.5	Queueing Network Model for Bottleneck Analysis	78
Figure 5.6	Reverse Modelling	81
Figure 5.7	An example to illustrate Reverse Modelling	81
Figure 5.8	QNM10: An example for Reverse Modelling	82
Figure 5.9	QNM11 - An example for Reverse Modelling	83

Chapter 1

Introduction

1.1 Performance Analysis of Computing Systems

Computing systems are becoming more and more complex requiring appropriate performance modelling tools for understanding their behavior as well as for the prediction of their performance. Performance analysis is useful during various stages in the lifetime of both centralized and distributed computing systems; during design, development, and implementation, during sizing and procurement, and during system upgrading for matching the demands of an evolving workload. Performance evaluation is also useful to assess if a computer system operates near its optimal capacity under given resource constraints. Therefore performance study is crucial for effective use of the systems.

Figure 1.1 System Model

Any performance model can be represented as a black box with set of inputs and outputs as shown in Figure 1.1. The inputs can be work load parameters such as think time,

demand of each device, number of customers, etc. Outputs can be performance measures, such as, mean response time, throughput, utilization, etc. The specific performance measures of interest are dependent on the type of the system. For example, response time is more crucial than other performance measures in a real time system, whereas throughput is a better description of performance for a batch system.

In different stages of the lifetime of a system one could use different performance evaluation techniques. Measurement is an appropriate approach once a system is built and instrumented. Instrumentation is costly in this method. Modelling may be the only performance evaluation technique that may be used before the construction of the system.

1.1.1 Different Performance Modelling Techniques

A model is an abstract representation of the system. By analyzing and evaluating the model, the behavior of the system can be inferred. An appropriate model should be simpler than the real system and relatively easy to analyze. Therefore, the construction of a model requires a process of considerable abstraction. Some characteristics which have a major effect on the system must be kept in the model. On the other hand, to make the model simpler, some other characteristics may be ignored, if they have less impact on system performance. There is no explicit specification of whether a feature should be included in the model or not. The building of an appropriate model is thus dependent mostly on the ability and experience of the modeler.

The major types of model are analytical models and simulation models. More details of a system can be aptly captured in a simulation model than in analytical models. However, the disadvantage of simulation is the high cost associated with model development and running experiments.

A simulation is the imitation of the operation of a real-world process or system over time. Simulation involves the generation of artificial history of a system, and the observation of that artificial history to draw inferences concerning the operating characteristics of the real system [JJ84]. The simulation models are *run* to get the solution. Simulation models may be classified as static or dynamic, deterministic or stochastic and discrete or continuous. Simulation can also be classified as event driven, trace driven and time driven.

This thesis is concerned with analytical models. The characteristics of the real system are represented as mathematical equations and inequalities in analytic models. Analytical models are *solved* by mathematical methods. The merits and demerits of using analytical models are briefly described.

- Analytical models are of lower cost compared to simulation modeling.
- Analytical models are some time used to gain insight in to the performance characteristics of the system under study.
- Analytical models are not only useful in evaluating existing system, but also can predict the performance of a future system.
- Analytical models have lower accuracy than simulation models due to restrictions imposed by the assumptions required for ensuring the feasibility and simplicity of analytical models.
- Size of the model has a strong impact on the time and space complexity associated with the analytical solutions.

The effectiveness of analytical modelling can be improved by integrating interval arithmetic with the analytical methods. Before describing interval-based analytical modelling, a brief overview of some known analytical models is given in the following sub-sections.

1.1.2 The Queueing Network Models

Queueing Network (QN) models have become important tools in the design and analysis of computer systems because they achieve a favorable balance between accuracy and efficiency [Kle76]. In such models, a computer system is represented as a network of queues which is evaluated analytically. A network of queues is a collection of service centers, which represents system hardware resources and their queues, as well as customers, which represent jobs or transactions. Customers arrive at the service center, wait in the queue if necessary, receive service from the server and depart. Customers can be divided into different classes. Customers with statistically identical characteristics, are grouped into a class. Each class is characterized by a number of parameters such as the workload intensity for the class and the service demand at each service center. The only dependence among the customers is that they compete for resources. Customers do not synchronize or communicate with one another.

If the population of customers in a given class of the QN system is fixed at all times then the QN is said to be closed. Here, a customer that has completed service can be thought of as leaving the system but replaced instantaneously by a statistically identical job. On the other hand if the customer population varies over time then that QN is said to be open. Here, customers arrive from outside, complete service and leave the system. Figure 1.2 is an example of a closed queueing system in which the Central Processing Unit (CPU) and Input/Output (I/O) devices are modelled by different kinds of service centers.

Queueing network models are easy to build and require a few parameters. Numerous solution methods have been developed to solve queueing models. Different solution techniques with the emphasis on practical considerations can be found in [LZGS84].

Figure 1.2 Example of closed Queueing System

QN model is a powerful analytical model to evaluate system performance, but has a number of limitations [Jai91]. It fails to capture the impact of system concurrency as well as the communication and synchronization among software processes in jobs. This is because in a standard QN, each customer is served by one server at a time, and simultaneous resource possession is not allowed. The multi-level service requests produced by a server which calls another server while serving a client cannot be modelled directly by QN.

1.1.3 Markov Chain Models

Concurrency in execution and simultaneous resource possession can be modelled with the help of Markov Chains [Pap91]. In a Markov Chain model we may visualize the system as a process moving from one state to another state. Each state represents a combination of processes in execution on the system. Each feasible execution state appears once at each level. To determine the probability of a particular state being the next state, we would need to know all of the previous states - for small models it is easy to keep track of all the history but for larger models it is complex. The rate at which a process moves from one state to another is known as transition rate.

Although Markov Chain models are able to capture the impact of system concurrency and the communication and synchronization among software processes they suffer from an exponential growth of the state space. This severely restricts the size of tractable models.

1.2 Motivation for using Interval Arithmetic

Uncertainties are sometimes associated with values for system parameters during various stages of system design. For example, at the early stages of design, the designer may not be aware of how much CPU time each task will require. However a range of values for CPU time may be known from previous experience. Associating an interval with such system parameters that form input parameters for models is appropriate in such a situation. Intervals or ranges may be associated with model outputs as well. The output of a simulation model is often expressed in terms of confidence intervals. Whenever possible, approximate solutions of analytical models are stated in conjunction with bounds on the error associated with the approximation.

Currently all the existing performance models accept a set of *single* valued parameters. In most real applications, input data is accurate to within a few percent only. When several such numbers are used as input data for a complex arithmetic formula, it can be quite difficult to estimate the accuracy of the result. One way of doing this is to perform a sensitivity analysis, varying every input parameter separately and observing the effect produced, which is very time consuming and involves high computational costs. Interval arithmetic provides a natural solution to this kind of problem by simply representing the input data as an interval.

Complex situations arise when the model is non-monotonic with respect to any of its input parameters. It is not guaranteed that by running the algorithm for various single point values, one might accurately estimate the minimum or maximum. Interval arithmetic

technique that can accept interval values for input parameters produces a result that is guaranteed to contain the actual interval for the performance metric.

Non-monotonic behavior has been observed in several different situations. In the classical studies of system thrashing [SPG92] it is shown that the throughput of a computer system increases initially with an increase in the level of multiprogramming, but soon reaches a plateau and starts decreasing with further increase in the multiprogramming level (similar to Figure 1.3). At very high levels of multiprogramming an application receives only a few pages of memory that results in a heavy page faulting activity known as thrashing.

Figure 1.3 An example of non-monotonicity

A recent study of client server distributed systems shows a non-monotonic relationship between system throughput and the duration of the time-out associated with the receiving of a response for Remote Procedure Calls (RPC) [RSB93]. RPC's are the mechanisms used by the client nodes for obtaining the desired service from servers. If the time-out

duration is small in comparison to the normal delay associated with request response cycle over the network, a large number of retries are unnecessarily generated. An increase in the time-out duration reduces the number of retries and system performance improves. However for very large values of the time-out duration a great deal of time is spent waiting when the message is lost and a retry is required. This tends to decrease the system throughput (see [RSB93] for a detailed discussion of this issue). In many situations monotonicity properties of system may be hard to determine. In all such situations an interval based performance analysis is warranted.

The main objective of this thesis is to investigate the feasibility of performing interval-based performance analysis that is concerned with computing upper and lower bounds on performance that corresponds to interval parameters. Similar motivations exist in other areas. For example the *optimization problem*, in general, is to find the optimum (maximum or minimum) value of a function in a given domain and to find the values of the variables where the optimum is reached in this domain. *Global optimization* means solving the optimization problem in an area given by a real-world problem [RR88]. During the past decades, the role of optimization has steadily increased in such diverse areas as, for example, electrical engineering, operational research, computer science and communication. *Linear and non-linear optimization* [Wag75], the search for an optimum of a function of *continuous* variables, has seen major breakthroughs in the fifties and sixties. The best known example is the simplex algorithm to solve linear programming problems, introduced by Dantzing [Dan63]. Major results in combinatorial optimization [PS82], the search for optima of functions of *discrete* variables, were obtained predominantly in the seventies.

Simulated annealing is a general approach for approximately solving the large combinatorial optimization problems [VA87]. The algorithm is based on an intriguing combination

of ideas from at first sight completely unrelated fields of science, viz. combinatorial optimization and statistical physics. On the one hand the algorithm can be viewed as an analogue of an algorithm used in statistical physics for computer simulation of the annealing of a solid to its minimum-energy state; on the other hand it can be considered as a generalization of the well known iterative improvement approach to combinatorial optimization problem.

Most optimization techniques can be used to find only approximate values of maxima or minima. The interval arithmetic based technique investigated in this thesis produces intervals that are guaranteed to contain the upper and lower bounds on performance. The research is motivated by a number of high level questions that include the following.

- How effective is the interval arithmetic in performance analysis?.
- Interval arithmetic is known to introduce some looseness in the computed result; that is, the actual interval for a performance measure of interest may be narrower than that computed by an interval arithmetic technique. Techniques for effectively controlling this looseness need investigation.
- Is the relational interval arithmetic provided in the BNR Prolog language appropriate for performing interval-based performance analysis?.

The work presented in the subsequent chapters of this thesis demonstrates the effectiveness of interval arithmetic in performance analysis by incorporating interval arithmetic based techniques in a number of popular analytic models. Techniques for controlling the looseness in computed results have been developed in this thesis. The effectiveness of BNR Prolog in solving small to medium scale problems is also demonstrated. Some of the results presented in the thesis constitute a paper [MR95] that will be published in the Proceedings of the International Workshop on Modeling, Analysis, and Simulation of Com-

puter and Telecommunication Systems. (MASCOTS'95), to be held in Durham, North-Carolina, January 18-20, 1995.

1.3 Outline of Thesis

This thesis is divided into six chapters. A brief introduction to Interval arithmetic and BNR Prolog is provided in Chapter 2. Previous application of interval arithmetic to performance analysis is also discussed at the end of the Chapter 2. During early stages of this research a problem in using interval arithmetic was detected (similar problems of using interval arithmetic have been detected in other domains such as optimization). The problems associated with using interval arithmetic and the solution methods are discussed in detail in Chapter 3. In Chapter 4, application of interval arithmetic to various analytical performance models are considered. The models studied include Queueing Network models that have been used extensively in various studies of computing system performance [LZGS84], Markov Chain models, and models of complex concurrent task systems which use two-level hierarchical decomposition techniques [TB86]. Bottleneck analysis and Reverse modelling are discussed in Chapter 5. Chapter 6 provides a summary of this research, the conclusions derived from this work, as well as a short discussion of directions for further research.

Chapter 2

Interval Analysis

2.1 Overview

This chapter mainly focuses on introductions to interval arithmetic and the programming language, BNR Prolog, used in this thesis. The basics of interval arithmetic is discussed in the Section 2.2, where various terminology used in interval arithmetic are defined. The basics of interval arithmetic were gathered from the references [AH83], [Han92], [Moo66], [Neu90], [Rat84], and [RR88]. In Section 2.3, the interval arithmetic support provided in BNR Prolog is discussed. A brief review of the previous works that used interval arithmetic in performance analysis is provided towards the end of this chapter.

2.2 Basics of Interval Arithmetic

Interval analysis was introduced by Moore [Moo66] in 1966. He is considered as the “father” of interval arithmetic. The main objective was to eliminate (or at least to account for) the errors generated in solving models. Although a number of standard optimization techniques for the minimization (maximization) of functions can be used to determine the upper and lower bounds on the result, the results produced by these techniques are not guaranteed to contain the actual result because of the truncation error, round off errors that might occur depending on the precision of the computer used to solve the given model. However, the interval arithmetic technique guarantees the result.

An *interval* defines a continuous range of real numbers lying between a lower and upper bound. It is represented as

$$X = [a, b], \{x: a \leq x \leq b\}$$

where a is known as *lower bound* of X and b is known as *upper bound* of X . An interval Y can also be represented as $[Y, Y^+]$ where Y is known as *lower bound* of Y and Y^+ is known as *upper bound* of Y .

A *point interval* is an interval with $a = b$, and it is represented by only one element. The *width* of an interval X is denoted by $w(X) = b - a$.

Interval arithmetic refers to the arithmetic operations involving interval numbers. The basic operations in interval arithmetic are provided here.

Addition: $[a, b] + [c, d] = [a + c, b + d]$

Subtraction: $[a, b] - [c, d] = [a - d, b - c]$

Multiplication: $[a, b] * [c, d] = [\min(ac, ad, bc, bd), \max(ac, ad, bc, bd)]$

Division: $[a, b] / [c, d] = [a, b] * [1/d, 1/c]$ if $0 \notin [c, d]$

Interval arithmetic and real arithmetic show similarity in the following features:

- Both are commutative and associative for addition and multiplication
- Division by zero is undefined and therefore not allowed.

Interval arithmetic differs from real arithmetic in the following features:

- Subtraction (division) is not the inverse operation of addition(multiplication) for interval arithmetic.

- While real arithmetic is distributive, interval arithmetic is only sub-distributive [RR88], i.e., for any three intervals I , J , and K , only the following relationship holds:

$$I \times (J + K) \subseteq I \times J + I \times K$$

In an interval if the lower bound and/or the upper bound is infinite then that interval is known as an *unbounded interval*. Examples of unbounded intervals are $R = [-\infty, 8]$, $S = [-5.2, \infty]$ and $T = [-\infty, \infty]$.

In the above rules of interval arithmetic, we excluded division by an interval containing zero, and the unbounded intervals. But it is often useful to remove these restrictions. The resulting arithmetic is called *extended interval arithmetic*. More details of extended interval arithmetic can be found in [Han92]. For the purpose of this thesis there is no need to know about extended interval arithmetic. Division by a point interval equal to zero is still undefined.

Suppose we subtract the interval $X = [a, b]$ from itself, we obtain $[a - b, b - a]$ as a result. We might expect to obtain $[0, 0]$. However, we do not (unless $b = a$). In general, when a given variable occurs more than once in an interval computation, it is treated as a different variable in each occurrence [Han92]. Thus $X - X$ is the same as $X - Y$ with Y having the same value but independent of X . This gives rise to a widening of the computed intervals and makes it more difficult to obtain sharp results in calculations. This problem is known as *dependency problem*. One should always be aware of this consideration and take appropriate steps to reduce this effect.

To reduce the effect of the dependency problem when computing with interval arithmetic, a generalized interval arithmetic has been developed [Han92]. BNR prolog uses this generalized interval arithmetic.

To maintain correctness of the result, outward rounding is done in languages that support interval arithmetic. Present day computers use floating point arithmetic, in that real numbers are approximated by a subset of the real numbers called machine representable numbers. A machine interval is an interval which has lower and upper bounds that are machine numbers. For example, the real number $1/3$ cannot be exactly represented by a machine number. It may however be enclosed in the machine representable interval $A = [0.33, 0.34]$ if we assume that the machine numbers are representable by two digit numbers. The bounds of a number are floating point numbers that define the range which may vary over lifetime of a computation. Various programming languages such as FORTRAN, Pascal, C, C++ and Prolog support interval arithmetic-based computations. Interval arithmetic is used very extensively in global optimization problems in order to guarantee that the global optimum point has been approximated within a given tolerance [RR88].

When a floating point number or an integer occurs in an expression containing an interval, that floating point number is coerced to a small interval containing the integer or floating point value. Since point intervals are usually intended to represent real numbers that may not be exactly representable as a floating point number, the region they span is of nonzero length [RR88].

2.3 Interval Arithmetic in BNR Prolog

BNR Prolog is one of the programming languages that support interval arithmetic and was developed at Computing Research Laboratory of Bell-Northern Research. Relational form of interval arithmetic was integrated into Prolog to form a general constraint Logic Programming Language. This idea is qualitatively different from other forms of automated interval arithmetic because the combination of a symbolic logic programming language and a mathematically correct arithmetic constraint system is considerably more powerful

than either taken separately. More details about BNR Prolog can be found in [BNRU88] and [BNRR88]. Discussions on programming in Prolog can be found in [CM87].

BNR Prolog introduces a new data type for numbers that is distinct from floating point or integer - the type *interval*. The bounds of an interval are floating point numbers that define its range. The range of an interval may vary over the life of computation. Interval in BNR Prolog can be integer-bounded or real-valued. There are three important characteristics of intervals provided in BNR Prolog that make them interesting and useful.

- Intervals allow the expression of arithmetic relationships in a fully logical and relational manner
- Intervals provide a means for proving statements about the true real numbers without being restricted to the limitations of a particular floating point representation.
- Intervals provide a mechanism for solving set of linear and nonlinear equations and inequalities.

When a floating point number or an integer occurs in an expression containing an interval, that floating point number is coerced to a small interval containing the integer or floating point value. Since point intervals are usually intended to represent real numbers that may not be exactly representable as a floating point number, the region they span is of nonzero length.

Expressions relating input parameters of analytical models to model outputs can be represented as a set of equalities and inequalities. Each such equality (inequality) can be expressed as a Prolog subgoal. The BNR Prolog interpreter evaluates the set of equalities and inequalities and computes the range associated with each interval in the program that satisfies the arithmetic relationship specified in each subgoal. As each equality (inequality)

is evaluated, additional constraints are generated that can narrow the width of the interval. Narrowing of an interval is caused by the raising of its lower bound and/or the lowering of its upper bound. Interval values narrow their widths as the computation moves forward and returns to a previous value by back-tracking. Back-tracking may undo the narrowing of an interval in a way that is analogous to the unbinding of ordinary variables.

The interval arithmetic feature of BNR Prolog is explained with the help of following examples. In the BNR Prolog program shown in Figure 2.1 the variables X and Y are of type interval whose interval ranges are $[1,5]$ and $[3,6]$ respectively. The relation $X \geq Y$ in this program narrows the ranges of both the interval variables X and Y , i.e., the lower bound on X is raised and the upper bound on Y is lowered (see Figure 2.1). The predicate `nl` is to print a blank line. Then the variables are printed using the `print_interval` predicate. In the BNR Prolog program shown in Figure 2.2, the intervals for variables X and Y are non-overlapping, and the relation $X \leq Y$ can never be true. This leads to the failure of the predicate (see Figure 2.2) and the result 'No'. The BNR Prolog program shown in Figure 2.3, is to show how unbounded intervals (infinite in lower bound and/or upper bound) are represented in this language. Variable X is an interval ranging from $-\infty$ to 6 and the interval variable Y ranges from 5 to $+\infty$. The relation $X \geq Y$ narrows the range of interval variables X and Y .

PROGRAM:

```
% Set the initial ranges
X: real(1.0, 5.0),
Y: real(3.0, 6.0),
X >= Y,
% Print Results
nl, print_interval(X),
nl, print_interval(Y).
```

RESULTS:

```
[3.0, 5.0]
[3.0, 5.0]
YES
```

Figure 2.1 BNR Prolog Example 1

PROGRAM:

```
% Set the initial ranges
X: real(5.0, 6.0),
Y: real(2.0, 4.0),
X =< Y,
% Print Results
nl, print_interval(X),
nl, print_interval(Y).
```

RESULTS:

NO

Figure 2.2 BNR Prolog Example 2

PROGRAM:

```
% Set the initial ranges
X: real(_, 6.0),
Y: real(5.0, _),
X >= Y,
% Print Results
nl, print_interval(X),
nl, print_interval(Y).
```

RESULTS:

```
[5.0, 6.0]
[5.0, 6.0]
YES
```

*Intervals
(before applying
inequality)*

*Intervals
(after applying
inequality)*

Figure 2.3 BNR Prolog Example 3

Interval arithmetic in BNR Prolog has been used in different areas such as Critical Path (PERT) analysis [OVF88], [OV89] and X-Ray Diffraction Crystallography[Old88]. Previous applications of interval arithmetic to performance analysis include the computation of robust throughput bounds for general Multiclass Queueing Networks [MWNP92], [MW94]. [MWNP92] presents a technique for the computation of upper and lower bounds on throughput for different customer classes in a queueing network with servers employing a FIFO queueing scheme. Bounds for various other methods of scheduling (such as Processor Sharing, Preemptive Priority, Non Preemptive Priority) at the server are discussed in [MW94]. Interval Arithmetic has also been used for performance analysis of Client-Server software system through bounds on task throughputs [WMN91]. Interval arithmetic based performance analysis of distributed computing system has been studied in [Maj91]. Interval arithmetic has been used for the computation of bounds on the performance of concurrent software using rendezvous style communication [MWNP91]. In [Lin94], BNR Prolog has been used in the computation of the bounds on the performance of certain classes of Client-Server systems in which clients are multi-threaded.

BNR Prolog can be used to write programs for the solution of analytical models but it may not always achieve the desired tightness in the interval for model output. This problem and its solutions are the major focus of attention for this thesis and are discussed in the subsequent chapters.

Chapter 3

Interval Arithmetic in Performance analysis

3.1 Overview

A number of motivations for interval-based performance analysis were presented in Chapter 1. When using interval arithmetic in any exact solution technique for an analytical performance model it was observed that if the inputs were not of point-intervals then the result obtained may turn out to be a very wide interval (containing the correct answer). The actual width of the resultant interval would be much smaller. A more detailed description of this problem is provided in Section 3.2. A method for solving this problem is discussed in Section 3.3.

3.2 A Problem associated with Interval Arithmetic

The effect of an interval input parameter on the precision of an output interval is illustrated clearly in the following simple example. Consider a BNR prolog program implementing the exact single class Mean Value Analysis (MVA) for a simple closed Queueing Network (QNM1) with a single queueing centre as presented in Figure 3.1. All the parameters of the model except the service demand at the queueing centre are single point values. The output of the MVA algorithm is the system throughput X specified in jobs per sec. The input parameters for QNM1 are as follows: Think time (Z) = 4 sec, Number of customers (N) = 3. Service demand at queueing centre (D) = $[3,4]$ sec

Figure 3.1 Single Class Mean Value Analysis

```

MVA1():-
    % Define number of customers
    N is 3.0,
    % Define initial Queue value
    Q is 0,
    % Input parameters for model (service demand is an interval)
    D:real(3,4), Z is 4,
    % Call predicate throughput that actually performs computation
    M is N + 1,
    throughput(D, Z, Q, M, 1).

% Return to the original predicate MVA1:
throughput(D, Z, Q, M, M): - !.

% Levels of recursions performed = no. of customers + 1
throughput(D, Z, Q, M, Level):-
    % MVA iteration specified by Level
    % Declare variables (_ signifies highest positive number (le 38))
    X:real(0,_),
    R:real(0,_),
    New_Q:real(0,_),
    % MVA equations
    R == D * (1 + Q),
    X == Level / (Z + R),
    New_Q == X * R,
    New_Level == Level + 1,
    % Print results for number of customers = level
    write(Level),nl,
    print_interval(X),nl,
    % start next level of recursion
    throughput(D, Z, New_Q, M, New_Level).
    
```

MVA1() is the main predicate of the MVA where all the inputs are specified. The actual throughput is calculated via the predicate *throughput()*. The predicate *throughput()* is called recursively N times. Detailed description of the MVA algorithm is given in the next chapter. Each equation in the exact MVA solution is represented by a clause or subgoal in the BNR Prolog program as shown in Figure 3.1.

The interval for system throughput X ($[X^-, X^+]$) may not always be as tight as possible. Similar observations have been made by other researchers in the context of a different problem domain [Ske74]. The rationale behind this looseness in the computed interval is explained with the help of the results presented in Table 3.1. The system throughput computed by the BNR Prolog program for a given number of customers ($N=1,2,3$) are given in the third column of this table. It is verified that the throughput for QNM1 decreases monotonically with an increase in D . The verification is presented in Appendix I. Thus it is possible to compute the exact interval for the throughput X obtained with the given interval for D ; X^- is computed by running a conventional MVA algorithm at D^+ and X^+ is computed by running the same algorithm at D^- . The actual values of X are presented in the second column of the table. As shown by the table, an interval X computed by the BNR Prolog program always contains the actual interval for X but the deviation in the computed interval of $[X^-, X^+]$ from its actual value or the looseness in the computed interval for X increases as N increases.

Table 3.1 Solutions for the QNM1 Model

N	Actual X	Computed X	% Deviation (in X^-)	% Deviation (in X^+)
1	[0.1250, 0.1429]	[0.1250, 0.1429]	0	0
2	[0.2000, 0.2414]	[0.1944, 0.2462]	2.80	0.08
3	[0.2344, 0.2969]	[0.2114, 0.3189]	9.80	7.41

Consider the program shown in Figure 3.1. In the first iteration ($N=1$) $Q_1 = 0$ that leads to $R_1 = [D_1^-, D_1^+]$. Thus $X^+ = 1 / (D_1^- + Z)$ and $X^- = 1 / (D_1^+ + Z)$ and the computed interval for X overlaps exactly with the actual interval for X . The value for Q_1 used at the beginning of the second level of iteration ($N=2$) is obtained by applying the multiplication rule given in Chapter 2.

$$Q_1 = XR_1 = [(D_1^- / (D_1^+ + Z)), (D_1^+ / (D_1^- + Z))]$$

Note that since X and R_1 are treated independently and Q_1^- is obtained by multiplying R_1^- with X^- and Q_1^+ by multiplying R_1^+ with X^+ and the fact X^- corresponds to R_1^+ whereas X^+ to R_1^- is lost. As a result although the computed interval for X contains the actual interval for X it is looser than the actual interval. As the number of iterations increases with N , the number of interval operations performed also increases. This tends to loosen up the interval and the deviation from the actual interval for X increases. A solution for this problem is presented in the following sections.

3.3 The Solution: Interval splitting methods

The looseness in model output captured in the previous section depends on the width of the interval parameter as well as the number of occurrences of interval parameter. It has been proved (in [Ske74]) that precision of the computed result depends on width of the input intervals. The problem arises due to the independent treatment of variables in each recursion. The more the number of occurrences of an interval variable, the less is the output precision. Considering the previous example of QNM1, when the number of customer was 1 the interval input parameters occurred only twice and the computed interval for X overlapped exactly with the actual interval for X . When the number of customers was increased, the number of recursion levels increased and the looseness in the precision of the output increased. However, even if number of recursion levels is large but the width of

the input interval parameter is small, one can obtain output interval parameters with good precision.

Let $f(x_1, x_2, \dots, x_n)$ be a rational function of the real variables $x_1, x_2, \dots, x_n$. The expression f contains a finite number of arithmetic operations. A rational interval function $F(X_1, X_2, \dots, X_n)$ is derived from f by replacing the arithmetic operations by the equivalent interval operations; f is called the real restriction of F . f can be rewritten as:

$$f(x_1, x_2, \dots, x_n) = f(c_1, c_2, \dots, c_n) + g(x_1 - c_1, x_2 - c_2, \dots, x_n - c_n),$$

where c_i is the center of the interval X_i .

If an interval function $F(X_1, X_2, \dots, X_n)$ is derived from the right-hand side of the above equation, then the interval function is said to be in *centered form*. The theorem is stated as follows:

Theorem 1: Let $F(X_1, X_2, \dots, X_p, X_{p+1}, X_{p+2}, \dots, X_n)$ be a rational interval function written in centered form. Each of the variables $X_{p+1}, X_{p+2}, \dots, X_n$ occurs just once in the function F . Let the width of the interval X_i be $w(X_i)$

Subdivide each of the interval $X_1, X_2, \dots, X_p$ into N divisions so that

$$X_i = \bigcup_{j=1}^N X_i^{(j)} \text{ with } w(X_i^{(j)}) = \frac{1}{N} w(X_i)$$

There is a positive number k such that

$$\bigcup_{i_1=1}^N \cdot \cdot \cdot \bigcup_{i_p=1}^N F(X_1^{(i_1)}, \dots, X_p^{(i_p)}, X_{p+1}, \dots, X_n) = \bar{F}(X_1, X_2, \dots, X_n) + E_N$$

where

$$\bar{F}(X_1, X_2, \dots, X_n) = \{f(x_1, x_2, \dots, x_n) \mid x_i \in X_i, i=1,2,\dots,n\},$$

$$\bar{F} \text{ and } F \text{ satisfy the equation: } \bar{F}(X_1, X_2, \dots, X_n) \subseteq F(X_1, X_2, \dots, X_n),$$

$$w(E_N) \leq \frac{k}{N^2} \max_i (W(X_i)^2) \quad i = 1, 2, \dots, n \text{ and } 0 \in E_N.$$

The proof is given in [Ske74]. The above expression for $w(E_N)$ implies that as the number of subintervals increases, the tightness of the output can be improved. This observation forms the basis of interval splitting techniques discussed. It is clear from the above discussion that the interval for the model output can be tightened by subdividing the given interval for an input parameter into a number of subintervals and then running the algorithm in each subinterval. The general notion of interval subdivision has also been used in domains such as optimization [Ske74]. An effective algorithm that uses interval splitting and is appropriate for implementing solutions to analytic performance models is presented in this section.

The basic approach taken by the algorithm is explained with the help of Figure 3.2, in which only a single interval parameter is considered. This approach can be easily extended to handle multiple interval parameters; such an extension is discussed at the end of this chapter. As shown in Figure 3.2, the given interval I is split into n subintervals of equal width $I_1, I_2, \dots, I_n$. The algorithm for solving the analytical model is applied for each subinterval I_i . The model output, P_i , for each subinterval I_i is collected and the output of the algorithm is given by $P = [P^+, P^-]$ where:

$$P^- = \min(P_1^-, P_2^-, \dots, P_n^-) \text{ and } P^+ = \max(P_1^+, P_2^+, \dots, P_n^+).$$

Figure 3.2 The Interval Splitting Approach

3.3.1 Blind Search Algorithm

From theorem 1, we know that the model output can be computed to any desired precision by using an infinite number of subintervals. Computational time, however, is directly proportional to the number of sub-intervals. Thus there exists a trade-off between the output precision and the computational time.

At first, in this algorithm one has to specify the input interval parameter for the model and the other single valued input parameters. Initial number of sub-divisions for the interval (ns), maximum number of iterations (i_{max}) and the tolerance value are also to be specified by the user. The input interval is split into ns sub divisions of equal width. The computing system model is solved for each sub-interval. The overall output of the system is computed from the outputs of each of these solutions. The number of subintervals ns is increased by a fixed value in the next iteration. The output obtained at the end of this iteration is compared with that obtained from the previous iteration. If the difference between two outputs are within the given tolerance value 'threshold' the algorithm stops (see

Figure 3.4). Otherwise the number of subintervals is increased and another iteration starts. Until the maximum number of iteration is reached or the tolerance value is reached between two successive iterations, the number of sub-division is increased in each iteration and the procedure is repeated. In this algorithm sub-divisions can be increased in any fashion.

The detailed algorithm is provided in Figure 3.3. This algorithm is further explained here with Figure 3.4, where the number of subdivision is 2 in the first iteration, and the subdivisions are increased to 2 times the previous number of subdivisions. Therefore the second iteration has 4 subdivisions and the 3rd iteration has 8 subdivisions and so on until the maximum number of iterations are reached or desired output precision is reached. In Figure 3.4, $X_0^{(0)}$ implies the initial interval. Subdivided intervals are represented as $X_{\text{iteration number}}^{(\text{subdivision number})}$, i.e., $X_1^{(2)}$ implies the second subinterval of first iteration and $X_3^{(5)}$ implies the fifth subinterval of the third iteration.

BEGIN

Inputs: Interval parameter, Single valued parameters, Initial number of sub-intervals(ns), maximum number of iterations(imax) and the tolerance value(threshold).

Initial Values: $i = 0$, $Xmin_previous = 0$, $Xmax_previous = 0$

WHILE $i < imax$ **DO**

$i = i + 1$

FOR $j = 1$ to ns **DO**

Compute throughput [Ymin, Ymax] for j-th sub-interval using MVA,

$Xmin_list[j] = Ymin,$

$Xmax_list[j] = Ymax,$

END FOR;

$Xmin_current = \min (Xmin_list)$

$Xmax_current = \max (Xmax_list)$

IF $((|Xmin_current - Xmin_previous| < threshold) \text{ AND}$

$(|Xmax_current - Xmax_previous| < threshold))$

THEN

break;

ELSE

Increase the no. of sub-intervals

$(ns = ns * c)$

$Xmin_previous = Xmin_current,$

$Xmax_previous = Xmax_current,$

END IF;

END WHILE;

$X = [Xmin_current, Xmax_current]$

END

Figure 3.3 The Blind Search Algorithm

Figure 3.4 An example of Blind search algorithm

The improvement in the computed throughput for QNM1 for $N = 2, 3$ obtained with interval splitting is presented in Table 3.2. For $N=1$, interval splitting is not needed because there is no deviation of computed X using interval arithmetic from the actual X . The factors for controlling the execution of the algorithm are included in the header for the table. Note that the system throughput X is a monotonic function of D ; such an example was chosen because it is possible to easily compute the actual interval for X by running a conventional MVA Algorithm for D^- and D^+ . Interval splitting is observed to have a significant effect. The intervals obtained as an output of splitting are much narrower than interval for X obtained without splitting.

Table 3.2 Effect of Interval Splitting in QNM1 (ns =15, Tolerance=0.0001)

N	Throughput (X) (actual)	Throughput (X) (without interval splitting)	Throughput(X) (with interval splitting)
2	[0.2000, 0.2414]	[0.1944, 0.2462]	[0.1996, 0.2418]
3	[0.2344, 0.2969]	[0.2114, 0.3189]	[0.2335, 0.2979]

Introducing limits on the upper and lower bounds on X along with other MVA equations, i.e., adding extra inequalities that correspond to the asymptotic upper and lower bounds for single class queueing network in the program aids in controlling the width of these intervals. The computation time required by the blind search algorithm to converge depends on the nature of the model being solved. A detailed discussion of this issue is presented in Chapter 4.

Note: Although the algorithm demonstrates the use of interval splitting in the context of single class Mean Value Analysis, the technique can be easily adapted to handle algorithms for the solution of other analytic models.

3.3.2 Extension to Multiple Interval parameters

In the above algorithm only one interval parameter was taken into consideration. This algorithm can be easily extended to handle multiple interval parameters. A simple extension that has produced satisfactory results is described here. Let $I_1, I_2, \dots, I_i, \dots, I_n$ be the interval parameters for the model. Each interval parameter I_i is split into m subintervals as $I_{i1}, I_{i2}, \dots, I_{im}$. The throughput is computed for all possible subinterval parameter combinations. If $[P_j^-, P_j^+]$ is the model output in the j -th combination of input subintervals, then the output of the algorithm $[P^-, P^+]$ is computed as:

$$P^- = \min(P_1^-, P_2^-, \dots, P_k^-) \text{ and } P^+ = \max(P_1^+, P_2^+, \dots, P_k^+), \text{ where } k=m^n.$$

The rationale for this approach is based on the property of interval arithmetic as given in theorem 1 as discussed in [Ske74].

The algorithm for multiple interval parameters is tested in the following single class QN model. The Queueing network (QNM2) which is considered is shown in Figure 3.5. QNM2 has three devices, a delay center and two interval parameters.

Figure 3.5 An example for multiple interval parameters (QNM2)

Input parameters for QNM2: $N = 5$, Think time = 4, $D_{dev1} = [3, 4]$, $D_{dev2} = [1, 1.5]$, $D_{dev3} = 2$

Both the intervals (D_{dev1} and D_{dev2}) are split simultaneously. The results obtained are presented in Table 3.3. Interval Splitting is found to be effective in this case too.

Table 3.3 An example of Multiple interval parameters

Description	Computed Throughput
without IS	[0.1778, 0.3333]
No of split=4	[0.2045, 0.3191]
No of split=9	[0.2134, 0.3095]
No of split=16	[0.2178, 0.3047]
No of split=25	[0.2205, 0.3017]
No of split=100	[0.2258, 0.2958]

Chapter 4

Application of Interval Arithmetic to Analytical Modelling

4.1 Overview

The interval based performance analysis using the interval splitting technique is applied to a number of different analytical models. Our experience with a representative set of cases that highlight different aspects of the interval based performance analysis is described here. The first application is concerned with queueing network models of computing systems and is discussed in Section 4.1. The second application is concerned with Markov Chain models of computing system and is described in Section 4.3. The third application is concerned with the performance analysis of concurrent software requiring a hierarchical decomposition method. At the higher level, the system behavior is specified by a Markov Chain. At the lower level, the transition rates among the states of the Markov Chain are computed using a QN model. This third application is discussed in Section 4.4.

4.2 Interval Arithmetic for Mean Value Analysis.

Queueing network modelling is a methodology for the analysis of computer system. The basics of QN model is explained in Section 1.1.2. A subset of general class of queueing network models is known as *separable* queueing network models (or product form QN models). The name *separable* comes from the fact that each service center can be sepa-

rated from the rest of the network, and examined in isolation[DB78]. The solution of the entire network then can be formed by combining these separate solutions. The QNs considered in this thesis are separable form, which obey the following conditions [Jai91]:

1. Job flow balance: for each class, the number of arrivals to a device must be equal to the number of departures from the device.

2. One step behavior: A state change can result only from single jobs entering the system, moving between pairs of devices in the system, or exiting from the system. This assumption asserts that simultaneous job moves will not be observed.

3. Device Homogeneity: A device's service rate for a particular class does not depend on the state of the system in any way except for the total device queue length and the designated class's queue length. This assumption implies the following:

- **Single-Resource Possession:** A job may not be present (waiting for service or receiving service) at two or more devices at the same time.
- **No Blocking:** A device renders service whenever jobs are present; its ability to render service is not controlled by any other device.
- **Independent Job Behavior:** Interaction among jobs is limited to queueing for physical devices; for example, there are no synchronization requirements.
- **Local Information:** A device's service rate depends only on local queue length and not on the state of the rest of the system.
- **Routing Homogeneity:** The job routing should be state independent. This implies that the probability of a job going from one device to another device does not depend upon the number of jobs at various devices.

- **Fair Service:** If service rates differ by class, the service rate for a class depends only on the queue length of that class at the device and not on the queue lengths of other classes. This means that the servers do not discriminate against jobs within a class depending on the queue lengths of other classes.

QNM Inputs and Outputs: The basic entities in queueing network models are service centers and customers. The input parameters are number of service centers, its type, its service demand and arrival rate (for open system) or the total customer population (for closed system). The output parameters are system response time, throughput, average number in system and the utilization of each center.

Depending on the job behavior, QN models are divided into single class QN models and multi class QN models. In the single class QN all the jobs are identical. Different behavior of the jobs are allowed in multi class QN model. The technique used to evaluate any closed queueing networks in this thesis is known as Mean Value Analysis (MVA). More detailed discussions are provided in the appropriate sub sections. For some queueing network examples to be discussed in the following sections, the output intervals of the BNR Prolog program runs with point intervals were compared with exact results computed with point form inputs using QNAP2 MVA solver [Pot84]. The results were found to be in close agreement.

4.2.1 Single Class Queueing Network Model

The solution technique Mean Value Analysis (MVA) for the single class QN model is based on a fundamental Little's law and service center residence time equations [LZGS84]. Those equations are listed here:

- The service center residence time equation

$$R_k(N) = D_k \text{ (for delay centers),}$$

$$R_k(N) = D_k (1 + Q_k(N-1)) \text{ (for queueing centers)}$$

where $R_k(N)$ the residence time at center k when there are N customers in the network, $Q_k(N-1)$ is the queueing delay for $N-1$ customers at the k -th device, and D_k is the demand of the k -th device.

- Little's law applied to the queueing network:

$$X(N) = \frac{N}{\left[Z + \sum_{k=1}^K R_k(N) \right]}$$

where $X(N)$ is the system throughput, K is the number of devices, and Z is the think time.

- Little's law applied to the service centers individually: $Q_k(N) = X(N) \times R_k(N)$

The algorithm for single class MVA is given in Figure 4.1. Interval splitting technique has been used with single class queueing networks of different sizes. Consider an example shown in Figure 4.2, which has three service queueing centers (CPU, Disk 1 and Disk 2) and a delay center (TERM). The workload parameters are defined along with the Figure 4.2. The results shown in Figure 4.3 demonstrate the effect of interval splitting on the looseness of the interval for system throughput. The variation in the interval $[X, X^+]$ produced is plotted against the corresponding number of subintervals. Note that a number of subintervals equal to zero corresponds to results obtained without any interval splitting. As expected, the tightness of the interval improves with the number of subintervals.

Inputs:

N = number of customers

Z = think time

K = number of devices (not including terminals)

D_k = service demand of the k -th device

Outputs:

X = system throughput

Q_k = average number of jobs at the k -th device

R_k = residence time of the k -th device

Initialization:

FOR $i = 1$ to K DO

$Q_k = 0$

END FOR;

Iterations:

FOR $n = 1$ to N DO

 BEGIN

 FOR $k = 1$ to K DO

$R_k = D_k$ (for delay centers)

$R_k = D_k(1 + Q_k(n-1))$ (for queueing centers)

 END FOR;

$$X = \frac{n}{Z + \sum_{k=1}^K R_k}$$

 FOR $k = 1$ to K DO

$Q_k = X * R_k$

 END FOR;

 END

END FOR;

Figure 4.1 Single Class Mean Value Analysis Algorithm

Figure 4.2 Single Class Queueing Network Model (QNM3)

Input parameters for QNM3: $N = 5$, $Z = [1, 10]$, $D_{CPU} = 2$, $D_{disk1} = 3$, $D_{disk2} = 1$

Figure 4.3 The effect of Blind search Algorithm

Another QN example (QNM4) is also tried with single class MVA. This model has five devices (see Figure 4.4). In this model, one of the device demands is given as an interval instead of think time.

Figure 4.4 QNM4: Single Class QN

Input parameters for QNM4: $N = 5$, $Z = 4$, $D_{dev1} = [3, 4]$, $D_{dev2} = 1$, $D_{dev3} = 1$, $D_{dev4} = 1$, $D_{dev5} = 3.25$. The results are presented in Table 4.1. Blind search algorithm is found to be effective in reducing the looseness in the throughput for the model.

Table 4.1 Another example of single class MVA (QNM4)

Description	Computed Throughput
Without split	[0.1805, 0.3333]
No. split = 4	[0.2275, 0.3307]
No. split = 8	[0.2357, 0.3219]
No. split = 12	[0.2384, 0.3189]
No. split = 16	[0.2398, 0.3175]
No. split = 20	[0.2406, 0.3166]
No. split = 50	[0.2426, 0.3145]

The main advantages of the single class QN model are the ease of implementing a solution and a lack of need for multiple sets of data. The disadvantage of this approach is that not all computing systems can be represented using single class QN model and a multi class QN is required. A discussion of this interval analysis in multi class models is presented next.

4.2.2 Multi Class Queueing Network Model

For systems in which the jobs being modelled have significantly different behaviors, such as systems with mixture of CPU and I/O bound jobs, a multi class model can provide more accurate results. This means that some systems can be modelled adequately only by multi class models, since the single class assumption that all jobs are statistically identical is unacceptable. In multi class models, the outputs are given in terms of the individual customer classes. Multi class models are models that require multiple sets of input parameters (one per class) and the solution techniques are more complex for multi class models. Let N_c and Z_c be the degree of multiprogramming and think time for class c whereas $D_{k,c}$ is the demand for class c at device k .

Figure 4.5 Multi Class Queueing Network Model (QNM5)

Input Parameters for QNM5: $N_1 = 1, Z_1 = 1, D_{CPU,1} = 2, D_{disk,1} = [3, 28]$

$N_2 = 1, Z_2 = 10, D_{CPU,2} = 2, D_{disk,2} = 15 - 0.5 \times D_{disk,1}$

Figure 4.6 Non-monotonic behavior of QNM5

The solution technique for multi class Mean Value Analysis [Jai91] is a natural extension of single class counterpart described earlier. A detailed description of multi class MVA is not important in the context of this discussion and is presented in Appendix II. The input parameters used for the queueing network model for the system are included Figure 4.5. The results of applying interval splitting to the example system of Figure 4.5 are presented in Table 4.2. Note that the non-monotonic behavior of the system was observed when the service demand at the disk was changed simultaneously for the two classes - it decreased for one class and increased for the other. Such a change in the model parameters may be the result of simultaneous changes in application characteristics (the mean number of disk visits for example) or a result of a change in disk subsystem management. For example, a change in the proportion of cache allocated to the two applications (with the total amount

of cache in the system remaining unchanged) can result in a decrease in the mean service demand for one application class and an increase for the other class. The changes in disk service demand for this example system are made in the following fashion:

$$D_{disk,2} = 15 - 0.5 \times D_{disk,1}$$

Table 4.2 Multi Class Mean Value Analysis

No of sub-intervals	Total Throughput (X_1+X_2)
0 (without splitting)	[0.02994, 0.10485]
5	[0.05128, 0.08641]
10	[0.05415, 0.08447]
15	[0.05514, 0.08381]
20	[0.05563, 0.08348]
25	[0.05593, 0.08328]
50	[0.05655, 0.08287]
75	[0.05675, 0.08273]
100	[0.05686, 0.08266]

The total throughput is important when we are interested in overall usage and performance of the system. It is clear from Figure 4.6 that the total system throughput is minimized somewhere in the middle of the ranges for the disk demands. Without interval arithmetic, there is no guarantee that this throughput minimum can be accurately estimated. For example, by running the conventional multi class MVA algorithm [RL80] at a number of sample points that lie in between the extreme values for the disk demands may not always be able to determine the minimum total throughput. For this type of problems (where non-monotonicity exists or when monotonicity properties of model are not known) interval arithmetic is extremely useful.

As shown in Table 4.2 both X^- and X^+ are effectively estimated by using Blind search interval splitting. Again a significant amount of tightness in the interval for the model out-

put was achieved by using interval splitting. The benefits seem to be less after 25 subintervals are reached.

Another QN example (QNM6) is also experimented with multi class MVA. This model has five queueing centers (see Figure 4.7). In this model the one of the device demand is given as an interval.

Figure 4.7 QNM6 - Multi Class QN example

Input Parameters for QNM6: $N_1 = 3$, $D_{dev1,1} = [4.5, 45]$, $D_{dev2,1} = 0.75$, $D_{dev3,1} = 0.75$, $D_{dev4,1} = 0$, $D_{dev5,1} = 0.75$,

$N_2 = 2$, $D_{dev1,2} = 0.1$, $D_{dev2,2} = 0$, $D_{dev3,2} = 0$, $D_{dev4,2} = 0.747$, $D_{dev5,2} = 0.05$,

The results are presented in Table 4.3. Blind search algorithm is found to be effective for this model as well.

Table 4.3 Another example for Multi Class MVA (QNM6)

Description	Computed throughput (X1)	Computed throughput (X2)
Without split	[0.0004, 0.4318]	[0.0071, 2.1953]
No. split = 5	[0.0147, 0.3454]	[0.2573, 1.9611]
No. split = 10	[0.0170, 0.2901]	[0.5771, 1.6897]
No. split = 15	[0.0178, 0.2628]	[0.7501, 1.5478]
No. split = 20	[0.01821, 0.2469]	[0.8486, 1.4666]
No. split = 25	[0.0184, 0.2366]	[0.9107, 1.4145]

In the exact single class MVA solution technique, computing the arrival instant queue length for population N is based on the time averaged queue length with population $N-1$. The time and space requirement for the exact algorithm depends on both on the number of centers and the customer population. Exact MVA requires time which is proportional to the product of the number of centers and the number of customers. A similar polynomial relationship between computation time and input parameters exists for multi class MVA. Approximate algorithms for MVA have been invented for reducing the computation time. The adaptability of approximate MVA techniques to interval analysis is the subject of discussion for the following sub section.

4.2.3 Approximate Mean Value Analysis

Exact MVA is a recursive algorithm. Computation of performance with population N in the network requires knowledge of performance with $N-1$ population. For small values of N , exact MVA is not computationally too expensive. However for large value of N , exact MVA may require a large computation time. To reduce the computational complexity, several approximate analysis techniques have been developed [Sch79], [CN82]. Here we describe one such approximation technique for single class QN's known as Schweitzer's approximation [Sch79].

This technique is based on the assumption that as the number of customers in a network increases, the queue length at each device increases proportionally. The residence time equation is thus modified as:

$$R_k(N) = D_k \text{ (for delay centers),}$$

$$R_k(N) = D_k (1 + ((N-1)/N) \times Q_k(N)) \text{ (for queueing centers)}$$

The algorithm is iterative. Each iteration starts with some values for the number of customers $Q_k(N)$ at various devices and ends by recomputing new values for $Q_k(N)$. If the new values are not close to the values obtained at the end of the previous iteration, the iteration continues. If they are sufficiently close, the computation stops. The initial values for queue length would not affect the final result, although they may affect the number of iterations required. The approximate algorithm is described in detail in Appendix III.

For the Queueing Network Model (QNM7, which has a similar structure as QNM3 in Figure 4.2) with input parameters: $N = 50$, $Z = 4$, $D_{CPU} = 2$, $D_{disk1} = [3, 4]$, $D_{disk2} = 1$, the approximate MVA technique along with interval arithmetic is applied and the results are given in Table 4.4. It is seen that interval splitting helps a lot in getting tighter interval. From the table it is clear that better accuracy is reached with longer computation time. So there is a trade-off of between the accuracy and the computation time.

Table 4.4 Approximate Mean Value Analysis for QNM7

Description	Throughput	Time (sec.)
Without split	[0.1412, 1.0000]	1.19
No. split = 4	[0.1412, 0.6712]	3.56
No. split = 6	[0.1699, 0.5446]	7.04
No. split = 8	[0.1888, 0.4891]	11.74
No. split = 10	[0.2010, 0.4583]	17.57
No. split = 12	[0.2096, 0.4387]	24.57
No. split = 14	[0.2159, 0.4251]	32.71

Description	Throughput	Time (sec.)
No. split = 16	[0.2207, 0.4152]	42.02
No. split = 18	[0.2245, 0.4076]	52.52
No. split = 20	[0.2276, 0.4016]	64.14
No. split = 22	[0.2302, 0.3968]	76.93
No. split = 24	[0.2324, 0.3928]	90.88
No. split = 26	[0.2342, 0.3895]	106.03
Actual value	[0.2500, 0.3333]	

4.3 Interval arithmetic for Markov Chain Models

4.3.1 Introduction

Applications consisting of concurrent processes cannot be analyzed by conventional queueing network models. Such a system is often analyzed by using a Markov Chain (MC) model. Markov Chain models are constructed with the knowledge of the precedence relationship between different processes in the system. Depending on the complexity of the system it may have many levels. For example, the Markov Chain model of the Client-Server Model I (see Figure 4.8) has 3 levels (see the Figure 4.10). Each feasible execution state appears once at each level, but it can appear at multiple levels. MC Models assume an exponential service time distribution for all the processes. Once the MC model is developed its balance equations are obtained and they are solved. The balance equations are solved using Gaussian elimination method [Kre88] using MATLAB [MAT92] and C programming language [KR88]. The major disadvantage of the MC model technique is that the state space of the MC grows exponentially with the increase in the size of the model. For the client-server examples to be discussed in the following sections, the output of the BNR prolog program runs with point intervals were compared with exact results obtained from the C and MATLAB programs implementing the Gaussian elimination method. The results were found to be in close agreement.

4.3.2 Client-Server Model I

Client-Server (CS) software structures are very common in distributed computing system. An example is transaction processing system. Remote procedure calls (RPC's) are widely used for intercommunication between the clients and the server and are supported by many operating systems. An example client-server system with two client tasks (client 1 and client 2) and a single server task (server) is shown in Figure 4.8. The client tasks behave in a cyclic fashion. The activities performed by the tasks during client response cycle are shown in Figure 4.9. The response cycle for a client task (say client i) starts with the client executing its own program for a mean period of time T_{ci} . At the end of this period, the client sends a request message to the server; after sending the message client i is blocked waiting for a reply. The request at the server joins a FIFO queue and is served with a mean server execution time T_{si} called the service time for client i . The response at the end of server execution (see Figure 4.9) conveys the results back to the client and allows it to continue. After receiving the response from the server, the client executes its own program for a time T_{ci} and then the cycle repeats.

Figure 4.8 Client-Server Model I (server with 1 phase)

Input parameters: $T_{c1} = [1,3]$, $T_{c2} = 10$, $T_{s1} = 1$, and $T_{s2} = 10$

Figure 4.9 Cycle time representation of Client-Server Model I

Figure 4.10 Markov Chain Model for Client-Server Model I

$$(\mu_{c1} + \mu_{c2}) \times P_1 = \mu_{s1} \times P_2 + \mu_{s2} \times P_3$$

$$(\mu_{s1} + \mu_{c2}) \times P_2 = \mu_{c1} \times P_1 + \mu_{s2} \times P_5$$

$$(\mu_{s2} + \mu_{c1}) \times P_3 = \mu_{c2} \times P_1 + \mu_{s1} \times P_4$$

$$\mu_{s2} \times P_4 = \mu_{c2} \times P_2$$

$$\mu_{s2} \times P_5 = \mu_{c1} \times P_3$$

Where

$$\mu_{c1} = \frac{1}{T_{c1}}, \mu_{c2} = \frac{1}{T_{c2}}, \mu_{s1} = \frac{1}{T_{s1}}, \mu_{s2} = \frac{1}{T_{s2}}$$

Figure 4.11 Balance equations for Client-Server Model I

The Markov Chain model of this simple client-server model is shown in Figure 4.10. This model has five states and each state represents a combination of tasks in execution on the system. The state S_1C_2 implies that the processes S_1 and C_2 are active in that state. The state S_1Q_2 implies that the server is executing the request from client 1 (S_1) and client 2 is waiting to receive a service from server (Q_2) in that state. Transition rates are obtained from the client and server execution times. The general method of solving Markov Chain model is as follows. After the Markov Chain model is developed, its balance equation are obtained (see the Figure 4.11). In Figure 4.11, P_i represents the probability of the system being in the state i and μ_k represents the transition rate as a result of completing k -th task. The balance equations along with the normalization equation $\sum_{i=1}^5 P_i = 1$ are then solved to get the steady state probability of the system being in each state of the MC model. Using those probabilities, one can find out the throughput of the system.

$$\text{Client 1 throughput} = \mu_{s1} \times (P_2 + P_4)$$

$$\text{Client 2 throughput} = \mu_{s2} \times (P_3 + P_5)$$

The results of solving this client-server MC model using interval analysis are shown in Table 4.5. The throughput of client 1 and client 2 are presented in the second and third columns respectively. The fourth column shows the computational time in seconds. Tighter results are obtained as the number of subintervals are increased demonstrating the effectiveness of the blind search interval splitting technique.

Table 4.5 Client-Server Model I Results

Description	C1 Throughput	C2 Throughput	Time (sec.)
Without IS	[0.1853, 0.4017]	[0.0130, 0.0872]	0.52
No. split = 2	[0.1628, 0.4205]	[0.0140, 0.0758]	0.93
No. split = 4	[0.1731, 0.4658]	[0.0141, 0.0712]	2.58
No. split = 10	[0.2199, 0.4704]	[0.0257, 0.0623]	14.15
No. Split = 20	[0.2140, 0.4150]	[0.0344, 0.0502]	51.82
No. split = 30	[0.2179, 0.4003]	[0.0349, 0.0475]	115.03
Actual value	[0.2222, 0.4000]	[0.0399, 0.0444]	2.58

4.3.3 Client-Server Model II

This software structure is exactly same as the previous model but the server has two phases as shown in Figure 4.12. Client tasks are shown as *Client 1* and *Client 2* and the Server task is represented as S_{phase1} and S_{phase2} . A Client task is blocked while its request is being served by the server. The blocking of client occurs in the S_{phase1} . At the end of this first phase a reply is sent to the client that it may resume operation. In S_{phase2} , the server executes clean-up, data commit or other post processing operations. The input parameters for the example system are included in Figure 4.12. The activities performed by the tasks during client response cycle are shown in Figure 4.13. The response cycle for a client task (say client 1) starts with client task 1 sending a request message to the server; after sending the message client 1 waits for the reply until its server task phase 1 (S_{11}) is completed. The request joins a FIFO queue and is served with a mean server execution

time T_{s11} called the service time for client 1 in phase 1. The reply conveys results to the client and allows it to continue. Then the server task enters its second phase of mean duration T_{s12} . The client executes its own program for a time T_{c1} and then the cycle repeats. Similar cycle exists for client 2. T_{s21} is the mean server execution time of client 2 in phase 1, T_{s22} is the mean server execution time of client 2 in phase 2, and the application program execution time for client 2 is given by T_{c2} . QN solution techniques do not solve this problem because of the second phase of service and the combination of different general service time distribution with unequal service demand exhibited by the clients with FIFO queueing.

Figure 4.12 Client-Server Model II

Input parameters: $T_{c1} = 1$, $T_{c2} = 10$, $T_{s11} = [0.5 \ 1.0]$, $T_{s12} = 0.5$, $T_{s21} = T_{s22} = 5$

Figure 4.13 Cycle time representation of Client-Server Model II

The Markov Chain model is shown in Figure 4.14. Transition rates are not shown in Figure 4.14 for the sake of simplicity. The descriptions of the various states are given in Table 4.6. With this small change in the Client - Server system (from model I to model II), the number of states is increased from 5 states to 15 states. The balance equations are developed and they are solved along with the normalizing equation $\sum_{i=1}^{15} P_i = 1$ to get the steady state probabilities of the system being in each state. Using those probabilities one can find out the throughput of the system.

$$\text{Client 1 throughput} = \mu_{s11} \times (P_2 + P_3)$$

$$\text{Client 2 throughput} = \mu_{s21} \times (P_9 + P_{10})$$

Where, $\mu_{s11} = \frac{1}{T_{s11}}$ and $\mu_{s21} = \frac{1}{T_{s21}}$. The results of solving the balance equation is shown in Table 4.7. The client throughputs (C1 and C2 throughputs) are presented in sec-

ond column and the third column. The fourth column shows the computational time in seconds. Without the interval splitting technique client 1 throughput is same as the initial range of the throughput specified in the program i.e. $[0.0, 1.0]$. The Blind search interval splitting is demonstrated to be very effective in this example. Considerable narrowing of the model output is achieved by increasing the number of subintervals.

Figure 4.14 Markov Chain Model for Client-Server Model II

Table 4.6 State Description of Client-Server Model II

State	Description
$C_1 C_2$	Client 1 and Client 2 executing their own application programs.
$S_{11} C_2$	Server executing phase1 of Client 1, Client 2 executing its own program.
$S_{21} C_1$	Server executing phase1 of Client 2, Client 1 executing its own program.
$S_{12} C_1 C_2$	Server executing phase2 of Client 1, Client 1 and Client 2 executing their own programs.
$S_{11} Q_2^1$	Server executing phase1 of Client 1, Client 2 waiting for service in first place of the Server queue.
$S_{21} Q_1^1$	Server executing phase1 of Client 2, Client 1 waiting for service in first place of the Server queue.
$S_{22} C_1 C_2$	Server executing phase2 of Client 2, Client 1 and Client 2 executing their own programs.
$S_{12} Q_1^1 C_2$	Server executing phase2 of Client 1, Client 1 waiting for service in first place of Server queue, Client 2 executing its own program.
$S_{12} C_1 Q_2^1$	Server executing phase2 of Client 1, Client 1 executing its own program, Client 2 waiting for service in first place of Server queue.
$S_{22} Q_1^1 C_2$	Server executing phase2 of Client 2, Client 1 waiting for service in first place of Server queue, Client 2 executing its own program.
$S_{22} Q_2^1 C_1$	Server executing phase2 of Client 2, Client 1 executing its own program, Client 2 waiting for service in first place of Server queue.
$S_{12} Q_1^1 Q_2^2$	Server executing phase2 of Client 1, Client 1 waiting for service in first place of the Server queue and Client 2 waiting for service in second place of the Server queue.
$S_{12} Q_1^2 Q_2^1$	Server executing phase2 of Client 1, Client 1 waiting for service in second place of the Server queue and Client 2 waiting for service in first place of the Server queue.
$S_{22} Q_1^1 Q_2^2$	Server executing phase2 of Client 2, Client 1 waiting for service in first place of the Server queue and Client 2 waiting for service in second place of the Server queue.
$S_{22} Q_1^2 Q_2^1$	Server executing phase2 of Client 2, Client 1 waiting for service in second place of the Server queue and Client 2 waiting for service in first place of the Server queue.

Table 4.7 Client-Server Model II Results

Description	C1 Throughput	C2 Throughput	Time (sec.)
Without IS	[0.0000, 1.0000]	[0.0000, 0.0615]	1.92
No. of splits = 5	[0.0657, 0.4450]	[0.0086, 0.0523]	19.55
No. of splits = 10	[0.0975, 0.3776]	[0.0145, 0.0423]	52.26
No. of splits = 15	[0.1004, 0.3402]	[0.0194, 0.0375]	63.57
No. of splits = 20	[0.1148, 0.3121]	[0.0193, 0.0354]	106.01
No. of splits = 50	[0.1280, 0.2807]	[0.0215, 0.0316]	604.31

4.4 Hierarchical Models of Concurrent System

4.4.1 Introduction

For solving certain class of parallel and distributed computing systems, neither the QN model nor Markov Chain model taken alone is sufficient. A combination of both QN model and Markov Chain model is required. A number of advanced techniques for the estimation of performance for such system have been proposed [TB86], [HT83]. In [KME89] looping constructs were introduced in decomposition method. In [PMLH94] decomposition technique was introduced to Rendezvous Network with Multi-Threaded Client. Comparison of relative merits of such techniques is not the focus of this thesis. A hierarchical modelling technique proposed by Thomasian and Bay [TB86] is chosen as an example for demonstrating the effectiveness of interval-based performance analysis in models that are more complex. An accurate algorithm based on hierarchical decomposition is used for solving concurrent task system. At the higher level the system behavior is specified by a Markov Chain. At the lower level, the transition rates among the states of the Markov Chain are computed using a QN solver. The following section presents the description of the model and the solution technique in the context of an example concurrent system. A more general discussion of the approach can be found in [TB86]. For this

kind of problem where monotonicity property is not proven (or hard to prove), one cannot solve the model at the two end points to get the results. Under such circumstances, only interval arithmetic can give the correct result.

4.4.2 Concurrent Task System Model

The distributed system on which the concurrent application runs is composed of two computing nodes inter-connected by a full duplex communication link as shown in Figure 4.15. The two computing nodes are identical, each consisting of one CPU and two Disks. This model takes into account the delay due to interprocess communication and the full duplex links are represented by two queues and shown in Figure 4.15. It is also possible to model the link between the CPU's by other communication models such as local area network.

Figure 4.15 Hardware Platform for Concurrent Tasks

Figure 4.16 Precedence graph for the Concurrent Application

Figure 4.17 Allocation of Tasks to Computing Systems

Figure 4.18 Concurrent Application: Higher level Markov Chain Model

The application structure is described by the precedence graph shown in Figure 4.16. Each node in the precedence graph represents a task with its own independent thread of control [CD73]. Directed edges represent precedence relationship among tasks. The task at the tail of edge is the predecessor for a task at the head of the edge. A task can start execution only

after all its predecessors have completed. The application runs continuously; as soon as task 5 is completed another statistically identical application is started immediately on the system. Task 1c and task 4c are communication tasks that are executed on the process receiving the message. Task 1c is initiated at the end of execution of task 1 while task 4c is initiated at the end of execution of task 4. Inputs for task 1c and task 4c are as follows: these two tasks make 9 visits to the communication server, the total CPU demand for task 1c is a variable factor in this example whereas task 4c has a total CPU demand of 100 msec. The service demands of the other tasks in the application are presented in the Table 4.8. Task 1 and 4 execute on computing node 1 while the other tasks execute on computing node 2, as shown in Figure 4.17. Note that although the system supports two way communication, the task allocation used for this example results is messages flowing from CS1 to CS2 only.

Table 4.8 Service Demands for Processes in Concurrent Application

Task	CPU demand (msec)	No. of Visits to Disk A	No. of Visits to Disk B	Disk Service Time per visits (msec)
1	Variable	5	5	50
2	800	15	15	50
3	800	15	15	50
4	1000	10	10	50
5	500	5	5	50

In the two-level hierarchical model the higher level model is specified by a Markov Chain (see Figure 4.18) in which each state is labelled with the labels for tasks that execute simultaneously on the distributed system. A closed multi class queueing network is used at the lower level for modelling resource contention. Not all the states in the Markov Chain gives rise to resource contention. For example, state 24 corresponds to the simultaneous execution of process 2 and 4 that execute on two different computing nodes and there is no contention for any resource. State 23 on the other hand corresponds to the execution of

two processes on the same computing node which can produce contention for common resource such as the CPU and the disks. Each state that gives rise to resource contention is shown as a darker node in Figure 4.18. A closed queueing network model is to be solved for each of these states. The requirements of a product form queueing network are assumed to be valid and an exact MVA algorithm is used for computing the throughput of a given class that corresponds to a given task appearing in the state label. These throughputs provide the transition rate for the Markov Chain model. For example, the solution of the queueing network for state 1c4c provides the throughput for two classes of customers - a class that corresponds to task 1c and a class that corresponds to process 4c. The first throughput is the transition rate from state 1c4c to state 234c while the second is the transition rate from state 1c4c to state 1c. The transition rates from other states that do not give rise to any resource contention can be easily computed from the total service demand at the resources used by the task. For example in state 1c4 task 1c executes in CS2 and task 4 executes in CS1 so there is no resource contention in that state. Steady state probabilities are computed by solving the set of balance equation along with the normalization equation

$$\sum_{i=1}^{16} P_i = 1.$$

4.4.3 Sensitivity Analysis for the Concurrent Software System

A BNR prolog program was written for solving the two-level model for the concurrent system of Figure 4.15. Note that the system throughput is obtained as $P_1 \mu_1$ where P_1 is the steady state probability of the system being in state 1 and μ_1 is the total rate of leaving state 1. Determination of sensitivity of the system performance to changes in the different parameters is often required during system design and tuning. Identification of system components that have a strong impact on performance is important. This information can then be used during system upgrading for identifying the device that causes the performance bottleneck. Such analysis can also be used in the identification of software modules

that have a strong impact on performance and therefore require special consideration during system redesign. As an example, the sensitivity of the system throughput to two parameters that correspond to the execution times of two tasks were computed. The results are presented in the Table 4.9.

Table 4.9 Effect of changes in Parameters for Concurrent Application

Case	Task	Parameter varied	Interval for parameter (msec)	Interval for throughput
I	lc	CPU demand	[100, 500]	[0.12634, 0.14623]
II	1	CPU demand	[250, 500]	[0.10426, 0.18011]

The variation in each of these parameters is presented in the interval shown in the fourth column of the table whereas the corresponding change in throughput is recorded in the fifth column. System performance seems to be fairly sensitive to the CPU demand of Task1: a change of approximately 72.75% in system throughput can be expected if the execution time for task 1 is changed from 250 to 500 msec. The message processing time corresponding to inter-process communication as captured in the CPU demand of task lc seems to have a relatively smaller impact on performance: only a change of 15.74% in system throughput is observed when this parameter is changed by a factor of 5.

4.5 Larger Models and Scalability of the Algorithm

The previous examples demonstrate that the interval arithmetic technique is an effective tool for performance analysis. How does the technique scale with a growth in the size of the system being modelled? This question is briefly investigated in this section. The motivation behind this investigation is the impact of interval arithmetic on the scalability of existing analytic models. Some models inherently exhibit poor scalability: an example is a Markov Chain in which the state space increases quickly with a growth in the system size

real systems, we often make assumptions that are often inaccurate. The problem is especially severe for multi class models with FIFO service and greatly differing service times. For exact analytic solutions, separable queueing networks require:

- stochastic assumptions such as knowledge of service time distributions at each service centers;
- all classes to have exponential service distribution with the same mean service time at a node that uses a FIFO service discipline.

These assumptions may be inaccurate for real systems. Approximate solution techniques may be applied but the bounds on error are often unknown. Performance bounds may be useful in such a situation. Another possible application of interval arithmetic in performance bounds based bottleneck analysis is explored.

For systems that can be modelled by single class queueing networks, the bottleneck device is that device that exhibits the highest service demand [Smi90]. With a sufficiently large number of customers this device saturates and the throughput (X) of the system is constrained by the service demand ($D_{bottleneck}$) for this device; i.e., $X \leq 1/D_{bottleneck}$. The situation is more complex in systems with multiple classes of jobs in the multiprogramming mix requiring the use of multi class queueing networks for modelling. The device with the highest service demand for a particular class can be characterized as the bottleneck device for that class. However, which one of these device will saturate and constraint the total system throughput (sum of the throughput for each class) can not be determined in such a simple manner. A technique based on performance bounds is used here for such bottleneck identification. Given the degree of multiprogramming for each class, this technique determines whether or not the system is bottlenecked and identifies the bottleneck device.

in this case is the same as that used for Figure 4.19, except that the number of customers is fixed at 5. These results seem to indicate that the interval arithmetic-based computation techniques scale well with the growth in system size. A number of other models were also analyzed. One of the largest models that was used was a single class queueing network with 50 devices and 5 customers. With a threshold value of 0.001 and $I_{\max}=10$ the execution time for algorithm splitting implementing an exact MVA was observed to be approximately 2.5 minutes on a Sparc 10. The results are encouraging and it seems that reasonable response times can be expected when using interval-based performance analysis on a performance workstation.

Figure 4.19 Variation in CPU time (sec) with Degree of Multiprogramming

Figure 4.20 Variation in CPU time (sec) with Number of Devices

Figure 4.21 Variation in CPU time (sec) with Number of Subintervals

The major drawback of the BNR Prolog program implementing the Blind Search Algorithm is its high memory requirement. Prolog stores the intermediate results in its stack space (for back tracking purposes). Because of this, larger models could not be solved on workstations with limited memory. The memory requirement may have been reduced if the interval arithmetic capability of languages such as C language based interval arithmetic or Pascal language based interval arithmetic were used, as these languages do not store the intermediate results in the stack. Most of the techniques discussed in this thesis except Reverse Modelling (discussed in Chapter 5) do not require this feature of Prolog. Also, the memory requirement is reduced by an improved algorithm known as Selective Search Algorithm. A discussion on this algorithm is given in Chapter 5. Another method of improving scalability with respect to memory requirements for the program is to parallelize the application and run it in multiple nodes. Further discussion on this is provided in Section 6.2 of Chapter 6.

Chapter 5

Improvements and Other Applications

5.1 Chapter Overview

An improvement to the Blind Search Algorithm is presented in this chapter as Selective Search Algorithm in Section 5.2. Two other applications of interval arithmetic in performance analysis are also presented in this chapter. The effectiveness of using interval arithmetic in performing bottleneck analysis and reverse modelling are discussed in Section 5.3 and Section 5.4 respectively.

5.2 The Selective Search Algorithm

When using interval splitting, the minimum of the model output (X^-) can occur only in one sub interval and not in all the subintervals. So is the case for the maximum of the model output X^+ . However the Blind Search algorithm, in each iteration, subdivides the original interval into finer subintervals and computes model outputs for each of these subintervals. It does not discard any sub interval as being infeasible. Is it possible to reduce the number of calculations and hence the computation time which is directly proportional to the number of calculations? Another method of interval splitting termed the *Selective Search* algorithm is presented in this section for the investigation of this question. In this method, as the name indicates, only selected subintervals are searched. A similar algorithm was used by [Ske74] in the optimization domain.

The input parameters for this algorithm are the same as for the blind search algorithm. The operations performed in the first iteration is the same as in the case of Blind search algorithm, but the subsequent iterations are different. The algorithm is explained with the help of Figure 5.1.

Figure 5.1 An example of Selective search method

In each iteration a given interval is split into a number of subintervals and the model is solved for each of these subintervals, and the outputs are stored in result-list. At the end of the iteration, the subintervals where the minimum of lower bounds and maximum of upper bounds occurred are found. Only these subintervals are further split in the next iteration, and the results corresponding to these subintervals are discarded from the result-list (as these subintervals will be represented by their split subintervals in the next iteration). In this specific example let us assume that at the end of the first iteration the minimum of lower bound occurs at $X_1^{(2)}$ and the maximum of the upper bound occurs at $X_1^{(1)}$. As a result both these subintervals are further subdivided in the second iteration and the results

from the first iteration for $X_1^{(2)}$ and $X_1^{(1)}$ are discarded (discarded subintervals are shown as shaded circles in Figure 5.1). At the end of the second iteration, results of subinterval $X_2^{(1)}$, $X_2^{(2)}$, $X_2^{(3)}$, and $X_2^{(4)}$ are compared to find minimum of the lower bounds and the maximum of the upper bounds. Let us assume that now, the minimum of lower bound occurs in subinterval $X_2^{(3)}$ and the maximum of upper bound occurs in subinterval $X_2^{(2)}$. The results of the subintervals $X_2^{(2)}$ and $X_2^{(3)}$ are discarded. The results of the subintervals of $X_2^{(1)}$ and $X_2^{(4)}$ are kept in result-list and the subintervals $X_2^{(2)}$ and $X_2^{(3)}$ are further subdivided. At the end of the third iteration, results of subinterval $X_2^{(1)}$, $X_2^{(4)}$, $X_3^{(1)}$, $X_3^{(2)}$, $X_3^{(3)}$, and $X_3^{(4)}$ are compared and the minimum of the lower bounds and the maximum of the upper bounds are determined. The process continues until either the maximum number of iterations are reached or the desired output precision is reached. In Figure 5.1, at each iteration, the intervals discarded from a previous iteration are indicated by a shaded circle.

As explained earlier some looseness is associated with the computed intervals. The interval that seems to contain the minimum (maximum) may turn out not to be the desired interval when further interval splitting is performed and more accurate results are obtained. That is why, the subintervals whose results are not discarded are stored in a list so that in future iteration if it is found that the minimum or maximum exists in other subintervals, those can be further subdivided. Thus, it is possible for the algorithm to go to an interval from a previous computation, if at the present time, the minimum/maximum is deemed to have occurred in the interval from the previous computation.

In this algorithm, the model is solved only for a fraction of the subintervals at each iteration. For example, in iteration 3, the model is solved 4 times (for subintervals $X_3^{(1)}$, $X_3^{(2)}$, $X_3^{(3)}$, and $X_3^{(4)}$), and the results from the previous iteration (for $X_2^{(1)}$ and $X_2^{(4)}$) along with the results from this iteration are compared. However, in the case of the blind search algorithm, at each iteration, the model is solved for all the subintervals and this number

increases as the number of interval splits increases with the increase in the level of iteration. A significant reduction in the computational time in the case of Selective Search algorithm over Blind Search algorithm is expected. A detailed algorithm is given in Figure 5.2.

In the algorithm, *result_list_lb* implies the list that has lower bound (lb) of the result and $\min(\text{result_list_lb})$ implies the smallest value of *result_list_lb*. Also, *result_list_ub* implies the list that has the upper bound (ub) of the result and $\max(\text{result_list_ub})$ implies the largest value of *result_list_ub*. *Xmin_previous* and *Xmax_previous* are single values. *Nmin_result_list_lb* and *Nmin_result_list_ub* are lower bounds and upper bounds respectively of the results of input subinterval corresponding to *Nmin* which was then split *nns* times. *Nmax_result_list_lb* and *Nmax_result_list_ub* are lower bounds and upper bounds respectively of the results of input subinterval corresponding to *Nmax* which was then split *nns* times. At the end of each iteration, *comb_res_list_lb* and *comb_res_list_ub* contain the lower bounds and upper bounds respectively of results of all the input subintervals that will be used in the next iteration.

BEGIN

Inputs: Interval parameter, Single valued parameters, Initial number of sub-intervals(ns), maximum number of iterations($imax$), tolerance value($threshold$), and narrowed number of sub-intervals(nns)

Initial Values: $i = 0$,

Split the input interval ns

FOR $j = 1$ to ns DO

Compute throughput $[X^-, X^+]$ for j -th sub-interval using MVA,

Store X^- and X^+ in result-list-lb and result-list-ub respectively,

END FOR;

$Xmin_previous = \min(\text{result-list-lb})$

$Xmax_previous = \max(\text{result-list-ub})$

Let $Nmin$ be the index of result_list_lb corresponding to $Xmin_previous$ i.e., $result_list_lb(Nmin) = Xmin_previous$. Let $Nmax$ be the index of result_list_ub corresponding to $Xmax_previous$, i.e., $result_list_ub(Nmax) = Xmax_previous$.

WHILE $i < imax$ DO

$i = i + 1$

Delete the values $Xmin_previous$ from result-list-lb & $Xmax_previous$ from result-list-ub and store the resultant list as new-result-list-lb & new-result-list-ub, Split input subinterval corresponding to $Nmin$ and $Nmax$, nns times,

FOR $k = 1$ to nns DO

Find throughput $[X^-, X^+]$ for k -th subinterval corresp. to $Nmin$ using MVA

Store X^- and X^+ in $Nmin$ -res-list-lb & $Nmin$ -res-list-ub respectively,

IF $Nmin = Nmax$ THEN

$Nmax$ -res-list-lb = $Nmin$ -res-list-lb

$Nmax$ -res-list-ub = $Nmin$ -res-list-ub

ELSE

Find throughput $[X^-, X^+]$ for k -th subinterval corresp. to $Nmax$ using MVA,

Store X^- and X^+ in $Nmax$ -res-list-lb & $Nmax$ -res-list-ub,

END IF;

END FOR;

Figure 5.2 Selective Search Algorithm

```

Combine the results (new-result-list-lb, Nmin-res-list-lb, Nmax-res-list-lb) in
comb-res-list-lb,
Combine the results (new-result-list-ub, Nmin-res-list-ub, Nmax-res-list-ub) in
comb-res-list-ub,
Xmin_current = min (comb-res-list-lb)
Xmax_current = max(comb-res-list-ub)
Let Nmin be the index of the input subinterval corresponding to Xmin_current
Let Nmax be the index of the input subinterval corresponding to Xmax_current
result-list-lb = comb-res-list-lb,
result-list-ub = comb-res-list-ub,
IF ((| Xmin_current - Xmin_previous | < threshold) AND
    (| Xmax_current - Xmax_previous | < threshold))
THEN
    break;
ELSE
    Xmin_previous = Xmin_current
    Xmax_previous = Xmax_current
END IF;
END WHILE;
X= [Xmin_current, Xmax_current]
END

```

Figure 5.2 Selective Search Algorithm (contd.)

To illustrate the effectiveness of this algorithm let us consider the QNM1 given in Figure 3.1 with $N = 5$. The results from both the Blind Search algorithm and the Selective Search algorithm are given in Table 5.1 and Table 5.2 respectively.

Table 5.1 Blind Search Algorithm Results for QNM1

Description	Computed Throughput	Deviation(%)	Computation time (sec.)
without IS	[0.20833, 0.33333]	[16.41, 0.94]	0.8
No of split=5	[0.23045, 0.33333]	[7.54, 0.94]	1.98
No of split=10	[0.23873, 0.33333]	[4.22, 0.94]	5.93
No of split=15	[0.24171, 0.33333]	[3.02, 0.94]	11.92
No of split=20	[0.24359, 0.33333]	[2.27, 0.94]	19.93
No of split=25	[0.24471, 0.33333]	[1.81, 0.94]	29.99
No of split=50	[0.24697, 0.33314]	[0.91, 0.88]	111.7
Exact value	[0.24923, 0.33024]		

Table 5.2 Selective Search Algorithm Results for QNM1

Description	Computed Throughput	Deviation(%)	Computation time (sec.)
Without IS	[0.20833, 0.33333]	[16.41, 0.94]	0.74
No. of iteration=1	[0.23043, 0.33333]	[7.54, 0.94]	1.5
No. of iteration=2	[0.24181, 0.33333]	[2.98, 0.94]	4.55
No. of iteration=3	[0.24471, 0.33140]	[1.81, 0.35]	7.46
No. of iteration=4	[0.247153, 0.33055]	[0.84, 0.094]	10.65
Exact value	[0.24923, 0.33024]

Figure 5.3 A Comparison of Blind search and Selective search Algorithm

In Table 5.1 and Table 5.2, the third column gives the percentage deviation of the computed results from the exact result and the fourth column gives the computation time in seconds. It is clear from the tables that comparable accuracy can be achieved in lesser computational time by using the selective search algorithm. For example, to get the deviation of the resultant throughput less than one percent (both in the upper and lower bounds) the Blind search algorithm takes 111.7 seconds (see Table 5.1) and the selective search algorithm takes only 10.65 seconds (see Table 5.2). Figure 5.3 shows the variation of percentage deviation (or percentage error) with respect to the computation time for both the algorithms.

For the purpose of comparing the Blind search algorithm with the Selective search algorithm, the model (QNM3) was solved using both the algorithms. Figure 5.4 shows the

variation in the system throughput interval $[X^-, X^+]$ produced plotted against the corresponding computation time taken in each algorithm. As expected, the tightness of the interval improves with the computational time in both the algorithms. Tighter interval is obtained in lesser time using selective search algorithm than blind search algorithm. This clearly implies that the selective search algorithm is more efficient one for this example.

Figure 5.4 Comparison of Selective and Blind Search Algorithm

5.3 Interval Arithmetic in Bottleneck Analysis

Interval Arithmetic can be applied to a wide range of problems. This section examines the application of interval arithmetic to bottleneck analysis. Identification of the performance bottleneck or the device that limits system performance is important during system design, planning and in any performance improvement project. To use queuing theory to analyze

real systems, we often make assumptions that are often inaccurate. The problem is especially severe for multi class models with FIFO service and greatly differing service times. For exact analytic solutions, separable queueing networks require:

- stochastic assumptions such as knowledge of service time distributions at each service centers;
- all classes to have exponential service distribution with the same mean service time at a node that uses a FIFO service discipline.

These assumptions may be inaccurate for real systems. Approximate solution techniques may be applied but the bounds on error are often unknown. Performance bounds may be useful in such a situation. Another possible application of interval arithmetic in performance bounds based bottleneck analysis is explored.

For systems that can be modelled by single class queueing networks, the bottleneck device is that device that exhibits the highest service demand [Smi90]. With a sufficiently large number of customers this device saturates and the throughput (X) of the system is constrained by the service demand ($D_{bottleneck}$) for this device; i.e., $X \leq 1/D_{bottleneck}$. The situation is more complex in systems with multiple classes of jobs in the multiprogramming mix requiring the use of multi class queueing networks for modelling. The device with the highest service demand for a particular class can be characterized as the bottleneck device for that class. However, which one of these device will saturate and constraint the total system throughput (sum of the throughput for each class) can not be determined in such a simple manner. A technique based on performance bounds is used here for such bottleneck identification. Given the degree of multiprogramming for each class, this technique determines whether or not the system is bottlenecked and identifies the bottleneck device.

A set of bounds called the Robust Box Bounds were introduced in [MWNP92]. The bounds are robust because they are not based on assumptions required for product form networks. Only the assumptions that the devices employ a FIFO queueing discipline, the device service times are independent with any Increasing Failure Rate distribution, and the number of visits made by a job to a device are independent random variables are required for the validity of the bounds. Two types of upper bounds and a lower bound on throughput for each class are computed. Since this thesis is concerned with the application of the interval arithmetic-based bounding technique for bottleneck identification, only the expression for these bounds are provided. Detailed derivation of the bounds can be found in [MWNP92]. The inequalities corresponding to the bounds are based on the following terminology. The system consists of C classes of jobs or applications and K devices. V_{ck} and S_{ck} represent the number of visits and service times per visit respectively for class c jobs or applications at device k . N_c is the number of applications in class c and Z_c is the think time for a class c application. The total throughput of class c is represented by X_c .

The first type of upper bound on throughput is termed the *no contention bound* since it is based on the optimistic assumption that a job never experiences any queueing delay at a server (equation(a)):

$$X_c \leq \frac{N_c}{Z_c + \sum_{k=1}^K V_{ck} S_{ck}} \quad \text{..... (a)}$$

The second type of upper bound is the *utilization based bound* that is based on the fact that the utilization of any device k can not exceed 1 (equation(b)):

$$\sum_{m=1}^C X_m V_{mk} S_{mk} \leq 1.0 \quad \text{..... (b)}$$

The lower bound on throughput is based on the *pessimistic assumption* that a class c job experiences the largest possible queueing delay at a device from all other jobs in the system (equation(c)):

$$X_c \geq \frac{N_c}{Z_c + \sum_{k=1}^K V_{ck} \sum_{m=1}^C N_m S_{mk} \min \left(1, \frac{N_c X_m V_{mk}}{N_m X_c V_{ck}} \right)} \quad \text{..... (c)}$$

The upper and lower bounds on throughput for class c are expressed in terms of the throughputs of other classes which are also unknown. This makes it difficult to compute numerical values for the bounds. An interval arithmetic-based technique (provided by BNR Prolog) is employed for computing the upper and lower bounds on throughputs. Two types of inequalities constitute the program used for computing the bounds: C inequalities (one per class) of the form equation (a) as well as K inequalities (one per device) of the form equation (b) and C inequalities of the form (c). Each inequality is expressed as a sub-goal in the BNR Prolog program. The program computes an interval for each X_c that is consistent with all the inequalities: X_c^- is the lower bound whereas X_c^+ is the upper bound on throughput for class c . Further details of the technique can be found in [MWNP92].

The performance bounds described in the previous paragraph are used in the analysis of bottlenecks in multi class queueing networks. A system is said to be *bottlenecked* if at least one of the devices is saturated. Bottleneck analysis refers to the identification of this device that constraints the total system throughput $X = \sum_c X_c$. Additional clauses of the following form were incorporated into the program used for the bounds computation, for each device k :

$$U_k = \sum_{c=1}^C X_c V_{ck} S_{ck}$$

Note that U_k is an interval and the system is diagnosed to be bottlenecked if there exists at least one device k with $U_k^+ = 1$. The devices with $U_k^+ = 1$ are identified to be the bottleneck devices. The identification of bottleneck is explained with the help of two sample queueing networks QNM8 (as shown in Figure 5.5) and QNM9. Both the examples have one delay center and two devices as shown in Figure 5.5.

Figure 5.5 Queueing Network Model for Bottleneck Analysis

The input Parameters for QNM8 are:

$$N_1 = 50, V_{11} = 1, S_{11} = 0.9, V_{12} = 1, S_{12} = 0.1, Z_1 = 10$$

$$N_2 = 1, V_{21} = 1, S_{21} = 0.2, V_{22} = 100, S_{22} = 0.1, Z_2 = 10$$

The input Parameters for QNM9 are:

same as QNM8 except that $N_1 = 1$ and $N_2 = 50$

The system is observed to be bottlenecked in each case and the bottleneck device is identified in Table 5.3. The validity of the analysis is checked by using the exact MVA for each of these networks. Note that both networks are in product form and the exact MVA algorithm is applicable. The system throughput X as computed by the MVA algorithm is displayed in the third column of the table. The fourth column contains the expected system

throughput (computed by the MVA algorithm) when the bottleneck device is replaced by a faster device: for QNM8, S_{11} and S_{21} are changed to 0.5 whereas for QNM9 S_{12} and S_{22} are changed to 0.05. The fifth column corresponds to the situation in which the device that is not the bottleneck device is replaced by a faster device: for QNM8, S_{21} and S_{22} are changed to 0.05 whereas for QNM9 S_{11} and S_{12} are changed to 0.5. As expected, in each case changing the bottleneck device produces a substantial gain in throughput while changing the other device does not produce any significant performance improvement.

Table 5.3 Bottleneck Identification in Multi Class Queueing Networks

Model	Bottleneck device	System Throughput(X)	X (Bottleneck device changed)	X (Other device changed)
QNM8	1	1.111	2.0	1.111
QNM9	2	0.162	0.273	0.163

Note that the bottleneck identification method based on performance bounds has two important attributes. Firstly, the bounds are not only valid for product form networks but also for a broad range of non-product form systems. Secondly, the time of execution of the algorithm is not dependent on the number of customers in the system. Other algorithms such as exact MVA are restricted to product form networks only and their execution time, grows with the population in each class. This investigation shows the potential of using interval arithmetic based bounding techniques for bottleneck analysis. Further study of larger systems is proposed.

5.4 Reverse Modelling

Distributed software systems are typically composed of a number of modules each of which may be developed separately. The modular construction of the system is also vital for incremental growth: the system may start with a basic configuration and additional modules may be added periodically over the entire lifetime of the system. Given that an existing system is to be augmented, it is often required to use the performance parameters

for the existing modules in conjunction with the overall performance objective for the entire system and produce a set of specifications for the additional module; meeting these specification will enable the system to achieve at least a minimum guaranteed level of performance. The reverse modelling is used to find the parameter of our interest by keeping all the other parameters unchanged. In general, given the analytic model, finding inputs that correspond to specific output is known as *reverse modelling* (see Figure 5.6). This is a new technique and there is no previous literature reported in this area. With help of the relational interval arithmetic capability of the BNR Prolog language, this *reverse modelling* is possible. Given the software architecture of the system, the characteristics of the component modules and the performance specification for the entire reverse modelling may be used to generate specifications for the individual modules. This is important during system design as well as for upgrading an existing system.

For example, consider a client server system as shown in Figure 5.7 and let us assume that only a single client module (client 1) and server module exists at present and the system needs to be expanded by adding another client module (client 2). Given the mean service time for the new client at the server, how does one produce a performance specification for a new client such that system performance is maintained at a desired level? For this kind of study reverse modelling is done by considering future performance specification and fixed workload characteristics as the input to the reverse modelling and the unknown workload characteristic is found for the new client module (client 2) as the output of the reverse modelling. Thus reverse modelling is used to find the parameter of our interest by keeping all the other parameters unchanged.

Figure 5.6 Reverse Modelling

Figure 5.7 An example to illustrate Reverse Modelling

Consider a simple example where there is a delay center and a single queueing center as shown in Figure 5.8. The input parameters for the model are given at the end of the Figure 5.8. For this model, the throughput (X) is monotonic with respect to think time (Z) for $N=3$ (proof is given in Appendix IV). A system with such a monotonic behavior is chosen for validating the result with ease. The technique is applicable to other types of systems as well. The application of the exact MVA algorithm to the two end points of the range for Z yields the results as given in Table 5.4

2 of/de 2

PM-1 3½"x4" PHOTOGRAPHIC MICROCOPY TARGET
NBS 1010a ANSI/ISO #2 EQUIVALENT

PRECISIONSM RESOLUTION TARGETS

Figure 5.8 QNM10: An example for Reverse Modelling

Input Parameters for QNM10 are: $N=3$, $D=4$ sec, $X=[0.1795, 0.2495]$

Table 5.4 Exact Throughputs for given Z

Think time (sec.)	Throughput
$Z = 10$	0.1795
$Z = 1$	0.2495

Let us assume that think time (Z) is not known. With the following minor changes in the existing MVA program, *reverse modelling* technique is applied to this problem. An interval $[0.1795, 0.2495]$ for system throughput (that corresponds to an interval $[1, 10]$ for Z) is fed to the analytic model. The corresponding interval for Z , an input workload parameter is obtained as output of the reverse modelling program. The value of Z found by the *reverse modelling* program is close to the actual interval for Z that produces the given throughput intervals. To reduce even the small deviation in the resultant Z value, interval splitting is applied. The improvements achieved with a large number of subintervals seems to be limited for this example. These values are given in Table 5.5.

Table 5.5 Reverse modelling results

Description	Z value
Without IS	$[0.1387, 10.4962]$
No. of Split = 10	$[0.1585, 10.0314]$
No. of Split = 20	$[0.1585, 10.0208]$
No. of Split = 30	$[0.1585, 10.0174]$

Description	Z value
No. of Split = 40	[0.1585, 10.0158]
No. of Split = 50	[0.1585, 10.0148]

Another example (QNM11) was studied for reverse modelling. This model has a delay server and two devices as shown in Figure 5.9.

Figure 5.9 QNM11 - An example for Reverse Modelling

The input parameters for QNM11 are: $N=2$, $D_1=4$ sec, $D_2=2$ sec, $X=[0.1159, 0.2029]$

The application of the exact MVA algorithm to the two end points of the range for Z yields the results as given in Table 5.6

Table 5.6 QNM11: Exact throughputs

Think time (sec.)	Throughput
$Z=10$	0.1159
$Z=1$	0.2029

Reverse Modelling is applied to this example with system throughput specified as $[0.1159, 0.2029]$. Even without interval splitting the resultant Z computed by reverse modelling is $[1, 10]$. The investigation presented in this section demonstrates the potential of using relational interval arithmetic for reverse modelling. Using this technique in conjunction with larger models can form an important direction in further research.

Chapter 6

Conclusions

6.1 Research Summary and Contributions

With the ever increasing complexity of the present day computer systems, performance analysis has become an important issue during the system design, development and implementation. Currently all the existing performance models accept a set of *single* valued parameters. In many situations, exact values of system parameters are not available, but a range or interval may be specified. Computation of intervals for performance measures that correspond to interval input parameters is therefore required. Complex situations arise when the model output is non-monotonic with respect to any of its input parameters or when the monotonicity properties of the model are unknown. It is not guaranteed that the minimum or maximum values for the performance metric can be accurately estimated by running the conventional model solver for various single point values, within the specified range of input parameters. Interval arithmetic provides a natural solution to this kind of problem by simply representing the input data as an interval.

In this thesis, the effectiveness of interval arithmetic in performance analysis has been demonstrated. To tighten the looseness in the results, two algorithms namely, Blind Search and Selective Search, which make use of interval splitting were introduced. Several models and computing systems such as Queueing Network models, Markov Chain models Client-Server Systems, and message passing Concurrent Software systems were solved using

the interval arithmetic-based techniques. A number of programs in MATLAB, QNAP2 and C were written to test the results obtained from BNR Prolog for specific point intervals. For example, the output intervals obtained from the BNR Prolog program for QNM3 were compared with exact results computed by the QNAP2 MVA solver. A close agreement was found. The application of interval arithmetic in sensitivity analysis and scalability for large queueing network systems has been studied. Techniques for identifying the bottleneck device based on performance bounds that are constraining the total throughput of a multi class system were presented. These techniques are valid for a broad class (including both product form and non-product form) of general queueing networks. A new concept called reverse modelling, in which the performance metric and some workload parameters are specified as input to the solver that computes a key workload parameter that meets the specified performance metrics, is investigated. The relational interval arithmetic provided by BNR Prolog has been found to be useful in this context.

Interval arithmetic has been found to be very effective in performance analysis of computing systems. The relational interval arithmetic provided in the BNR Prolog language has been found to be appropriate for performing interval-based performance analysis.

The main contributions of this thesis are:

- Demonstration of the effectiveness of Interval Arithmetic in solving various performance modelling problems through several popular analytic techniques.
- Algorithms to reduce the looseness in the outputs introduced by the use of interval arithmetic engine.

A part of the work from this thesis is written as a paper "Interval-Based Performance Analysis of Computing Systems" [MR95]. In this thesis, the largest QNM studied had 50 devices and 5 customers. As far as computational complexity is concerned, the algorithm

seems to scale well. However, as discussed on Section 4.5, the storage requirement for the program may cause a problem for larger models.

6.2 Future Work

Future research can be carried out to address the high memory requirements of using BNR Prolog to solve analytical models. An effort can also be made to invent algorithms to further reduce the computational time. One approach to address both these issues is to parallelize the program and run it in multiple computing nodes. Investigation of techniques for parallelizing program's using interval splitting and running the program on parallel hardware is underway at the department of Computer Science in Brandeis University [Sur94]. Execution time of the interval splitting algorithms depends on the initial number of subintervals as well as the number of subintervals used in the subsequent iterations. Techniques for choosing appropriate values for these parameters warrants investigation. Bottleneck analysis and reverse modelling of larger and more complex models can form an important direction in future research. A generalized reverse modelling tool can also be developed, which, when given the desired output intervals, can accurately predict the input intervals for a class of models. During early stages of software design, input parameters may not be accurately known but a wide range may be specified by the designer. With progress in system design, as more and more system parameters become known, tighter intervals may be provided for the input parameters. Interval-based performance analysis can be useful in these various stages of system design. Investigating the use of interval-based performance analysis during design or re-engineering of software systems can form an interesting direction for further research.

References

- [AH83] Alefeld and Herzberger, "Introduction to Interval Computation," Academic press, 1983
- [BNRR88] Bell-Northern Research Ltd., Ottawa, Canada, BNR Prolog Reference Manual, 1988.
- [BNRU88] Bell-Northern Research Ltd., Ottawa, Canada, BNR Prolog User Guide, 1988.
- [CD73] E. G. Coffman and P. J. Denning, "Operating systems Theory," Prentice-Hall, 1973.
- [CM87] Clocksin-Mellish, "Programming in Prolog".Springer-Verlag Berlin Heidelberg, January 1987
- [CN82] K. M. Chandy, and D. Neuse, "A Heuristic Algorithm for Queueing Network Models for Computing Systems," *Communication of the ACM*, pp. 126-134, 1982.
- [Dan63] G. B. Dantzing, Linear Programming and Extensions, 1963, Princeton University press, Princeton.
- [DB78] P. J. Denning, and J. P. Buzen, "The Operational Analysis of Queueing Network Models," *Computing Surveys*, pp. 225-261, 1978.
- [Han92] Elden Hansen, "Global Optimization using interval analysis," Marcel Dekker Inc. 1992.

- [HT83] Philip Heidelberger, and K. S. Trivedi, "Analytical Queueing Models for Programs with Internal Concurrency, *IEEE Transactions on Computers*, vol. c-32 no. 1, pp. 73-82, January 1983.
- [Jai91] Raj Jain, "The Art of Computer Systems Performance Analysis," John Wiley & Sons, Inc. 1991
- [JJ84] Jerry Banks and John S. Carson, Discrete-Event System Simulation, Prentice-Hall, Inc., Englewood Cliffs, 1984.
- [Kle76] L. Kleinrock, "Queueing Systems," John Wiley & Sons, Inc. 1976.
- [KME89] A. Kapelnikov, R. R. Muntz, and M. D. Ercegovic, "A modeling methodology for the analysis of concurrent systems and computations," *Journal of Parallel and Distributed Computing*, no. 6, pp. 568-597, 1989
- [KR88] B.W. Kernighan and D.M. Ritchie, "The C programming language", 2nd ed. Englewood Cliffs, N.J., Prentice Hall 1988.
- [Kre88] E. Kreyszig, "Advanced engineering mathematics", 6th ed., New York, Wiley, 1988
- [Lin94] Jingping Lin, "Performance Analysis of Client-Server Systems with Multi-Threaded Client," M.Eng. Thesis, System and Computer Eng. Dept., Carleton University, 1994.
- [LZGS84] E.D. Lazowska, J. Zahorjan, G.S. Graham and K.C. Sevcik, "Quantitative System Performance", Prentice-Hall, 1984.

- [Maj91] S. Majumdar, "Interval Arithmetic for performance analysis of Distributed Computing Systems", *Canadian Conference on Electrical and Computing Engineering*, pp. 32.3.1-32.3.4, September 1991.
- [MAT92] MATLAB High-Performance Numeric Computation and Visualization Software, Reference Guide, The MathWorks, Inc., October 1992.
- [Moo66] R.E. Moore, "Interval Analysis", Prentice-Hall, 1986.
- [MR95] S. Majumdar, and R. Ramadoss, "Interval-Based Performance Analysis of Computing systems", SCE-94-22, Real-Time and Distributed Systems Group, Carleton University. It will be published in the preceeding of the International Workshop on MASCOTS'95, Durham NC, January 1995
- [MW94] S. Majumdar, C. M. Woodside, "Robust bounds and throughput guarantees for general closed multiclass queueing networks," *Performance Evaluation* (submitted for publication), 1994.
- [MWNP91] S. Majumdar, C. M. Woodside, J. E. Neilson and D. C. Petriu, "Performance bounds for concurrent software with rendezvous," *Performance Evaluation*, vol. 13, no. 4, pp. 207-236, 1991.
- [MWNP92] S. Majumdar, C. M. Woodside, J. E. Neilson and D. C. Petriu, "Robust Box Bounds: Network performance guarantees for closed multiclass queueing networks with minimal stochastic assumptions," *Proceedings of Info-com'92*, pp. 2006-2016, May 1992.
- [Neu90] Neumaier, "Interval Methods for Systems of Equations," Cambridge University Press 1990.

- [Old88] W. Older, "The Application of Relational Arithmetic to X-Ray Diffraction crystallography," Bell Northern Research, Computing Research Laboratory, P.O. Box 3511, Station A, Ottawa, Canada, February 1988.
- [OV89] William Older and Andre Vellino, "Extending Prolog with Constraint Arithmetic on Real Intervals", Bell Northern Research, Computing Research Laboratory, P.O. Box 3511, Station A, Ottawa, Canada,
- [OVF88] William Older, Andre Vellino and Bijan Forrahi, "Relational Arithmetic in Prolog using Interval constraints", Bell Northern Research, Computing Research Laboratory, P.O. Box 3511, Station A, Ottawa, Canada, 1988.
- [Pap91] A. Papoulis, "Probability, random variables, and stochastic processes", 3rd ed. New York, McGraw-Hill. 1991.
- [PMLH94] D. Petriu, S. Majumdar, J. Lin, and C. E. Hrischuk, "Analytic performance estimation of client-server systems with multi threaded clients," *Proceedings of MASCOTS'94*, January 1994.
- [Pot84] Potier, D. "New Users" Introduction to Q_WAP2. Technical report no. 40, INRIA, France, Oct 1984
- [PS82] C. H. Papadimitriou, and K. Steiglitz, *Combinational Optimization: Algorithms and Complexity*, Prentice-Hall, New York, 1982.
- [Rat84] H. Ratschek, "Computer methods for the range of functions," Chichester, Horwood, 1984.
- [RL80] M. Reiser and S. S. Lavenberg, "Mean-value analysis of closed multichain queueing networks," *Journal of ACM*, vol. 27, pp. 313-322, April 1980.

- [RR88] H. Ratschek and J. Rokne, "New Computer Methods for Global Optimization," Ellis Horwood, 1988.
- [RSB93] J. Rolia, M. Starkey, and G. Boersma, "Modeling RPC Performance," *Proceedings of CASCON'93*, pp. 677-690, October 1993.
- [Sch79] P. Schweitzer, "Approximate Analysis of Multi class Closed Networks of Queue," *Proceedings of International Conference on Stochastic Control and Optimization*, 1979.
- [Ske74] S. Skelobe, "Computation of rational interval functions", *BIT*, vol. 14, pp. 87-95, 1974.
- [Smi90] Connie U. Smith, "Performance Engineering of Software Systems," 1990, Addison-Wesley Publishing Company, Inc.
- [SPG92] A. Silberschatz, J. L. Peterson, and P. Galvin, "Operating System Concepts," (Third Edition). Addison-Wesley, 1992
- [Sur94] Suresh Kalathur, Department of Computer Science, Brandeis University, Massachusetts, Personal communication, October 1994.
- [TB86] A. Thomasian and P. Bay, "Analytic queueing network models for parallel processing of task systems", *IEEE Transactions on Computers*, vol. C-35, PP. 1045-1054 December 1986.
- [VA87] P. J. M. Van Laarhoven, and E. H. L. Aarts, *Simulated Annealing: Theory and Application*, Reidel, Dordrecht, 1987.
- [Wag75] H. M. Wagner, "Principles of Operations Research", 1975, Prentice-Hall, Englewood Cliffs.

- [WMN91] C. M. Woodside, S. Majumdar, J. E. Neilson, "Interval Arithmetic for Computing Performance Guarantees in Client - Server Software", *Proceedings of 1991 International Conference on computing and information (ICCI'91)*, pp. 535-546, May 1991.

Appendix I

Monotonicity of Throughput with respect to Service time

(For $N = 1, 2$, and 3)

Consider a single class MVA with one device (QNM1). Let $X(n)$, $R(n)$ and $Q(n)$ be the system throughput, system response time and queueing delay for a system with n customers respectively. It is known that $Q(0) = 0$. Following are the MVA equations for such a model. Let S be the service time of the device, V be the visit ratio to the device and Z be the think time,

For $N = 1$,

$$R(1) = S \times V \times (1 + 0) = S \times V$$

$$X(1) = \frac{1}{(Z + S \times V)}$$

$$Q(1) = X(1) \times R(1) = \frac{S \times V}{Z + S \times V}$$

For $N = 2$,

$$R(2) = S \times V (1 + Q(1)) = S \times V (1 + \frac{S \times V}{Z + S \times V})$$

$$X(2) = \frac{2}{Z + R(2)} = \frac{2(Z + S \times V)}{(Z + S \times V)^2 + (S \times V)^2}$$

$$Q(2) = X(2) \times R(2) = \frac{2 \times (Z + SV)}{(Z + SV)^2 + (SV)^2} \times (SV) \times (1 + \frac{SV}{Z + SV})$$

For $N = 3$,

$$R(3) = S \times V (1 + Q(2)) = S \times V + \frac{2 \times (SV)^2 \times (Z + 2SV)}{(Z + SV)^2 + (SV)^2}$$

$$X(3) = \frac{3 \times [(Z + SV)^2 + (SV)^2]}{(Z + SV) ((Z + SV)^2 + (SV)^2) + 2(SV)^2 (Z + 2SV)}$$

In order to prove that the throughput is monotonic with respect to S , X values are differentiated with respect to S (for number of customers = 1, 2,3) and is shown that all of the first derivatives have negative slope.

$$X(1) = \frac{1}{(Z + S \times V)}$$

$$\frac{d}{dS}X(1) = -\frac{V}{(Z + SV)^2}$$

$$X(2) = \frac{2(Z + S \times V)}{(Z + S \times V)^2 + (S \times V)^2}$$

$$\frac{d}{dS}X(2) = \frac{2 \times V}{(Z + SV)^2 + (SV)^2} - \frac{2(Z + SV)(2 \times ZV + 4SV^2)}{((Z + SV)^2 + (SV)^2)^2}$$

$$\frac{d}{dS}X(2) = -\frac{(2VZ^2 + 4S^2V^3 + 8ZSV^2)}{((Z + SV)^2 + (SV)^2)^2}$$

$$X(3) = \frac{3 \times [(Z + SV)^2 + (SV)^2]}{(Z + SV)((Z + SV)^2 + (SV)^2) + 2(SV)^2(Z + 2SV)}$$

$$\frac{d}{dS}X(3) = -\frac{(14Z^3SV^3 + 72ZS^3V^4 + 72Z^2S^2V^3 + 36S^4V^5 + 3Z^4V)}{((Z + SV)[(Z + SV)^2 + (SV)^2] + 2 \times (SV)^2(Z + 2SV))^2}$$

Hence it is proved that the throughput is monotonic with service time for $N = 1, 2$, and 3

Appendix II

Multi Class Mean Value Analysis

(from [LZGS84])

Inputs:

C = number of classes

K = number of devices (not including terminals)

N_c = number of customers in class c

Z_c = think time of class c

$D_{c,k}$ = service demand of the c -th class at k -th device

Outputs:

X_c = class c -th system throughput

Q_k = average number of customers at k -th device

$R_{c,k}$ = residence time of the c -th class at k -th device

Initialization:

FOR $k = 1$ to K DO

$Q_k(\vec{0}) = 0$

END FOR;

Iteration:

FOR $n = 1$ to $\sum_{c=1}^C N_c$ DO

FOR each feasible population $\vec{n} \equiv (n_1, \dots, n_C)$ with n total customers DO

BEGIN

FOR $c = 1$ to C DO

FOR $k = 1$ to K DO

$$R_{c,k} = D_{c,k} \text{ (for delay centers)}$$

$$R_{c,k} = D_{c,k} (1 + Q_k(\overline{n - 1_c})) \text{ (for queueing center)}$$

where $\overline{n - 1_c}$ is population $\bar{n}$ with one customer removed

END FOR;

END FOR;

FOR $c = 1$ to C DO

$$X_c = \frac{n_c}{Z_c + \sum_{k=1}^K R_{c,k}}$$

END FOR;

FOR $k = 1$ to K DO

$$Q_k(\bar{n}) = \sum_{c=1}^C X_c R_{c,k} = 1$$

END FOR;

END;

END FOR;

END FOR;

More details about the algorithm are available in [LZGS84].

Appendix III

Schweitzer's Approximate Solution Technique

(from [Jai91])

Inputs:

N = number of customers

Z = think time

K = number of devices (not including terminals)

D_k = service demand of the k -th device

ϵ = maximum allowable error in queue length

Outputs:

X = system throughput

Q_k = average number of jobs at the k -th device

R_k = residence time of the k -th device

Initialization:

FOR $i = 1$ to K DO

$$Q_k = N / K$$

END FOR;

Iterations:

WHILE $\max(|Q_k - X * R_k|) > \epsilon$ DO

BEGIN

FOR $k = 1$ to K DO

$$R_k = D_k \text{ (for delay centers)}$$

$$R_k = D_k(1 + Q_k * ((N-1) / N)) \text{ (for queueing centers)}$$

```

END FOR;


$$X = \frac{N}{Z + \sum_{k=1}^K R_k}$$


FOR k = 1 to K DO

     $Q_k = X * R_k$ 

END FOR;

END

END WHILE;

```

More details about the algorithm are available in [Jai91].

Appendix IV

Monotonicity of Throughput with respect to Think time

(For $N = 1, 2$, and 3)

Consider a single class MVA with one device (QNM10). Let $X(n)$, $R(n)$ and $Q(n)$ be the system throughput, system response time and queueing delay for n customers respectively. It is known that $Q(0) = 0$. Following are the MVA equations for such a model. Let D be the service demand of the device and Z be the think time.

For $N=1$,

$$R(1) = D \times (1 + 0) = D$$

$$X(1) = \frac{1}{(Z+D)}$$

$$Q(1) = X(1) \times R(1) = \frac{D}{Z+D}$$

For $N=2$,

$$R(2) = D \times (1 + Q(1)) = D \times (1 + \frac{D}{Z+D})$$

$$X(2) = \frac{2}{Z+R(2)} = \frac{2(Z+D)}{(Z+D)^2 + D^2}$$

$$Q(2) = X(2) \times R(2) = \frac{2 \times (Z+D)}{(Z+D)^2 + D^2} \times D (1 + \frac{D}{Z+D})$$

For $N=3$,

$$R(3) = D \times (1 + Q(2)) = D + \frac{2 \times D^2 \times (Z+D)}{(Z+D)^2 + D^2} \times (1 + \frac{2D}{(Z+D)})$$

$$X(3) = \frac{3[D^2 + (Z+D)^2]}{(Z+D)[D^2 + (Z+D)^2] + 2D^2(Z+2D)}$$

In order to prove that the throughput is monotonic with respect to Z , X values are differentiated with respect to Z (for number of customers = 1, 2, 3) and is shown that all of the first derivatives have

$$X(1) = \frac{1}{(Z+D)}$$

$$\frac{d}{dZ}X(1) = \frac{-1}{(Z+D)^2}$$

$$X(2) = \frac{2(Z+D)}{(Z+D)^2 + D^2}$$

$$\frac{d}{dZ}X(2) = \frac{2}{(Z+D)^2 + D^2} - \frac{4(Z+D)^2}{[(Z+D)^2 + D^2]^2}$$

on simplification,

$$\frac{d}{dZ}X(2) = -\frac{(2 \times Z^2 + 4 \times Z \times D)}{[(Z+D)^2 + D^2]^2}$$

$$X(3) = \frac{3[D^2 + (Z+D)^2]}{(Z+D)[D^2 + (Z+D)^2] + 2D^2(Z+2D)}$$

differentiating $X(3)$ with respect to Z and on simplification,

$$\frac{d}{dZ}X(3) = -\frac{3Z^4 + 18D^2Z^2 + 12DZ^3}{[(Z+D)(Z^2 + 2D^2 + 2ZD) + 4D^3 + 2ZD^2]^2}$$

Hence it is proved that the throughput is monotonic with think time for $N = 1, 2$, and 3 .

END

3 1 - 0 5 - 9 5

FIN