1

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes,
sur microfiche

NOTICE

The quality of this microfiche 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 a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche 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 mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

57
National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Division Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

55634

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

Barry Renton Thomas

Date of Birth — Date de naissance

August 29, 1939

Country of Birth — Lieu de naissance

Canada

Permanent Address — Résidence fixe

57 Grosvenor Avenue
Ottawa, Ontario
K1S 4S3

Title of Thesis — Titre de la thèse

Errors in Software Resource Prediction and
Their Propagation in Computer System Models

University — Université

Carleton

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

M. Eng. (Elect)

Year this degree conferred — Année d'obtention de ce grade

1982

Name of Supervisor — Nom du directeur de thèse

Prof. C. M. Woodside

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film

L'auteur se réserve les autres droits de publication, ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur

Date

April 18, 1982

Signature

Barry R. Thomas

ERRORS IN SOFTWARE RESOURCE PREDICTION AND THEIR
PROPAGATION IN COMPUTER SYSTEM MODELS

by

Barry R. Thomas

Submitted to the
Faculty of Graduate Studies and Research
in partial fulfillment of
the requirements for the degree of
Master of Engineering

Department of Systems and Computer Engineering
Faculty of Engineering
Carleton University

March 1982

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

"ERRORS IN SOFTWARE RESOURCE PREDICTION AND THEIR
PROPAGATION IN COMPUTER SYSTEM MODELS"

by Barry R. Thomas, in partial fulfillment of the requirements for
the degree of Master of Engineering.

Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

Department of Systems and Computer Engineering

March 1982

ACKNOWLEDGEMENTS

6 The encouragement, constructive criticisms, helpful suggestions and support of Professor C. M. Woodside were important for the conducting of the research and the production of this thesis.

Diane Dodds skillfully decoded my scrawl and produced the manuscript.

The whole project would have collapsed if it were not for the support and tolerance of my wife, Helen and our boys. To each of these and to the other students and staff who create the positive learning environment of the Department of Systems and Computer Engineering, I say thank you.

ABSTRACT

A methodology to obtain computer system performance parameters from software specifications is presented. Difficulties and errors in obtaining computer workloads from FORTRAN source code are examined. The propagation of these errors in computer systems models is studied.

It is found that an error in the workload on a saturated device of a highly loaded system can result in large errors in the predicted system performance parameters, while an error in an unsaturated device of a highly loaded system has a minimal effect on the predicted system performance. At low loads the error is propagated almost linearly.

TABLE OF CONTENTS

Acknowledgements	i
Abstract	ii
Table of Contents	iii
List of Figures	v
Chapter 1 - Introduction	1
1.0 Introduction	1
1.1 Summary of Related Work	2
1.2 Related Trends in Software Development	4
1.3 Outline and Contribution of This Thesis	6
Chapter 2 - Analytic Computer Workload and Models	7
2.0 Basic Concepts and Terminology	7
2.1 Methods of Modelling Computer Systems	11
2.2 The Graphical Representation of Computer Programs	13
2.3 A Graph Reduction Technique	19
2.3.1 An Illustrative Example	20
2.4 GAP - An Implementation of the Graph Reduction Algorithm	26
2.5 Summary	32
Chapter 3 - Workload Specifications	33
3.0 Introduction	33
3.1 Workloads of FORTRAN 77 Program on a Honeywell Level 66 Computer System	34
3.2 Obtaining Workloads from Programs	38
3.2.1 Some Difficulties	38
3.2.2 Resource Requirements of Some FORTRAN Programs	46
3.2.3 Predicting Using the Methods of Chapter 2	49
3.2.4 Data Dependency and Interdependency	51
3.2.5 Summary of Findings	55
3.3 The Propagation of Errors In The Workload Model	55
3.4 Summary	58

Chapter 4 - Models to Obtain Performance Parameters From Workloads	59
4.0 Introduction	59
4.1 Single Class Bottleneck Model	61
4.2 Single Class Mean Value Algorithm	64
4.3 An Illustrative Example	65
4.4 Summary	67
Chapter 5 - The Propagation of Workload Errors in Performance Models	69
5.0 Introduction	69
5.1 Effect of Workload Errors Using Bottleneck Constraints	70
5.1.1 Case I - Lightly Loaded System	71
5.1.2 Case II - Heavily Loaded System - Error in Non-Bottlenecked Device	72
5.1.3 Case III - Heavily Loaded System - Error in Bottlenecked Device	73
5.1.4 Case IV - An Error Causes a Mistake in the Identification of the Bottlenecked Device	74
5.1.5 Summary of Error Analysis Using the Single Class Bottlenecked Constraints	77
5.2 The Analysis of Workload Error Propagation Using More Exact Methods	77
5.3 Extended Mean Value Algorithm	83
5.3.1 Comparison of Bottleneck and Mean Value Algorithms	86
5.4 Illustrative Examples	89
5.4.1 High and Low Loads	89
5.4.2 Moderately Loaded System	94
5.5 Conclusions	98
Chapter 6 - Conclusions	99
References	100
Appendix	

LIST OF FIGURES

2.1	Model of a Simple Computer System	8
2.2	The Flow of Information to Model a Computer System	12
2.3	Graphical Symbols	14
2.4	An Example of Graphical Representation of a Program	16
2.5	An Example of the Graphical Representation of a Collapsed Node	17
2.6	FORTTRAN Code Illustrating Erroroneous and Correct Graphical Representation	18
2.7	Reduction of Nodes in Series	21
2.8	Reduction of a Loop	21
2.9	Reduction of Nodes in Parallel	22
2.10	An Illustrative Example - Graphical Representation	24
2.11	Steps in Reduction of Graph	25
	(a) Graph After First Series of Reduction	
	(b) Equivalent Node	
2.12	Overview of Graph Analysis Program (GAP)	27
2.13	Typical Interactive Session Using GAP	28
3.1	Example of STATUS Output	35
3.2	Example of REPORT Output	36
3.3	Costs of Executing STATUS Command	37
3.4	Costs of Executing REPORT Command	37
3.5	Lines of Code Versus Resources Used for a Number of FORTRAN Programs	39
3.6	Typical Output of PM Option on CP-6	41
3.7	Time To Execute a FORTRAN CALL Statement Versus the Number of Parameters	43
3.8	User Execution Time Versus Number of WRITES	44
3.9	User Execution Time Versus Number of Parameters of a FORTRAN WRITE Statement	45
3.10	Summary of Execution and Service Times Versus Number of Statements for Some FORTRAN Programs	47
3.11	Measured Total CPU Times Versus Fitted Times	50
3.12	Analysis of Some FORTRAN Programs	52
3.13	Example of Two Nested Loops	54
3.14	Graph of a Simple Program	57
4.1	Model of Computer System	60
4.2	Typical Single Class Bottleneck Constraints	63
5.1	Bottleneck Error	75
5.2	Response Time Sensitivity - Bottleneck Model	76
5.3	An Open Queue	78

5.4	Plots of $\bar{w}$ versus U and $\frac{\delta \bar{w}}{\delta x}$ versus U for M/M/1 Queue	80
5.5	Simple Closed Queueing Model	81
5.6	Rate of Change of Response Time With Changing CPU Service Times for Various CPU Utilizations	82
5.7	Simple Closed Central Server Model	90
5.8	Sensitivity of Response Time	91
5.9	Sensitivity of Throughput	92
5.10	Sensitivity of Throughput to Errors in Workload of a Non Bottleneck Device	93
5.11	Summary of Results for Model of Figure 5.7 for Low and High Loads	95
5.12	Moderate Loads - Observations	96
5.13	Moderate Loads - Observations	97

Chapter 1

INTRODUCTION

1.0 Introduction

Managers and designers want to know how well proposed new software will perform on a computer system. They also want to know how sensitive the predicted performance is to the accuracy of the estimates they make and which estimates are most critical.

This thesis explores the problems in obtaining computer workloads, usually in seconds of required service time, from program source code, the range of accuracy to be expected and the possible resultant errors in predicted system performance (response time, throughput and utilization). At high loads it is shown that the accurate estimate of the workload of the most utilized device is critical in determining system performance.

It is assumed that the target system for the proposed software is known such that an analytical model exists or may be developed using the tools presented in Chapter 4.

Some of the reasons to predict the performance of a computer system with proposed software are:

1. If the performance does not meet specifications, modifications to the software can be explored at early stages of development when costs are lower.
2. If the performance of an existing computer will degrade below

acceptable levels appropriate actions can be taken. If this involves the purchase of additional hardware, considered a capital expenditure in many organizations, there is lead time to obtain budget approval and delivery of new hardware.

3. The administration of most organizations requires that large purchases be justified on a cost-benefit basis. Managers require estimates of both the total cost of proposed software plus estimates of the benefits to be derived from its use. The cost of running a proposed program has to include the cost of the system on which it runs.
4. By identifying the critical elements, the method allows the design effort and expenditures to be directed where there is the greatest potential return on investment.

Initial estimates can be made as soon as the basic structure of the software is determined. The estimates are updated when more accurate information is available. This process can continue until measurements on the operational computer system are made.

1.1 Summary of Related Work

Sanguinetti has done initial work on complex programs which include parallel computations [SAN78]. He starts with the Program Process Modelling Language (PPML) developed by Riddle. PPML is described as "a modelling method for systems which are composed of independent tasks which operate potentially in parallel and communicate with one another in a uniform manner". Sanguinetti demonstrates how PPML maps into Message Transfer Expression (MTE) and he presents

some analytical techniques to reduce MTE's. His method is not fully developed. The concept of obtaining the parameters from initial specifications is important.

Booth and Wiscek [B0080] develop a methodology which is based on extending abstract data types to include performance parameters. These are the terms "performance abstract data types". Abstract data types were developed to reduce the amount of complexity or detail that a programmer must consider at one time [GUT77]. Basically, they separate detailed manipulation of data from the user. In this sense they are part of the movement to modularity in programming. Booth and Wiscek associate a cost function with each data type and provide analytic methods to obtain the total cost. The total cost is the workload of that program on a particular system.

DOTY Associates have produced Handbook of Procedures for Estimating Computer System Sizing and Timing Parameters [DOT80] for the United States Department of Defence. Their focus is upon embedded computer systems. They recognize that "the primary difficulty is the definition of a workload that a computer system or subsystem must perform" [DOT80, p.2]. They describe the need for research in the area of workload prediction, and point to the importance of analogy and past experience in estimating workloads.

Beizer [BEI78] gives a detailed outline of methods to analyze simple programs. A digraph is constructed to represent the program. Techniques are given to reduce the graph. He discusses in detail methods

to handle interrupts, resource contention, etc. While his emphasis is upon small systems, many of his insights and methods are applicable to larger systems. His book is an excellent reference.

Smith [SMI80] also uses a graphical representation of the program and provides techniques to reduce the graph based on the work of Kelly [KEL74]. The method presented in Chapter 2 is based on the work of Smith.

Recently, BGS System Inc., have marketed the software modelling package CRYSTAL, which will analyze the expected resource requirements of programs using as input either the number of instructions and the MIPS (millions of instructions per second) or the CPU service time plus disk accesses, etc. CRYSTAL in conjunction with the computer system modelling package BEST/1, can be used to provide performance parameters, [BUZ81]. At present CRYSTAL only uses expected values and does not include variances. Information Research Associates have also marketed a product to predict system performance from program descriptions [EDP81].

1.2 Related Trends in Software Development

One of the most important aspects of recent developments in techniques for the systematic design of well-structured and reliable software architecture for this thesis is modularization [YEH78, COM81]. Modularization is the division of a large program into separate modules. Each module is a logical component of the overall system which is linked to other modules in a clear, concise manner.

Data which is local to one module is hidden and unavailable to other modules. This separation of a program into modules means that the resource requirement of each module may be estimated separately. Or, if existing modules are used in a program, their resource requirements may be directly measured. ADA, the new programming language sponsored by the United States Department of Defence supports modularization through its packages and tasks [COM81, BOW81].

There is a continuous effort to assist the designer and programmer by the development of tools [RID80]. The focus of much of this research is upon efficient coding methods and proof of correctness. These are important aspects of software production. There is also concern that workload parameters be available early in the design process [DOT80]. The explorations of this thesis are part of the move toward the production of efficient prediction tools.

If the analysis of a proposed additional workload suggests a degradation of a system, the workload may be reduced by finding a more efficient algorithm or by flow analysis. The study of efficient algorithms is an important topic [KNU68, AH074]. If an algorithm is found which uses fewer calculations and data transfers then the workload might be reduced to an acceptable level.

Flow analysis is a prerequisite to many important types of code improvement. Data flow analysis and control flow analysis study existing programs to locate unnecessary calculation and data transfer [HEC77]. An early analysis of the performance of a computer system could

justify the application of code improvement to an existing workload, so that an additional workload will not degrade the system.

1.3 Outline and Contribution of This Thesis

Chapter 2, based primarily on the work of Smith [SMI80], presents tools to obtain workloads from source code or source code estimates. Chapter 4 uses these workloads and a model of the computer system to obtain expected response times and throughputs. Chapter 3 and 5 represent the major contribution of this thesis. Chapter 3 is a study of the relationship between the source code of some FORTRAN77 programs and workloads on the Honeywell Level 66 Computer System of Carleton University with an emphasis on the expected accuracy of predictions. Chapter 5 studies the effect of an error in the estimated workload upon the predicted computer system performance.

Chapter 2

ANALYTIC COMPUTER WORKLOAD MODELS

2.0 Basic Concepts and Terminology

The operation of a computer system can be conceptualized as jobs representing programs moving among servers. Servers may be hardware units such as central processing units (CPU) or disks, or they may be software items such as schedulers. Jobs which have similar patterns of service are grouped into classes. Figure 2.1 illustrates a model of a simple computer system which consists of N active terminals, a CPU and two disk units. A job circulates from its initiation by a user at a terminal to the CPU, then with probability P_1 to the terminal, P_2 to disk 1 and P_3 to disk 2. From disk 1 and disk 2, the job always goes to the CPU. In general

$$\sum_{j=1}^M P_{ij} = 1 \text{ for all } i \quad (2.1)$$

where

P_{ij} = probability of a transition from device i to device j

M = total number of devices

A model in which jobs always move between the CPU and other devices is a central server model. One in which there is a fixed number of jobs is closed. One in which there is both a source of new jobs and a sink by which jobs leave the system is open. A model may be open for some job classes and closed for others. The average time between the

FIGURE 2.1: MODEL OF A SIMPLE COMPUTER SYSTEM

initiation of a class r job and its return to the user's terminal is the response time $T(r)$. The average time between the completion of one job and the initiation of the next job at an active terminal is the think time $TH(r)$. Jobs may arise at sources other than terminals such as a sensing device. Then $TH(r)$ is used to represent this time.

Batch jobs are scheduled and run by the computer. They may have been entered either at a terminal or by another input-output (I/O) device such as a card reader. Jobs which are entered directly from a terminal, as in the example, are "interactive".

The average time for a class r job to complete one round trip is the cycle time $C(r)$.

$$C(r) = T(r) + TH(r) \quad (2.2)$$

The throughput of class r jobs, $f(i,r)$ is the average rate at which class r jobs are serviced as perceived from device i . Normally, the throughput in interactions per second is specified with reference to the terminal or sensing device. It will be represented by $f(r)$. A visit is one pass through a device. $y(i)$ is the number of visits per interaction to device i . $y(i) = 1$ for the reference device. If there are $N(r)$ jobs in a class with a throughput $f(r)$ and a cycle time $C(r)$

$$f(r) = \frac{N(r)}{C(r)} \quad (\text{Little's Formula}) \quad (2.3)$$

This simple formula is basic for analyzing computer systems.

The utilization of device i by class r jobs, $U(i,r)$, is the fraction of time that device i is busy servicing class r jobs.

Thus

$$\sum_r U(i,r) \leq s(i) \text{ for all } i$$

where $s(i)$ = number of parallel hardware servers making up device i

The workload of class r jobs at device i is the average time to process a class r job per visit, $x(i,r)$ or per interaction $z(i,r)$. $y(i,r)$ is the number of visits per interaction of a class r job.

$$z(i,r) = y(i,r) \times x(i,r) \quad (2.4)$$

The workload which a program presents to a device per visit is the sum over the average of all the functions performed multiplied by the average time to perform each function

$$x(i,r) = \sum_j p_j(i) x_j(i,r) \quad (2.5)$$

where $p_j(i)$ represent the average number of functions j on device i
 $x_j(i,r)$ average time per function j
 j - represent a logical function

This chapter discusses methods of obtaining workloads from programs.

A module is a logical unit of a program which has a finite clearly defined relationship to other logical units within the program. A module may be a procedure, a subprogram, a function, or a single machine instruction or a combination of these which form a distinct logical unit.

2.1 Methods of Modelling Computer Systems

The mapping of programs to system performance parameters is by means of direct measurements or models. Modelling is usually in two steps. First, workload are obtained from program code, then system parameters are obtained from workloads. The first is the subject of this chapter, the second is the subject of Chapter 4.

There are three basic methods of modelling computer systems - simulation, prototypes and analytically. These can be used independently or jointly.

Simulations can be accurate but are costly and require skilled analysis of the results. It can be used to model complex situations. It is not considered in this thesis.

Tractable analytic models of computer systems usually involve assumptions that limit their accuracy. They provide a relatively quick method of locating problems and in choosing between various design options. They are the focus of this thesis.

Prototypes can provide useful information, particularly once a design has been chosen. In designing software it may be possible to obtain workloads by actually running some of the modules on an existing system. If a mapping is available between the execution speeds of two systems, then the expected execution time on a target system can be estimated from a development system. Embedded system may be designed this way. Care must be taken in this procedure because

FIGURE 2.2: THE FLOW OF INFORMATION TO MODEL A COMPUTER SYSTEM

- 1) While there may be a known relationship between the MPS (millions of instructions per second) or KOPS (thousands of operations per second) on two processors, the ratio of individual instructions may vary from these [LIA80].
- 2) The fetch cycles on the two machines may be different and affect the execution times.
- 3) The compilation of identical source code may be different on two processors.

This method is not available at early stages of writing programs. It requires that accurate tools be available to monitor the execution of the program (see [PLA81]).

Figure 2.2 diagrams the logical mixing of hardware and software elements to produce a system model from which system performance characteristics may be determined. In the next section we examine how a program may be represented by a graph and in subsequent sections explore how this graph may be collapsed to a single equivalent node.

2.2 The Graphical Representation of Computer Programs

Figure 2.3 presents a set of symbols which can be used to represent a computer program. These are a modification of the symbols used by Smith [SMI80]. Each block represents a logical segment of the program called a "module". In top down design a module may initially represent an amalgamation of many functions. As the design of the program progresses a module itself becomes a graphical structure. This process continues

GRAPHICAL REPRESENTATION:

SYMBOL	NAME	USAGE
basic BA	a functional component whose characteristics are defined at this level.
collapsed CL	a functional component whose characteristics are defined at a lower level. The characteristics at this level represent an aggregation.
repetition RE	the head of a loop which is to be repeated N times. It can have an execution time associated with it.
fork	the beginning of a concurrent segment of a program.
join	end of concurrency begun at last fork. Concurrent loops cannot intersect.
or-node BO DO	any node with two or more arcs leaving. Each arc has a probability associated with it. A basic or-node (BO) has resource requirements associated with it. A dummy or-node has no resource requirements associated with it. The sum of the leaving probabilities must be 1.
end BE DE	a node which together with the repetition node brackets a segment of code. If it has resource requirements associated with it, it is called a basic end (BE), otherwise called a dummy end (DE).
terminal BT DT	a node which together with an or-node brackets a segment of code. If it has resource requirements associated with it, it is called a basic terminal (BT), otherwise called a dummy terminal (DT).

FIGURE 2.3: GRAPHICAL SYMBOLS

until logical units are reached for which resource requirements are known on the target machine. Each module can have a mean resource requirement and variance. Those which do not are termed "dummy".

In creating a graphical representation of a program for analysis there is no need to follow the exact flow of the program. It is the total resource requirements which are important. Section 2.3 shows that the expected resource requirements of modules in series and their variances add. Thus, the resource requirement of serial modules may be amalgamated. In many programs there is code or modules which will not normally be executed. This could be code to handle rare or exceptional situations. Any code which has a low probability of execution can usually be ignored.

In specifying an or-node care must be taken that it is a random data-dependent or-node and not simply a means of expressing a deterministic sequential process. Figure 2.6 shows code which could be erroneously modelled as an or-node. If only the mean value is required, both representations are equivalent. However, in calculating the variance, representation (a) could lead to the incorrect introduction of variance. It treats a deterministic situation as if it were random. In section 2.3 it is shown that two nodes in parallel, each with a probability of execution, has a variance which is a function of the variance of each branch, their probability of execution and the difference in their expected resource requirement. In the example of Figure 2.6 B is always executed 9 times and A is executed once.

MEANING

The program starts at B1. This is a basic or-node (B0), thus it would have a resource requirement and an associated variance. Depending on the input data it goes either to B2 with probability P1 or B3 with probability P2. $P1 + P2 = 1$. Because B2 and B3 are basic nodes they will have expected resource requirements and an associated variance.

B4 is a basic terminal node signifying the end of a set of branches. Or (B0, D0) and terminal (DT, BT) nodes must occur in pairs.

B5 is a repetition node. N is a random variable with a mean value and a variance. B5 can also have an expected resource requirement and an associated variance.

B6 is a dummy or-node. Branch to B7 has the probability P3, while to B8 the probability is P4. $P3 + P4 = 1$.

B7 is a collapsed node. It will have a mean and variance found by analysing its associated graph. B8 is another basic node.

B9 is a dummy terminal node - with no resource requirements.

B10 is a basic end. Each repetition node has an end node (BE or DE) associated with it to demark the extent of its operation. Nodes B5, B6, B9, B10 are each executed N times on the average. Node B7 is executed an average of $P3 \cdot N$ times and B8 $P4 \cdot N$.

FIGURE 2.4: AN EXAMPLE OF GRAPHICAL REPRESENTATION OF A PROGRAM
(B1 TO B10 ARE PROGRAM MODULES)

SYMBOLISM

MEANING

This module commences at C1 which has expected resource requirement and variance.
 C2 is a fork. This means that C3 and C4 are performed concurrently. Logically this could be by separate servers or by the same servers in competition with each other C5 is a join or synchronization point, both C3 and C4 must be completed before C5 can be executed.
 The analysis of concurrency is more complex than other situations considered.

FIGURE 2.5: EXAMPLE OF THE GRAPHICAL REPRESENTATION OF A COLLAPSED NODE

DO 10, I=1,10

(1)

IF (I.EQ.5) THEN

(A)

ELSE

(B)

ENDIF

(2)

10 CONTINUE

Fortran Code

Erroneous Graphical
Representation
(a)

Correct
Representation
(b)

FIGURE 2.6: FORTRAN CODE ILLUSTRATING ERRORNEOUS AND CORRECT GRAPHICAL REPRESENTATION

Each node of the graph except for dummy nodes has resource requirements represented by a mean and if desired variance. The resource requirements may be expressed as the number of instructions, seconds of CPU time, disk accesses, etc. The resource requirement may be a vector consisting of one or more of these components. In this case there would be several expected values and associated variances for each module.

2.3 A Graph Reduction Technique

The objective of graph reduction methods is to replace the resource requirements of a graph by the resource requirements of a single equivalent node. One method of accomplishing this is to trace each possible path from a beginning to a terminal point, calculate the resource requirements for this path and the probability of following it. The expected resource requirement is the sum of paths weighted by their probability. For small graphs with simple discrete resource requirements this is feasible, for large problems it is impractical. Beizer [BEI78] presents two methods to reduce graphs. One uses techniques similar to those presented in this chapter, the second involves representing the graph as a matrix and solving the matrix. The method presented here follows that of Smith [SMI80] who followed [KEL74].

In order to obtain a single equivalent node the following restrictions apply:

1. There must be a single start and a single end node. Dummy nodes can be used to change any graph to this situation.

2. Loops cannot intersect.
3. Or-nodes and terminal nodes must occur in pairs.
4. Repetition nodes and end nodes must occur in pairs.
5. The $x(i)$'s and $N(i)$'s must be independent.
6. The sum of probabilities out of an or-node must equal one.

Figures 2.7, 2.8, 2.9 represent the reduction formulae. $EX(i)$ represents the resource requirement of node i and $VARX(i)$ the associated variance. $EN(i)$ is the expected number of repetitions of loop i and $VARN(i)$ its associated variance. $EX(T)$ is the equivalent expected resource requirement and $VARX(T)$ the associated variance. Equations 2.3.7 and 2.3.8 can be shown to be equivalent by expanding the right hand side and using the fact that

$$P(i) = 1 - \sum_{\substack{j=1 \\ i \neq j}}^M P(j)$$

2.3.1 An Illustrative Example

Figure 2.10 shows a graph which is to be reduced to a single equivalent node. The numbers (5,0) and (10,10) under A and C respectively, represent the expected number of repetitions and its variance for the loops. The other number in brackets (a,b) represents the resource requirements. "a" is the expected value and "b" its variance.

First equations 2.3.1 and 2.3.2 are applied to the series nodes F and G resulting in

$$EX(T) = \sum_{i=1}^M EX(i) \quad (2.3.1)$$

$$VARX(T) = \sum_{i=1}^M VARX(i) \quad (2.3.2)$$

FIGURE 2.7: REDUCTION OF NODES IN SERIES

$$EX(T) = EN * EX(i) \quad (2.3.3)$$

$$VARX(T) = EN * VARX(i) + (EX(i))^2 * VARN \quad (2.3.4)$$

FIGURE 2.8: REDUCTION OF A LOOP

$$\sum_{i=1}^M P(i) = 1$$

2.3.5

$$EX(T) = \sum_{i=1}^M P(i) EX(i)$$

2.3.6

$$VARX(T) = \sum_{i=1}^M P(i) VARX(i) + \sum_{(i,j) \in I} P(i) P(j) (EX(i) - EX(j))^2$$

2.3.7

where I is the set of all pairs (i,j) with i and j in {1,...,M}

$$= \sum_{i=1}^M P(i) VARX(i) + \sum_{i=1}^M P(i) (EX(i))^2 - \left[\sum_{i=1}^M P(i) EX(i) \right]^2$$

2.3.8

FIGURE 2.9: REDUCTION OF NODES IN PARALLEL

$$\begin{aligned} EX(FG) &= EX(F) + EX(G) \\ &= 5 + 10 \\ &= 15 \end{aligned}$$

$$\begin{aligned} VARX(FG) &= VARX(F) + VARX(G) \\ &= 5 + 1 \\ &= 6 \end{aligned}$$

Applying equations 2.3.3 and 2.3.4 to C and D gives

$$\begin{aligned} EX(CD) &= EN(C) * EX(D) \\ &= 10 * 2 \\ &= 20 \end{aligned}$$

$$\begin{aligned} VARX(CD) &= EN(C) VARX(D) + (EX(D))^2 * VARN(C) \\ &= 10 * 1 + (2)^2 * 10 \\ &= 50 \end{aligned}$$

The application of equations 2.3.6 and 2.3.7 to CD, E and FG gives

$$\begin{aligned} EX(CDEFG) &= .3 * 20 + .2 * 15 + .5 * 15 \\ &= 16.5 \end{aligned}$$

$$\begin{aligned} VARX(CDEFG) &= (.3 * 50 + .2 * 1 + .5 * 6) + (.2 * .3 * (20-15)^2 \\ &\quad + .3 * .5 * (20-15)^2 + .2 * .5 * (15-15)^2) \\ &= 18.2 + 5.25 \\ &= 23.45 \end{aligned}$$

The application of the series rules to B, CDEFG, H and I gives

FIGURE 2.10: AN ILLUSTRATIVE EXAMPLE - GRAPHICAL REPRESENTATION

(a) Graph After First Series of Reductions

(b) Equivalent Node

FIGURE 2.11: STEPS IN REDUCTION OF GRAPH

$$\begin{aligned} \text{EX}(\text{BCDEFGHI}) &= 1 + 16.5 + 1 + 1 \\ &= 19.5 \end{aligned}$$

$$\begin{aligned} \text{VARX}(\text{BCDEFGHI}) &= 1 + 23.45 + 1 + 0 \\ &= 25.45 \end{aligned}$$

Finally, applying the loop rule gives the expected resource requirement $\text{EX}(T)$ and its variance $\text{VARX}(T)$ for a single equivalent node of Figure 2.11(b)

$$\begin{aligned} \text{EX}(T) &= 5 * 19.5 \\ &= 97.5 \end{aligned}$$

$$\begin{aligned} \text{VARX}(T) &= 5 * 25.45 + (19.5)^2 * (0) \\ &= 127.25 \end{aligned}$$

2.4 GAP - An Implementation of the Graph Reduction Algorithm

The above reduction method has been programmed in FORTRAN77 for the Honeywell Level-66 computer at Carleton University. Figure 2.12 shows the logical structure of the program. The graph to be analyzed may be entered either from a formatted file or interactively. The program is structured such that originally it is most convenient to enter the graph interactively. Figure 2.13 illustrates a typical session to enter the graph of the example of section 2.3.4. Once a graph has been entered it can be stored on a file and either modified in place or else modified while in GAP.

A copy of MAIN, the control portion and CALCU, the calculation portion of the program, are included in Appendix A. The basic method are

<u>Module Name</u>	<u>Functions</u>
MAIN	This is the control module for the program. The user may interactively call the other modules at various logical points to perform various functions. For example, output can be evoked at several places to check the values entered. Readin is evoked to enter a new set of values for a node or to change the values of a node.
READGRAF	Reads in a graph given its name and file location. It must be formulated as per Output.
READIN	Reads in either new or modified parameters for one node of a graph.
OUTPUT	Prints out the graph - it can be used to check the graph entered at several points in the execution of the program.
CALCU	Performs the analysis on a graph structure reducing it to a mean value and a variance.

FIGURE 2.12: OVERVIEW OF GRAPH ANALYSIS PROGRAM

!*L

THERE ARE 3 OPTIONS

1. Enter a new graph from an existing file
2. Enter a new graph interactively
3. Make a change in the existing graph,
including where the output is directed
or adding additional nodes.

ENTER 1, 2, or 3

?2

Please give graph name.

?EX1

Please give number of nodes.

?9

Number of resources, max=6

?1

Give node name enter ? to obtain a listing of nodes entered

End to finish

?A

Nodetype, ? to obtain a listing

??

THE NODE TYPES ARE SIGNIFIED BY TWO CHARACTERS

BA	- BASIC
BE	- BASIC,END OF LOOP
BO	- BASIC OR
BT	- BASIC TERMINUS OF BRANCH
CL	- COLLAPSED NODE
RE	- REPETITION
DU	- DUMMY
DE	- DUMMY,END OF LOOP
DO	- DUMMY OR
DT	- DUMMY TERMINUS OF BRANCH

Please give node type

?RE

Enter resource requirement and var node A resource 1

?0,0

Next node

?B

Give # repetitions and variance

?5,0

(Continued on page 29)

FIGURE 2.13: TYPICAL INTERACTIVE SESSION USING GAP

Give node name enter ? to obtain a listing of nodes entered
End to finish
?B

Nodetype, ? to obtain a listing
?B0

Enter resource requirement and var node B resource 1
?1,1

Give node name (must be 3 char)and branching prob.,end to finish
?C .3
?E .2
?F .5
?END

Give node name enter ? to obtain a listing of nodes entered
End to finish
?C

Nodetype, ? to obtain a listing
?RE

Enter resource requirement and var node C resource 1
?0,0

Next node
?D

Give # repetitions and variance
?10,10

Give node name enter ? to obtain a listing of nodes entered
End to finish
?D

Nodetype, ? to obtain a listing
?BE

Enter resource requirement and var node D resource 1
?2,1

Next node
?H

Give node name enter ? to obtain a listing of nodes entered
End to finish
?E

Nodetype, ? to obtain a listing
?BA

(Continued on page 30)

FIGURE 2.13: TYPICAL INTERACTIVE SESSION USING GAP (Continued)

Enter resource requirement and var node E resource 1
?15,1

Next node
?H

Give node name enter ? to obtain a listing of nodes entered
End to finish
?F

Nodetype, ? to obtain a listing
?BA

Enter resource requirement and var node F resource 1
?5,5

Next node
?G

Give node name enter ? to obtain a listing of nodes entered
End to finish
?G

Nodetype, ? to obtain a listing
?BA

Enter resource requirement and var node G resource 1
?10,1

Next node
?H

Give node name enter ? to obtain a listing of nodes entered
End to finish
?H

Nodetype, ? to obtain a listing
?BT

Enter resource requirement and var node H resource 1
?1,1

Next node
?I

Give node name enter ? to obtain a listing of nodes entered
End to finish
?I

Nodetype, ? to obtain a listing
?BE

(Continued on page 31)

FIGURE 2.13: TYPICAL INTERACTIVE SESSION USING GAP (Continued)

Enter resource requirement and var node I resource 1
?1,0

Next node
?END

Give node name enter ? to obtain a listing of nodes entered
End to finish
?END

Copy of graph?
yes - enter TRUE(T)
no - FALSE(F)

?F

Do you wish to make any changes
yes - enter TRUE(T)
no - FALSE(F)

?F

Please give unit number of output file
?6

GRAPH NAME= EX1 NUMBER OF NODES= 9 NUMBER OF RESOURCES= 1

NODE		NEXT NODE(PROB)		REPETITIONS		RESOURCE 1	
NAME	TYPE			NO.	VAR	REP	VAR
A	RE	B	(1.000)	5.000	0.000	0.000	0.000
B	BO	C	(.300)			1.000	1.000
		E	(.200)				
		F	(.500)				
C	RE	D	(1.000)	10.00	10.00	0.000	0.000
D	BE	H	(1.000)			2.000	1.000
E	BA	H	(1.000)			15.00	1.000
F	BA	G	(1.000)			5.000	5.000
G	BA	H	(1.000)			10.00	1.000
H	BT	I	(1.000)			1.000	1.000
I	BE	END	(1.000)			1.000	0.000

RESOURCE 1 REQUIRES 97.50 UNITS WITH VAR 127.2
RUN AGAIN, TRUE OR FALSE?
?F

STOP

FIGURE 2.13: TYPICAL INTERACTIVE SESSION USING GAP (Continued)

that each node names a next node, except for an or-node which has a series of next nodes each with a given probability. These probabilities must sum to one. When an or-node is encountered, its various branches are traced to their corresponding terminal node. When each branch has been traced, the group of parallel nodes are replaced using formulas 2.3.6 and 2.3.7. Since these can be nested, intermediate values are maintained on a series of stacks.

When a repetition node is encountered values are stored on stacks until the corresponding end node is reached. The equivalent node values are calculated using formulas 2.3.3 and 2.3.4. Since it is possible to nest these in any order, stack pointers are used to keep track of the various loops and branches. Effectively, the program begins from the innermost loops or branches and moves outward until the resource requirement of a single equivalent node is calculated.

2.5 Summary

A graphical method of representing the resource requirement of a computer program has been presented and a methodology to reduce the graph to an equivalent mean value and variance given.

U

53

Chapter 3

WORKLOAD SPECIFICATION.

3.0 Introduction

This chapter examines methods and difficulties in obtaining and predicting CPU workloads in seconds of processing time from lines of code. Specifically, we measure the CPU time required to run FORTRAN77 programs on the Honeywell Level-66 Computer System of Carleton University and seek to relate this to the number of lines of code and other program variables. This is an exploration to understand how workloads might be predicted from estimates of the number of lines of code and other program parameters. We examine a number of programs and seek to find a relationship which can be used in prediction.

Software, hardware and hybrid software-hardware monitors have been developed to measure the resource usage of programs. Plattner and Nevergelt [PLA81] suggests that additional research should be directed to their development. Merrill [MER79] takes a different position and show that accounting data, which most large systems collect, can provide sufficient accuracy for most system analysis. While accounting information does not provide the microinformation that special hardware and software monitors do, it is readily available, is a normal workload of the system and provides sufficient information for this thesis.

3.1 Workloads of FORTRAN77 Programs on a Honeywell Level-66 Computer System

Program CPU workloads are composed of two factors - the time to execute the code of the program and the time required by the operating system to service the program. These can be obtained from the accounting information of the Carleton University Honeywell Level-66 Computer using two commands, STATUS and REPORT.

STATUS provides the cumulative use of resources from logon until it is invoked. REPORT provides the same information for each job step. A job step is one pass from IBEX (Interactive Batch Executive), the primary command and control processor for the CP-6 system, to the user's invoked program and back to IBEX.

Figures 3.1 and 3.2 are examples of the data obtained by the STATUS and REPORT commands. Most of the entries are self-explanatory. PMMES (Privilege Master Mode Entries) is the number of operating system calls.

To estimate the cost of the STATUS command it was repeated sequentially a number of times under varying conditions. Figure 3.3 summarizes the observations.

It was assumed that REPORT provided the net cost of the operation of the different programs. To verify this assumption a series of programs were run using both the REPORT and STATUS commands. As Figure 3.4 illustrates there is a difference which is assumed to be the cost of executing the REPORT command. Thus within an acceptable accuracy, REPORT provides the information desired. An alternative monitoring device was not available to verify the data on STATUS and REPORT.

```

09:06:57 NOV 16 '81 PQ040669, THOMAS          ID= 19683
SUBMITTED 09:05:32 NOV 16 '81
          @ME351
          PRIORITY
          PARTITION
          STEP
          STEP CONDITION CODE
          END STATUS: NORMAL

STARTED 09:05:35 NOV 16 '81
ELAPSED 1:21.98
PERMANENT DISK USED 0
PEAK TEMPORARY DISK 50
PEAK MEMORY 31
RESOURCE MEMORY REQUIRED 31
CPU TIME USED 0:18.44
PROC EXEC TIME 0:05.58 @ 1.2000/min
USER EXEC TIME 0:02.29 @ 1.2000/min
PROC SERVICE TIME 0:09.55 @ 1.2000/min
JOB STEPS 8 @ .0000/step
PMMEs ISSUED 2751 @ .0000/pmme
PROC MEMORY USED 5.25 @ .0400/pg-min
USER MEMORY USED .63 @ .0400/pg-min
TEMP DISK USED 39.16 @ .0005/pg-min
DISK ACCESSES 989 @ .0004/access
FORMS USED

STDRMT LP@ME351 14 @ .0240/form + 0 @ .0000/mnt .34
TOTAL CHARGE = 1.36 1.36
  
```

FIGURE 3.1: EXAMPLE OF A STATUS OUTPUT

09:06:52 NOV 16 '81	PQ040669, THOMAS	@ME351	ID= 19683
SUBMITTED	09:05:32 NOV 16 '81	PRIORITY	7
		PARTITION	2
STARTED	09:06:46 NOV 16 '81	STEP	8
ELAPSED	0:05.89	PROCESSOR(1st)	PCL
PERMANENT DISK USED	0	STEP CONDITION CODE	0
PEAK TEMPORARY DISK	0	END STATUS: NORMAL	
PEAK MEMORY	11		
RESOURCE MEMORY REQUIRED	11		
CPU TIME USED	0:02.11		
PROC EXEC TIME	0:00.36 @ 1.2000/min		.01
PROC SERVICE TIME	0:01.75 @ 1.2000/min		.04
JOB STEPS	1 @ .0000/step		
PMES ISSUED	955 @ .0000/pmme		
PROC MEMORY USED	.61 @ .0400/pg-min		.02
TEMP DISK USED	4.90 @ .0005/pg-min		
DISK ACCESSES	74 @ .0004/access		.03
FORMS USED			
STDRMT	LP@ME351	11 @ .0240/form + 0 @ .0000/mnt	.26
TOTAL CHARGE =			.36

FIGURE 3.2 EXAMPLE OF "REPORT" OUTPUT

	<u>Mean</u>	<u>Standard Deviation</u>
No. of trials	5	
Total number of observations	37	
Total CPU time used per STATUS	.173	.008
Processor execution time per STATUS	.036	.005
Processor service time per STATUS	.137	.005
PMES	37.65	.59
Disk accesses	7.2	.45

FIGURE 3.3: COSTS OF EXECUTING STATUS COMMAND

	<u>Mean</u>	<u>Standard Deviation</u>
No. of trials	5	
Total number of REPORTS	31	
Total CPU time used per REPORT	.441	.023
Processor execution time per REPORT	.037	.007
Processor service time per REPORT	.404	.023
PMES	79.4	1.7
Disk accesses	29.6	2.2

FIGURE 3.4: COST OF EXECUTING REPORT COMMAND

The program workload per job was obtained by executing the program using the REPORT command to obtain the resource requirements.

3.2 Obtaining Workloads from Programs

3.2.1 Some Difficulties

The relationship between workloads and FORTRAN source code lines was investigated. Four methods of characterizing the FORTRAN programs were studied on a series of programs chosen to illustrate some of the difficulties involved. The results are summarized in Figure 3.5.

The first program, P1, consisted of a loop containing real and integer arithmetic operations. The assignment of variables took place within the loop allowing simple optimization. The loop was repeated a large number of times (500,000) to minimize the effect of set-up time, etc.

P2 also consisted of a large number of loops of real and integer arithmetic operations. The assignment of values to variables took place outside the loop and the program was designed to minimize optimization by a compiler. It is based on a benchmark module designed by Curnow and Wichmann [CUR76].

P3 consists of a loop with a series of write statements.

P4 and P5 are identical programs except in P4 the write statement has a single parameter whereas in P5 the write statements have 10 parameters.

Program	Mean Total CPU Time Used	Mean User Execution Time	Mean User Service Time	1	2	3	4	Ratio of Mean User Execution Time and Column			
				FORTRAN Statements No. Weights	FORTRAN Statements Weighted	FORTRAN Statements PM Option Weighted	Object Code Statements	1	2	3	4
P1	6.41	6.25	.06	4.5x10 ⁶	4.5x10 ⁶	9x10 ⁶	9.5x10 ⁶	1.39x10 ⁻⁶	1.39x10 ⁻⁶	6.94x10 ⁻⁷	6.58x10 ⁻⁷
P2	48.76	48.57	.09	6x10 ⁶	6x10 ⁶	47x10 ⁶	40x10 ⁶	8.10x10 ⁻⁶	8.10x10 ⁻⁶	1.03x10 ⁻⁶	1.21x10 ⁻⁶
P3	3.39	2.35	.96	1892	12,603	7206	2.858x10 ⁶	1.3x10 ⁻³	1.86x10 ⁻⁴	3.26x10 ⁻⁴	8.22x10 ⁻⁷
P4	31.60	31.21	.19	1,68x10 ⁶	8.642x10 ⁶	36.216x10 ⁶	38.488x10 ⁶	1.8x10 ⁻⁵	3.61x10 ⁻⁶	8.62x10 ⁻⁷	8.11x10 ⁻⁷
P5	6.03	4.99	.94	1013	8513	13015	5.882x10 ⁴	4.92x10 ⁻³	5.8x10 ⁻⁴	3.83x10 ⁻⁴	8.48x10 ⁻⁷
P6	2.35	1.45	.81	1013	8513	5515	1.7092x10 ⁶	1.42x10 ⁻³	1.70x10 ⁻⁴	2.63x10 ⁻⁴	8.50x10 ⁻⁷

FIGURE 3.5: LINES OF CODE VERSUS RESOURCES USED FOR A NUMBER OF FORTRAN PROGRAMS

P6 consists of a loop repeated many times which calls subroutines which in turn use the FORTRAN intrinsic functions SIN and SQRT.

The four methods of characterizing the FORTRAN programs and their relative merit are discussed below.

1. Unweighted FORTRAN lines of code. This is a crude method which produced very inconsistent results. Each line of executable FORTRAN code is given equal weight. There is a great difference between the service time of a write statement and an assignment statement. In FORTRAN77 it is also possible to place several assignment statements on a single line. Thus two programs which only differed in this one detail could be judged as being different.

2. Weighted lines of code using the PM option on CP-6. The PM option on CP-6 assigns weights to each line of code. Figure 3.6 shows a typical output. As it presently exists it does not provide sufficient improvement over other methods to justify its use. There are two weaknesses. First, the analysis which it performs fails to recognize calls to subroutines and intrinsic FORTRAN functions. This could be improved and more accurate weighting obtained. The second weakness is that it requires that the program be assembled. This could also be modified and such an analysis of code could be a valuable addition to the tools available to the programmer and software engineer.

3. Weighted FORTRAN lines of code. This is a variation of 2. If the resource requirement of a proposed program is to be estimated, then several estimates could be made - the number of statements, the number of reads, the number of writes, the number of calls and others as

FORTRAN 77 VERSION B01 SOURCE=ARITH

COMPILE UNIT 001 SEP 14 '81 09:19:50.50 4

**** EXECUTABLE STATEMENTS ****

EXEC COST	STMT	SEC	OCTLOC	STTYPE	LB LABEL	EXEC COST	STMT	SEC	OCTLOC	STTYPE	LB LABEL
	1	0	0	PROC	0		5	0	4	ASSIGN	0
	6	0	20	DO	0	\$\$\$	7	0	24	ASSIGN	0
\$\$\$\$	8	0	34	ASSIGN	0	\$\$\$\$	9	0	47	ASSIGN	0
	10	0	61	ASSIGN	0	\$\$\$\$	11	0	66	ASSIGN	0
\$\$\$\$	12	0	73	ASSIGN	0	\$\$\$\$	13	0	103	ASSIGN	0
\$\$\$\$	14	0	115	ASSIGN	0	\$\$\$\$	15	0	127	ASSIGN	0
\$\$\$	16	0	134	ASSIGN	0		17	0	141	NULL	0
	18	0	144	ENDPRC	0						10S

**** STATEMENT TYPE TOTALS ****

1 NULL	11 ASSIGN	1 DO	1 PROC	1 ENDPRC	1 LABELS	
# MAJOR STMTS=	11	CODE/MAJOR= 9.63	# STMTS=	15	CODE/STMT= 7.06	POINT TOTAL= 84.3
COST/MAJOR STMT=	10.1	COST/STMT= 7.4	TOTAL COST=	111.1		

0-4 \$
4-6 \$
6-8 \$
8-10 \$
10-12 \$
12-14 \$
14-16 \$
16-18 \$
18-20 \$
20-22 \$
22-?? \$
ERRORS FOUND: 0
ERR SEVERITY LEVEL: 0

FIGURE 3.6: TYPICAL OUTPUT OF PM OPTION ON CP-6

experience dictates. This weighted value could be used to obtain the estimated resource requirement if the CPU time of each element is known.

4. OBJECT code. The use of the object code produced the most consistent results. However, as a means of predicting resource requirements it is not usually available. If one knew the code that would be generated by the compiler it would be possible to perform an exact analysis of the expected resource requirements. If the hardware existed the program could be run and monitored. If existing modules are to be transferred to new hardware then there are situations in which such detailed analysis might be of value.

The workload of a CALL statement with different numbers of parameters was obtained by running a basic program in which the number of parameters was varied. Figure 3.7 illustrates the result. As the number of parameters increased the time required to complete the switch increased. The values in Figure 3.7 are for a particular case and while the effect of the number of parameters is general, the actual values will change.

A similar effect occurs with the number of elements included in a WRITE statement. Figures 3.8 and 3.9 illustrate the workloads for different numbers of parameters. These results were obtained by varying the number of WRITE parameters in a program run a number of times.

Thus, there is a problem in specifying a line of code in FORTRAN or even to specify what workload a single operation such as a WRITE or

FIGURE 3.7: TIME TO EXECUTE A FORTRAN CALL STATEMENT VERSUS
NUMBER OF PARAMETERS

FIGURE 3.8: . USER EXECUTION TIME .VERSUS NUMBER OF WRITES

FIGURE 3.9: USER EXECUTION TIME VERSUS NO. OF PARAMETERS OF A
FORTRAN WRITE STATEMENT

CALL imposes on the CPU. A detailed analysis of the object code generated by the compiler can avoid this, however, at that point the program may be run and its workload directly measured. In the next section it is seen that a statistical average may be obtained.

3.2.2 Resource Requirements of Some FORTRAN Programs

In a given situation it may be possible to analyze a number of existing programs to obtain a relationship between lines of code and the resultant workload. Ideally, this would involve a parser which would automatically produce a list of all the FORTRAN codes of interest and give a weight to each one - an improved version of the PM option mentioned in the previous section. To investigate this concept a FORTRAN77 program was written which inserted CALL statements into existing programs so that the number of statements, READS, WRITES, SUBROUTINE calls would be counted to see the relationship to the resource requirements. Figure 3.10 records the results obtained. It was theorized that a linear relationship existed between the parameters such that

$$\text{TOTAL CPU TIME} = K_1 + K_2 * (S) + K_3 * (R) + K_4 * (W) + K_5 * (C) \quad (3.1)$$

where

K_1, K_2, K_3, K_4, K_5 are constants for a given set of programs

S = number of executable FORTRAN lines of code

R = number of READ or INPUT statements

W = number of WRITE, PRINT or OUTPUT statements

C = number of CALL statements

Program	No. of Statements	READS	WRITES	No. of CALLS	CPU		PET	PST	UST	DA	READS & WRITES
					Total	UET					
P1	111324	-	420	-	4.04	3.32	-	.11	.61	38	420
P2	85434	-	155	-	1.64	1.14	-	.13	.37	33	155
P3	960100	-	48	-	21.21	20.90	-	.13	.18	25	48
P4	1100004	-	-	-	53.12	52.79	-	.13	.20	25	
P5	7509	-	120	-	1.06	.70	-	.11	.25	31	320
P6	6826	14	115	17	1.33	.71	-	.27	.35	57	129
P7	13646	28	230	34	2.15	1.39	-	.25	.51	61	258
P8	29940	28	318	34	3.02	2.16	-	.24	.62	65	246
P9	60161	40	70	8	1.30	.78	-	.19	.33	44	110
P10	4231690	54	209	11	27.36	26.56	-	.20	.60	51	263
P11	200020	48	224	9	3.08	2.28	-	.21	.59	53	272
P12	423856	74	479	14	6.03	4.08	-	.18	1.05	54	553
P13	13646	28	230	34	2.18	1.39	-	.26	.53	61	255
P14	4154	102	124	22	1.47	.70	-	.20	.57	54	226
P15	2079	51	62	11	.92	.35	-	.20	.37	50	113
P16	2095	48	58	8	.91	.35	-	.20	.36	53	106
P17	4186	96	116	16	1.43	.69	-	.21	.53	54	212
P18	85409	40	70	8	1.41	.92	-	.18	.37	44	170
P19	3356290	53	208	11	22.00	21.23	-	.19	.58	51	261
P20	266540	48	224	9	3.40	2.63	-	.20	.57	53	272
P21	520069	73	478	14	6.34	5.26	-	.20	.99	54	551

FIGURE 3.10: SUMMARY OF EXECUTION AND SERVICE TIMES VERSUS NUMBER OF STATEMENTS FOR SOME FORTRAN PROGRAMS

A least square linear regression was attempted on the data of Figure 3.10. It was found that P3 and P4 were significantly different (more than 3 standard deviations from the expected value). The reason is that P3 involves the use of many intrinsic FORTRAN functions. P4 involves complicated arithmetic multiplications and divisions and in structure is significantly different to the other programs. When these programs are removed from consideration the following results were obtained:

		$T = K_i / \text{Std. Error}$
$K_1 = .375$	Std. Error = .084	= 4.43
$K_2 = 6.03 \times 10^{-6}$	= .030 $\times 10^{-6}$	198
$K_3 = 2.70 \times 10^{-4}$	= 12.3 $\times 10^{-4}$	.219
$K_4 = 62.7 \times 10^{-4}$	= 2.64 $\times 10^{-4}$	23.7
$K_5 = 96.4 \times 10^{-4}$	= 34.4 $\times 10^{-4}$	2.82

The T ratio suggests that only K_2 is stable. This was confirmed by considering subgroups. More observations would improve the results.

When an equation of the form

$$\text{TOTAL CPU TIME} = K_1 + K_2 S \quad (3.2)$$

is considered, the following results were obtained:

$K_1 = 1.78$	Std. Error = .22	$T = 7.98$
$K_2 = 6.09 \times 10^{-6}$	= .18 $\times 10^{-6}$	$T = 34.2$

Figure 3.11 compares the actual TOTAL CPU TIME to the fitted times using equations 3.1 and 3.2. The more parameters provide more consistent results

and even in the worst case, the time of the fitted curve is only out by .22 seconds in 1.64 seconds or 13%. Three other equations of the form:

$$\text{TOTAL CPU TIME} = K_1 + K_2 * S + K_3 (R+W) \quad (3.3)$$

$$\text{USER EXECUTION TIME} = K_4 + K_5 * S + K_6 (R+W) \quad (3.4)$$

$$\text{USER SERVICE TIME} = K_7 + K_8 * S + K_9 (R+W) \quad (3.5)$$

were fitted to the data. The following values for K_i were obtained:

$$K_1 = .328 \quad \text{Std. Error} = .173 \quad T = 1.9$$

$$K_2 = 5.99 \times 10^{-6} \quad \text{Std. Error} = 7.04 \times 10^{-8} \quad T = 85.03$$

$$K_3 = 6.06 \times 10^{-3} \quad \text{Std. Error} = .623 \times 10^{-3} \quad T = 9.7$$

$$K_4 = -.036 \quad \text{Std. Error} = .175 \quad T = -.203$$

$$K_5 = 5.98 \times 10^{-6} \quad \text{Std. Error} = .071 \quad T = 83.6$$

$$K_6 = 4.62 \times 10^{-3} \quad \text{Std. Error} = .633 \times 10^{-3} \quad T = 7.3$$

$$K_7 = .161 \quad \text{Std. Error} = .034 \quad T = 4.7$$

$$K_8 = 1.19 \times 10^{-8} \quad \text{Std. Error} = 1.39 \times 10^{-8} \quad T = .854$$

$$K_9 = 1.47 \times 10^{-3} \quad \text{Std. Error} = .12 \times 10^{-3} \quad T = 11.9$$

Equation 3.3 with rounded values of $K_1=.3$, $K_2=6 \times 10^{-6}$ and $K_3=6 \times 10^{-3}$

is used on page 66.

3.2.3 Predicting Using the Methods of Chapter 2

Four programs were analyzed using the techniques outlined in Chapter 2. The input data was known. The number of statements and the number of read/writes were estimated from the FORTRAN code. Then using these values

Program	Measure Total CPU Time	Equation 3.1		Equation 3.2		Equation 3.3	
		Fitted	Residuals	Fitted	Residuals	Fitted	Residuals
P1	4.04	3.68	.36	2.46	1.58	3.54	.50
P2	1.64	1.86	-.22	2.30	-.66	1.78	-.14
P5	1.06	1.17	-.11	1.82	-.76	1.10	-.04
P6	1.33	1.31	.02	1.82	-.49	1.15	.18
P7	2.15	2.24	-.09	1.86	.29	1.97	.18
P8	3.02	2.89	.13	1.96	1.06	2.00	1.02
P9	1.30	1.26	.04	2.14	-.84	1.36	-.06
P10	27.36	27.31	.05	27.54	-.19	27.28	.08
P11	3.08	3.09	-.01	3.00	.08	3.18	-.10
P12	6.03	6.09	-.06	4.36	1.67	6.22	-.19
P13	2.18	2.24	-.06	1.86	.32	1.97	.21
P14	1.47	1.42	.05	1.80	-.33	1.72	-.25
P15	.92	.90	.02	1.79	-.87	1.03	-.11
P16	.91	.84	.07	1.79	-.88	.98	-.07
P17	1.43	1.31	.12	1.80	-.37	1.64	-.21
P18	1.41	1.42	-.01	2.30	-.89	1.87	-.46
P19	22.00	22.03	-.02	22.22	-.22	22.02	-.02
P20	3.40	3.48	-.08	3.40	.00	3.57	-.17
P21	6.45	6.66	-.21	4.94	1.50	6.79	-.34
Standard Deviation		.147		.891		.349	

FIGURE 3.11: MEASURED TOTAL CPU TIMES VERSUS FITTED TIMES

and equations (3.3) and (3.4) and the constants of the previous section, the TOTAL CPU time and User Execution Time were estimated. The values obtained and the measured values are compared in Figure 3.12. On these relatively small programs there are errors of 25%. The analyses of the number of lines of code was carefully done. On the other hand, there was no attempt to weigh the statement. The programs were similar to those which formed the basis of Figure 3.11. This indicates that errors of 25% can be expected in prediction even when the source code and input data is known. However, better accuracy can be expected if the lines of FORTRAN are given weights.

3.2.4 Data Dependency and Interdependence

The results predicted in the previous section assumed that the input data was deterministically known. If this is not the case the methods of Chapter 2 permit a mean and variance to be used. In the case of scientific programs the variance may be large.

In scientific programs the distribution of the input may be difficult to know a priori. And even when the input data is known, the form of the program may involve iterations whose number is only a poorly understood function of the input. For example, in many simulation programs, the program is run until a desired stability is reached. One of the purposes of the program might be to investigate this rate of convergence to an acceptable answer. These problems do not invalidate the method. They do point to the need to use the methods with wisdom and understanding.

Program	Predicted Number FORTRAN Instruction	Read Writes	Measured Total CPU Time	Predicted Total CPU Time	Error Time	% Error	Measured User Execution Time	Predicted User Execution Time	Error Time	% Error
P1	1547	66	.82	.74	.08	10%	.34	.28	.06	18%
P2	9.393×10^4	127	1.91	1.66	.25	13%	1.13	1.11	.02	2%
P3	3.581×10^4	232	2.57	1.95	.62	24%	1.69	1.25	.44	26%
P4	2921	340	2.42	2.41	.01	.4%	1.35	1.55	-.20	15%

FIGURE 3.12: ANALYSIS OF SOME FORTRAN PROGRAMS

To illustrate how the method can handle data dependency let us consider a simple example. Suppose that there are two nested loops as illustrated in Figure 3.13. The outer loop can be executed 1 or 3 times with equal probability. The inner loop can also independently be executed 1 or 3 times. The resource requirement of each loop is 10 resource units.

Applying equations (2.3.3) and (2.3.4) successively it is evident that the equivalent single node has an expected time of 40 units with a standard deviation of 24.5. There is a wide spread in total time.

A more serious problem is when the mean and variance of the data elements are not independent. The calculated results applying formulas (2.3.3) and (2.3.4) could be wrong. In the above example if N_1 and N_2 are the same variable, then the possible outcomes are:

$$\begin{array}{lll} N_1 = 1 & N_2 = 1 & \text{TOTAL RESOURCE REQUIRED} = 10 \\ N_1 = 3 & N_2 = 3 & \text{TOTAL RESOURCE REQUIRED} = 90 \end{array}$$

$$\begin{aligned} \text{EX}(T) &= [10 + 90] * 1/2 = 50 \\ \text{VARX}(T) &= 1/2 [40^2 + 40^2] = 1600 \end{aligned}$$

In this example the two loops can be amalgamated to one loop which is executed either 1 or 9 times. The mean number of executions is now 5 with a variance of 16. Equation (2.3.3) and (2.3.4) now give the correct results

$$\begin{array}{ll} \text{EX}(T) = \text{EN} * \text{EX}(i) & \text{VART} = \text{EN} * \text{VARX}(i) + \text{VARN} * (\text{EX}(i))^2 \\ = 5 * 10 & = 5 * 0 + 16 * (10)^2 \\ = 50 & = 1600 \end{array}$$

FIGURE 3.13: EXAMPLE OF TWO NESTED LOOPS

Possibilities		Total Time
$N_1 = 1$	$N_2 = 1$	10
	3	30
3	$N_2 = 1,1,1$	30
	1,1,3	0
	1,3,1	50
	3,1,1	50
	1,3,3	70
	3,1,3	70
	3,3,1	70
	3,3,3	90

$$EX(T) = \sum P_i T$$

$$= \frac{1}{4}(10+30) + \frac{1}{16}(30+3+50+3+70+90)$$

$$= 40$$

$$VARX(T) = \sum P_i [T-ET]^2$$

$$= \frac{1}{4} [30^2 + 10^2] + \frac{1}{16} [10^2 + 3 \cdot 10^2 + 3 \cdot 30^2 + 50^2]$$

$$= 600$$

Apply Formula

$$1. \quad EN_1 = 2 \quad VARN_1 = 1$$

$$EN_2 = 2 \quad VARN_2 = 1$$

2. Inner Loop

$$EX(2) = EN_2 \cdot EX(i)$$

$$= 2 \cdot 10 = 20$$

$$VARX(2) = EN \cdot VARX(i) + VARN \cdot (EX(i))^2$$

$$= 2 \cdot 0 + 1 \cdot 10^2$$

$$= 100$$

3. Outer Loop

$$EX(T) = EN_1 \cdot EX(i)$$

$$= 2 \cdot 20 = 40$$

$$VARX(T) = EN_1 \cdot VARX(i) + VARN_1 \cdot (EX(i))^2$$

This suggests the importance in formulating the graphical representation of a program of maintaining the independence assumption.

3.2.5 Summary of Findings

This case study on FORTRAN77 program resource requirements shows that there is uncertainty in workload predictions. This uncertainty depends upon:

1. How accurately the number of lines of code is known. At an early stage in the software development the errors may be substantial.
2. How accurately the mapping function from the FORTRAN to resource requirements is known and applied. If each FORTRAN line is known in terms of machine code, as was approached in studying the object code, high accuracy (1-10%) can be obtained. On the other hand, a crude count of FORTRAN lines can result in errors of 25% or more.
3. How accurately the input data of the program is known for data dependent execution.
4. How well the independence assumption is maintained in the model. In the previous section we saw a modelling error which could be introduced.

3.3 The Propagation of Errors in the Workload Model

If a program is developed in module fashion it may be that the resource requirements of a number of modules are known by measurement and there are only one or more modules to program. Or it may be that it

is desired to investigate the change in the resource requirement of one or two modules. Figure 3.14 illustrates a situation in which the resource requirement $x(i,r)$ of each module is known and it is desired to ascertain the effect of an error or change in $x(D,r)$ of $\Delta x(D,r)$. If it is only the mean value which is of interest the effective change in the total resource requirement for the graph $\Delta z(j,r)$ is

$$\Delta z(j,r) = y(D,r) * \Delta x(D,r)$$

where $y(D,r)$ is the weight of module D and may be obtained by considering a unit response requirement at D. In the example illustrated in Figure 3.14

$$y(D,r) = N * (1-p)$$

$$\therefore \Delta z(j,r) = N * (1-p) * \Delta x(D,r)$$

If there are several modules which are in error or changed, the total effect is the linear addition of the factors plus any cross terms. Thus, in the example if there is also a change or error in module C of $\Delta x(C,r)$ then

$$\Delta z(j,r) = N * (1-p) * \Delta x(D,r) + N * P * \Delta x(C,r)$$

If N is in error ΔN and module C in error $\Delta x(C,r)$ the error in the equivalent workload

$$\Delta z(j,r) = \Delta N * [x(A,r) + x(B,r) + P * x(C,r) + (1-p) * x(D,r) + x(E,r) + x(F,r)] + N * P * \Delta x(C,r) + \Delta N * P * \Delta x(C,r)$$

FIGURE 3.14: GRAPH OF A SIMPLE PROGRAM

Obviously, for more than a few errors it is simpler to redo the calculation. If the distribution or the variances of values are known these can be used in the methodology outlined in Chapter 2.

3.4 Summary

In this chapter the focus has been on problems and difficulties in implementing the methodology outlined in Chapter 2. It was shown that accuracies of 25% can be obtained in obtaining CPU times from FORTRAN source code by assigning weights to different statements. Higher accuracies can be obtained using more detailed weightings.

No attempt has been made to estimate how accurately the number of lines of FORTRAN77 code can be made prior to the coding of a problem. The error made in these estimates will make the predicted resource requirements less accurate. Top down design of software systems will allow more accurate estimates to be obtained as individual modules are completed. If the target system is available resource requirements could be obtained by compilation of modules and direct measurements.

At the initial stages, there can be substantial errors in the estimates of resource requirements. The effect of these on performance parameters is the subject of the following chapters.

Chapter 4

MODELS TO OBTAIN PERFORMANCE PARAMETERS FROM WORKLOADS

4.0 Introduction

Figure 4.1 represents a computer system consisting of N active terminals, a central processing unit and two disk units (DK1 and DK2). Conceptually, a token representing a job is envisioned moving among the devices. There are N tokens corresponding to the N active terminals. The branching probabilities and quantity of work required at each service center is found by knowing the workloads at each center and the average number of times it must be visited. The CPU workload per job is found using the methods of the previous chapters. The number of disk accesses are found by analyzing how data and programs are stored and the access patterns.

One of the major advances during the 1970's in obtaining performance parameters has been the application of closed network queueing theory. Coupled with this has been the development of efficient algorithms to analysis networks to which the product form solution applies [GEL80, REI80]. The product form solution applies to four service disciplines - Infinite Server (IS), Processing Sharing (PS), First Come First Served (FCFS) and Last Come First Served Preemptive Resume (LCFSPR). Infinite Server is often used to model a set of terminals. Processor Sharing is an approximation to Round Robin introduced by Kleinrock (KLE76). Jobs enter service immediately on arrival, each job receiving $1/K$ of the service time,

Service Discipline Class 1				
TER	Terminals	Infinite Server	$x(\text{TER},1)=14 \text{ sec}$	$y(\text{TER},1)=1$
CPU	Central Processor	Processor Sharing	$x(\text{CPU},1)=.4$	$y(\text{CPU},1)=10$
DK1	Disk 1	FCFS	$x(\text{DK1},1)=.6$	$y(\text{DK1},1)=4$
DK2	Disk 2	FCFS	$x(\text{DK2},1)=.4$	$y(\text{DK1},1)=6$

FIGURE 4.1: MODEL OF COMPUTER SYSTEM

where K is the number of jobs or tokens at the service center. The service time at a FCFS must be exponentially distributed and independent of job class, though it can be dependent on queue length.

In the next chapter we will examine the propagation of workload errors by examining the asymptotic constraints to the closed queueing model and a modification of the mean value algorithm. These are now presented.

4.1 Single Class Bottleneck Model

Often it is sufficient to obtain bounds on the solutions. [BOW81] One such bound is provided by the Bottleneck Model. Two asymptotic constraints to the closed queueing model of a computer system are obtained by considering a very light loaded system and a heavily loaded system. The minimum time that it can take a job to go through a series of servers is the sum of the service times at all the devices it visits.

$$C(r) \geq \sum_{i=1}^M y(i,r) x(i,r) \quad (4.1)$$

where

$C(r)$ = cycle time of a class r job

$x(i,r)$ = mean service time at device i per visit

$y(i,r)$ = number of visits to device i per interaction

M = number of devices in system

If there are $N(r)$ jobs in the system and applying Little's Formula

$f(r)$ = throughput of class r jobs

$$= \frac{N(r)}{C(r)} \quad (4.2)$$

$$f(r) \leq \frac{N(r)}{\sum_{i=1}^m y(i,r) x(i,r)} \quad (4.3)$$

The equal signs of equations (4.1 and (4.2) apply for a single job ($N(r)=1$) in a computer system. It applies approximately at low utilizations. ($N(r) \ll N * (r)$ - the bottleneck number, see page 64).

The second constraint is at high utilization. Jobs cannot go through the system faster than they are served at the slowest device. This may be expressed as a utilization constraint

$$f(i,r) x(i,r) \leq U_{\max}(i,r) \text{ for } i = 1, \dots, m \quad (4.4)$$

$$f(r) y(i,r) x(i,r) \leq U_{\max}(i,r) \text{ for } i = 1, \dots, m \quad (4.5)$$

where $U_{\max}(i,r)$ is the maximum allowed utilization of device i , usually this is $s(i)$ where $s(i)$ is the maximum number of servers at device i

$$U_{\max}(i,r) \leq s(i) \quad (4.6)$$

$f(i,r)$ is the visit rate of class r jobs to device i . Recall that there are only a single class r in the system

$$\therefore f(r) \leq \frac{1}{\max_i [y(i,r) x(i,r)]} \quad (4.7)$$

$$C(r) = \frac{N(r)}{f(r)}$$

$$C(r) \geq N(r) \max_i [y(i,r) x(i,r)] \quad (4.8)$$

FIGURE 4.2: TYPICAL SINGLE CLASS BOTTLENECK CONSTRAINTS

The number of jobs at which the constraint represented by equations (4.6) and (4.1) intersect is termed the saturation number, or bottleneck number and is represented by an asterisk $*$ - N^* .

Figure 4.2 illustrates a plot of cycle time versus number in system for the system illustrated in Figure 4.1. The bottleneck constraint curve is shown by solid lines, an exact analysis is shown by dotted lines for comparison. Note that the constraint lines apply for any service discipline whereas the exact analysis presumes the service scheduling shown.

4.2 Single Class Mean Value Algorithm

The mean value algorithm developed by Reiser and Lavenberg, through the clever application of Little's Formula, allows the solution of closed queueing models that have the product form. The basic approach is to recursively consider a job circulating through a system with one less job present.

Variables:

$f(i,k)$ - throughput at device i with k jobs in system

i - number of device $i = 1, \dots, M$

k - number of jobs in the system

M - total number of devices

$n(i,k)$ - expected number of jobs at i with k jobs in system

N - maximum number of jobs in systems

$S(i)$ - service discipline at i

$U(i,k)$ - utilization of i with k jobs in system

$w(i,k)$ - expected time at i with k jobs in system
 $x(i)$ - service time at i
 $y(i)$ - number of visits to i per visit to reference device

BEGIN

READ $y(i)$, $x(i)$, $S(i)$ for $i = 1, \dots, M$

$n(i,0) := 0$ for $i = 1, \dots, M$

FOR $K = 1$ to N DO

BEGIN

FOR $i = 1$ to M DO

IF ($S(i) = \text{FCFS}$) ; FCFC = FIRST COME FIRST SERVED

$w(i,k) := x(i) [1 + n(i,k-1)]$

ELSE IF ($S(i) = \text{IS}$) ; IS = Infinite Server

$w(i,k) := x(i)$;

ENDIF

ENDFOR

$f(i,k) = k / \sum_{i=1}^M y(i) w(i,k)$

$f(i,k) = y(i) f(i,k)$ for $i = 2, \dots, M$

$U(i,k) = f(i,k) x(i,k)$ for $i = 1, \dots, M$

$n(i,k) = f(i,k) w(i,k)$ for $i = 1, \dots, M$

ENDFOR

END (MEAN VALUE ALGORITHM)

The algorithm is easily extended to the multiclass situation. Bruell and Balbo [BRU80] present details of the algorithm and discuss its implementation on a digital computer.

4.3 An Illustrative Example

We will consider a hypothetical example to illustrate the method. Consider an existing computer system consisting of a CPU, a set of terminals, and two disks. In Chapter 3, page 49 it was found that the CPU workload including operating system overhead could be approximated by an equation of the form

$$\begin{aligned} \text{TOTAL CPU TIME} = & 0.3 + 6 \times 10^{-6} * (\text{number of lines of code}) \\ & + 6 \times 10^{-3} * (\text{number of read and write statements}) \end{aligned}$$

The computer can be modelled as a central server model with terminals with a 15 second service time and two disk units each with a mean service time of 0.01 seconds per access.

We want to know how the system will behave if a varying number of identical programs are added. It has been estimated that each program will have

53,000 lines of code per execution

30 read-write statements per execution

9 accesses per execution to disk 1

10 accesses per execution to disk 2

The total CPU service time

$$\begin{aligned} &= 0.3 + 6 \times 10^{-6} (53,000) + 6 \times 10^{-3} (30) \\ &= 0.80 \text{ seconds per execution} \end{aligned}$$

The asymptotes of section 4.1 are

(a) for a lightly loaded system

$$\begin{aligned} \text{Cycle Time} = C(r) &\geq \sum_{i=1}^M y(i,r) x(i,r) \\ &\geq 1 * 15 + 1 * .8 + 9 * .1 + 10 * .1 \\ &\geq 15.99 \text{ seconds} \end{aligned}$$

$$\therefore \text{Response Time} \geq .99 \text{ seconds}$$

$$\begin{aligned} \text{Throughput} = f(r) &= \frac{N(r)}{C(r)} \\ &= \frac{N(r)}{15.99} \\ &= .0625 * N(r) \frac{\text{jobs}}{\text{seconds}} \end{aligned}$$

(b) For a heavily loaded system

$$\begin{aligned} \text{Throughput} = f(r) &= \frac{1}{\max [y(i,r) x(i,r)]} \\ &= \frac{1}{.8} = 1.25 \text{ jobs/sec.} \end{aligned}$$

$$\text{Cycle Time} = 0.8 N(r)$$

$$\text{Response Time} = 0.8 N(r) - 15$$

Using the mean value algorithm of section 4.2, the following values are obtained

<u>Number of Jobs</u>	<u>Expected Response Time</u>	<u>Expected Throughput</u>
1	0.99 seconds	0.0625 jobs/sec.
5	1.18 seconds	0.3090
10	1.55	.6042
15	2.22	.8712
20	3.48	1.0823
25	5.76	1.2043
30	9.12	1.2438
35	13.0089	1.2996
40	17.0003	1.2500

On the basis of these results decisions can be made concerning the suitability of the existing system to handle this load. For example,

if 40 of these jobs are to be run concurrently and an average response time of 17 seconds is not acceptable, then either modification to the existing computer system must be made or the proposed program carefully examined to ascertain if it can be modified to use less CPU time.

The effect of an error in the estimates are the subject of the next chapter. In section 5.4 the effect of errors in the parameters of this example are considered.

4.4 Summary

Methods of obtaining throughput, response times and utilization bounds using the bottleneck constraint method were summarized. An outline of the mean value algorithm developed by Lavenberg and Reiser was presented. An example was considered which illustrated how these could be applied as part of the predicting process. In the next chapter the effect of errors on the constraints and the values obtained by the mean value algorithm are presented.

Chapter 5

THE PROPAGATION OF WORKLOAD ERRORS IN PERFORMANCE MODELS

Introduction

In Chapter 3, we investigated the accuracy of obtaining computer workloads from program source code. It was seen that there could be errors and uncertainty in these workloads. This would be especially true when the workloads are based on estimates made in the early stages of software development.

In this chapter, using the modelling techniques outlined in the previous chapter, we examine the effect of these errors on performance characteristics. It is shown that errors in the workload estimates of resources with low utilization may have a minor effect if another resource of the computer system is highly utilized. On the other hand errors in the bottleneck resource of a heavily loaded system causes substantial changes in the response times.

The knowledge of the relative importance of the accuracy of the workloads for various devices can guide the analyst to direct greater design efforts to the more critical elements. Or conversely less effort may be directed to those elements which have been identified by the techniques of this chapter as being less sensitive to errors.

In the next section we look at the effect of workload errors on the bottleneck constraints. These are useful as a vehicle to understand

the propagation of a workload error. They can also be used to alert the designer to problems at an early stage of software development when it is relatively inexpensive to make basic design changes. We examine a few simple cases for which exact solutions are known. We then present a modification of the single class mean value algorithm presented in the previous chapter which calculates the rate of change of the response time with respect to a change in the service time. The effect of an error can be estimated using this rate of change

$$\Delta T(r) = \frac{\delta T(r)}{\delta x(i)} \Delta x(i) \quad (5.1)$$

where $\Delta T(r)$ - change in response time $T(r)$

$\Delta x(i)$ - small change in service time of device i , $x(i)$.

In section 5.4 we present several examples which compare the results obtained using the bottleneck error constraints and the modified mean value algorithm.

5.1 Effect of Workload Errors Using Bottleneck Constraints

In Chapter 4, the bottleneck constraint model was introduced as a first approximation to analyze the performance of a computer system. Now it will be used to examine the effect of an error in estimating resource requirements upon performance parameters, in particular throughput and response time. We focus on closed systems with a single job class.

The bottleneck model is accurate only for the asymptotic cases of

a very lightly or heavily loaded system. We examine these two cases. The case of a heavily loaded system is further divided as to whether the error is in the bottlenecked device or not.

5.1.1. Case I - Lightly Loaded System

A lightly loaded system is one with little contention between jobs. The number of jobs $N(r)$, is much less than the saturation number $N^*(r)$. Each job enters service immediately at every device. Equations (4.1), (4.2), and (4.3) apply. In addition

$$T(r) = C(r) - TH(r) \quad (5.1)$$

where $T(r)$ = response time of class r

$TH(r)$ = think time or service time at the reference device

The rates of change of the response time and throughput due to a change in the service times may be found by differentiation. Recall that the reference device, normally the terminal, is the device from which the response time is measured

$$\begin{aligned} \frac{\delta T(r)}{\delta x(j,r)} &= y(j,r) && \text{if } j \text{ is not the reference device} \\ &= 0 && \text{if } j \text{ is the reference device} \end{aligned} \quad (5.2)$$

$$\begin{aligned} \frac{\delta f(r)}{\delta x(j,r)} &= - \frac{N(r)}{C^2(r)} y(j,r) \text{ for } j=1, \dots, M \\ &= - \frac{f(r)}{C(r)} y(j,r) \end{aligned} \quad (5.3)$$

The resultant errors in the response time $\Delta T(r)$ or the throughput $\Delta f(r)$ due to an error $\Delta x(j,r)$ in the service time of device j can be found by taking differences or approximately from the above ratio.

$$\begin{aligned} \Delta T(r) &= y(i,r) \Delta x(j,r) \text{ if } j \text{ is not the reference device} \\ &= 0 \text{ if } j \text{ is the reference device} \end{aligned} \quad (5.4)$$

$$\Delta f(r) = - \frac{N(r) y(j,r) \Delta x(j,r)}{\left(\sum_{i=1}^M y(i,r) x(i,r) \right) \left(\sum_{i=1}^M y(i,r) x(i,r) + y(j,r) \Delta x(j,r) \right)} \quad (5.5)$$

$$\approx - \frac{N(r) y(j,r) \Delta x(j,r)}{c^2(r)} \quad (5.6)$$

5.1.2 Case II - Heavily Loaded System - Error in Non-Bottlenecked Device

A heavily loaded system is one in which $N(r) \gg N^*(r)$. The error occurs in a resource which is not the bottlenecked resource. It is assumed that the error does not cause a change in the bottlenecked resource. Equations (4.7) and (4.8) of Chapter 4 apply

$$f(r) = \frac{1}{y(b,r) x(b,r)} \text{ where } b \text{ refers to the bottlenecked device}$$

Since $f(r)$ is not a function of $x(i,r)$ $i \neq b$, an error in $x(i,r)$ $i \neq b$ has no effect on the throughput. The response time

$$\begin{aligned} T(r) &= C(r) - TH(r) = \frac{N(r)}{f(r)} - TH(r) \\ &= N(r) y(b,r) x(b,r) - TH(r) \end{aligned} \quad (5.7)$$

is also independent of an error in the workload at a non-bottlenecked

device unless that device is the reference device

$$\text{i.e. } \frac{\delta f(r)}{\delta x(i,r)} = 0 \quad i \neq b \quad (5.8)$$

$$\frac{\delta T(r)}{\delta x(i,r)} = 0 \quad i \neq b \quad (5.9)$$

5.1.3 Case III - Heavily Loaded System - Error in Bottlenecked Device

Again $N(r) \gg N^*(r)$ and it is assumed that the error does not cause a change in the bottlenecked device

$$\begin{aligned} \frac{\delta f(r)}{\delta x(b,r)} &= - \frac{1}{y(b,r) x^2(b,r)} \\ &= - \frac{f(r)}{x(b,r)} \end{aligned} \quad (5.10)$$

$$\frac{\delta T(r)}{\delta x(b,r)} = N(r) y(b,r) \quad (5.11)$$

The actual and approximate errors in the performance parameters due to an error $\Delta x(b,r)$, in the resource requirement at the bottlenecked device are

$$\begin{aligned} \Delta f(r) &= \frac{\Delta x(b,r)}{y(b,r) x(b,r) [x(b,r) + x(b,r)]} \\ &\approx - \frac{f(r)}{x(b,r)} \Delta x(b,r) \end{aligned} \quad (5.12)$$

$$\Delta T(r) = N(r) y(b,r) \Delta x(b,r) \quad (5.13)$$

Equation (5.13) illustrates the multiplicative effect of an error in the service time of the bottleneck device. If there are 40 jobs, each job visits the bottlenecked device 5 times, and an error of .1 seconds is made in the resource requirement of each job, the resultant error in the response time is 20 seconds, 200 times greater.

5.1.4 Case IV - An Error Causes a Mistake in the Identification of the Bottlenecked Device

In a lightly loaded system equation (5.4) - (5.6) apply in all situations. In a heavily loaded system it is necessary to redo the analyses using the correct bottlenecked device. The error in the response time $\Delta T(r)$ due to an error $\Delta x(k,r)$ which causes k to become the bottlenecked device is

$$\Delta T(r) = N y(k,r) \Delta x(k,r) - N [y(b,r) x(b,r) - y(k,r) x(k,r)] \quad (5.14)$$

If the workload requirement of the bottlenecked device was overestimated such that another device is actually the bottlenecked device then k represents the initial bottlenecked device and b the new bottlenecked device in the above equation.

FIGURE 5.1: BOTTLENECK ERRORS

FIGURE 5.2: PLOT OF $\frac{1}{y(i,r)} \frac{\delta T(r)}{\delta x(i,r)}$ VS. $N(r)$

5.1.5. Summary of Error Analysis Using the Single Class Bottlenecked Constraints

Figure 5.1 illustrates the constraints of a single class system and the possible effects of an error. At low utilizations an error in any device shifts constraint AB an amount $y(i,r)\Delta x(i,r)$ to A'B'. At high utilization the slope of line BC is changed to B'C'. This slope is a function only of the bottleneck throughput. Figure 5.2 is a plot of $\frac{1}{y(i,r)} \frac{\delta T(r)}{\delta x(i,r)}$ vs N. In section 5.4 we will examine how values obtained using a more exact method of analysis compare to those predicted by bottleneck analysis.

5.2 The Analysis of Workload Error Propagation Using More Exact Methods

In this section error propagation in more exact models are considered. An M/G/1 queue is an open system with Poisson arrivals and a general service discipline. Figure 5.3 shows a model of such a server. Kleinrock [KLE76] shows that expected wait for such an open system

$$\bar{W} = \frac{U\bar{x}(1+C_b^2)}{2(1-U)}$$

where $\bar{W}$ = expected waiting time

U = utilization = $f\bar{x}$

f = Poisson arrival rate

$\bar{x}$ = general service rate - expected value

C_b^2 = coefficient of variation = $\frac{\sigma^2}{\bar{x}^2} = \frac{x^2}{\bar{x}^2} - 1$

this is a measure of how close the service rate is to exponential. $C^2 \gg 1$ implies a long tail to the distribution and a large variance.

FIGURE 5.3: AN OPEN QUEUE

If it is assumed that C_b^2 remains constant, that is $\sigma^2 = \bar{x}^2 C_b^2$

$$\frac{\delta \bar{w}}{\delta \bar{x}} = \frac{(1+C_b^2)}{2} f \bar{x} \left[\frac{2-f\bar{x}}{(1-f\bar{x})^2} \right]$$

If $C_b^2 = 1$ (M/M/1) then

$$\frac{\delta \bar{w}}{\delta \bar{x}} = \frac{U(2-U)}{(1-U)^2} \quad 0 < U < 1$$

Figure 5.4 shows a plot of $\frac{\delta \bar{w}}{\delta \bar{x}}$ versus U for an M/M/1 queue. This figure shows that a small error in $\bar{x}$ at high utilization has a dramatic effect upon the mean waiting time. Indeed around 75% utilization is the turning point; even a small error in $\bar{x}$ will have a large effect upon the error in $\bar{w}$. At a utilization of .9 an error in $\bar{x}$ is multiplied 100 fold in $\bar{w}$. Conversely at low utilizations the effect is minimal.

Figure 5.5 shows a simple closed queueing model. The response time for this model is

$$T = \frac{N\bar{x}}{1-P(0)} + TH$$

$$\text{where } P(0) = \left[\sum_{n=0}^N \frac{N!}{(N-n)!} (f\bar{x})^n \right]^{-1}$$

$$\text{or } T = \frac{N\bar{x}S}{S-T} + TH$$

$$\text{where } S = \sum_{n=0}^N \frac{N!}{(N-n)!} (f\bar{x})^n$$

FIGURE 5.4: PLOTS OF $\bar{w}$ VERSUS U AND $\frac{\delta \bar{w}}{\delta \bar{x}}$ VERSUS U FOR M/M/1 QUEUE

$$P(n) = \frac{N!}{(N-n)!} \left(\frac{f}{\mu}\right)^n P(0) \quad P(0) = \left[\sum_{n=0}^N \frac{N!}{(N-n)!} (f\bar{x})^n \right]^{-1}$$

$$\text{Cycle Time} = \frac{N\bar{x}}{1-P(0)}$$

$$\text{Response Time} = \frac{N\bar{x}}{1-P(0)} - TH$$

FIGURE 5.5: SIMPLE CLOSED QUEUEING MODEL

FIGURE 5.6: RATE OF CHANGE OF RESPONSE TIMES WITH CHANGING CPU SERVICE TIMES FOR VARIOUS CPU UTILIZATIONS

$$\text{now } \frac{\delta T}{\delta \bar{x}} = \frac{N}{S-1} S + \bar{x} \frac{\delta S}{\delta \bar{x}} - \frac{N \bar{x} S}{(S-1)^2} \frac{\delta S}{\delta \bar{x}}$$

$$\text{where } \frac{\delta S}{\delta \bar{x}} = \frac{1}{\bar{x}} \sum_{n=0}^N \frac{n N!}{(N-n)!} (f \bar{x})^n$$

$$\text{or } \frac{\delta T}{\delta \bar{x}} = \frac{N}{S-1} S - \frac{\bar{x}}{S-1} \frac{\delta S}{\delta \bar{x}}$$

Figure 5.6 shows $\frac{\delta T}{\delta \bar{x}}$ plotted against CPU utilization for values of N. The two limiting cases at 0 utilization of $\frac{\delta T}{\delta \bar{x}} = 1$ and at 100% utilization of $\frac{\delta T}{\delta \bar{x}} = N$ predicted by the bottleneck method are seen. For the major range of practical interest of between .4 and .8 utilization the multiplication factors varies between 2 and $1.5 + N/3$. At utilizations above .8 the N factor is rapidly approach.

5.3 Extended Mean Value Algorithm

In Chapter 4, the basic mean value algorithm for single class systems with load independent service times was presented. This is now extended to obtain $\frac{\delta P}{\delta x(j)}$, the rate of change of the performance parameter P with respect to a change in the service time at device j. P can be $f(i,k)$, $n(i,k)$, $U(i,k)$ or $w(i,k)$ defined on page 83. The boundary conditions are with no jobs in the system the average number of jobs at each device is zero and the rate of change in the average number of jobs at each device due to a change in the service time of device j is zero. As in the basic mean value algorithm the ratio of the number of visits to each device per interaction (visit to reference device) must be known or calculated. For simplicity, 1 will be the number of reference device.

Variables

$f(i,k)$ - throughput at device i with k jobs in system
 i - number of device $i=1,\dots,M$
 j - number of device with error in previous time $1 \leq j \leq M$
 k - number of jobs in the system
 M - total number of devices
 $n(i,k)$ - expected number of jobs at i with k jobs in the system
 N - maximum number of jobs in system
 $SD(i)$ - service discipline of device i
 $T(k)$ - response time with k in the system
 $U(i,k)$ - utilization of i with k jobs in the system
 $w(i,k)$ - expected time at i with k jobs in the system
 $x(i)$ - service time at i
 $y(i)$ - number of visits to i per visit to reference device

BEGIN

```

    READ SD(i),x(i),y(i),j      for i = 1,...,M
    n(i,0):=0                    for i = 1,...,M
     $\frac{\delta n(i,0)}{\delta x(j)} := 0$       for i = 1,...,M

    FOR k=1 to N DO
        FOR i=1 to M DO
            IF (SD(i)=FCFS)      ; FCFS = First Come First Serve
                                   or Processor Sharing

                w(i,k):= x(i) * (1+n(i,k-1))

            ELSEIF (SD(i)=IS)    ; IS = Infinite Server

                w(i,k):= x(i)
    
```

ENDIF

IF ((SD(i) = FCFS) AND (i = j))

$$\frac{\delta w(i,k)}{\delta x(j)} := (1+n(i,k-1)) + x(i) * \frac{\delta n(i,k-1)}{\delta x(j)}$$

ELSEIF ((SD(i) = FCFS) AND (i ≠ j))

$$\frac{\delta w(i,k)}{\delta x(j)} := x(i) * \frac{\delta n(i,k-1)}{\delta x(j)}$$

ELSEIF ((SD(i) = IS) AND (i=j))

$$\frac{\delta w(i,k)}{\delta x(j)} := 1$$

ELSEIF ((SD(i) = IS) AND (i ≠ j))

$$\frac{\delta w(i,k)}{\delta x(j)} := 0$$

ENDIF

ENDFOR

$$f(1,k) := k / \sum_{i=1}^M y(i) * w(i,k)$$

$$\frac{\delta f(1,k)}{\delta x(j)} := \frac{f^2(1,k)}{k} * \sum_{i=1}^M y(i) * \frac{\delta w(i,k)}{\delta x(j)}$$

$$f(i,k) := y(i) * f(1,k) \quad \text{for } i = 2, \dots, M$$

$$\frac{\delta f(i,k)}{\delta x(j)} := y(i) * \frac{\delta f(1,k)}{\delta x(j)} \quad \text{for } i = 2, \dots, M$$

$$U(i,k) := f(i,k) * x(i,k) \quad \text{for } i = 1, \dots, M$$

$$\frac{\delta U(i,k)}{\delta x(j)} := x(i) * \frac{\delta f(i,k)}{\delta x(j)} \quad \text{for } i = 1, \dots, M \quad i \neq j$$

$$:= x(i) * \frac{\delta f(i,k)}{\delta x(j)} + f(i,k) \quad i = j$$

$$n(i,k) := f(i,k) w(i,k) \quad \text{for } i = 1, \dots, M$$

$$\frac{\delta n(i,k)}{\delta x(j)} := f(i,k) \frac{\delta w(i,k)}{\delta x(j)} + w(i,k) \frac{\delta f(i,k)}{\delta x(j)} \quad \text{for } i = 1, \dots, M$$

$$T(k) := \sum_{i=2}^M y(i) w(i,k)$$

$$\frac{\delta T(k)}{\delta x(j)} := \sum_{i=2}^M y(i) \frac{\delta w(i,k)}{\delta x(j)}$$

ENDFOR

END {ALGORITHM}

The algorithm was programmed in FORTRAN77. The resultant performance parameter P' , due to a small change in service time $\Delta x(j)$ may be calculated

$$P' = P + \frac{\delta P}{\delta x(j)} \Delta x(j)$$

This is only valid for small errors in $x(j)$. The basic mean value algorithm can be used to calculate different cases if there is a larger uncertainty in $x(j)$.

5.3.1. Comparison of Bottleneck and Mean Value Algorithms

The bottleneck method provides values at low and high loads. With one job in the system, the mean value algorithm gives

$$w(i,1) = x(i)$$

$$\text{and } \frac{\delta w(i,1)}{\delta x(j)} = \begin{cases} 0 & i \neq j \\ 1 & i = j \end{cases}$$

$$\begin{aligned} \text{thus } \frac{\delta T(1)}{\delta x(j)} &= \sum_{j=z}^M y(i) \frac{\delta w(i,k)}{\delta x(j)} \\ &= y(j) \end{aligned}$$

This is the same result obtained in section 5.1.1.

In a heavily loaded system, the bottleneck method uses the fact that the bottlenecked device determines the throughput for the system. This is now shown to correspond to the results of the mean value algorithm.

Assume that all $y(i), x(i)$ are distinct and nonzero. Choose j such that

$$y(j) x(j) > y(i) x(i) \text{ for all } i \neq j$$

The formula used to obtain $n(i,k)$ may be rewritten using the formula for $w(i,k)$ and $f(i,k)$ thus

$$\begin{aligned} n(i,k) &= f(i,k) w(i,k) \\ &= f(i,k) y(i) x(i) [1+n(i,k-1)] \end{aligned}$$

Solving for $f(i,k)$

$$f(i,k) = \frac{n(i,k)}{y(i) x(i) [1+n(i,k-1)]} \text{ for all } i$$

This is true for all i and in particular for $i=j$. Thus

$$\frac{n(j,k)}{n(i,k)} = \frac{y(j) x(j) [1+n(j,k-1)]}{x(i) x(i) [1+n(i,k-1)]}$$

$$\text{let } a_1 = \frac{y(j) x(j)}{y(i) x(i)}$$

since $y(j) x(j) > y(i) x(i)$

$$a_1 > 1$$

It follows

$$\frac{n(j,1)}{n(i,1)} = a_1$$

22
OF / DE

4

$$\frac{n(j,2)}{n(i,2)} = a_1 \frac{[1 + a_1 n(i,1)]}{1 + n(i,1)} = a_1 a_2$$

$$n(i,1) > 0, \quad a_1 > 1$$

$$a_2 > 1$$

$$\text{hence } \frac{n(j,k)}{n(i,k)} = a_k a_{k-1} \dots a_2 a_1$$

$$\text{where all } a_k, a_{k-1} \dots a_1 > 1$$

now

$$f(l,k) = \frac{K}{\sum_{i=1}^M y(i) w(i,k)}$$

$$K = \sum_{i=1}^M n(i,k)$$

$$f(l,k) = \frac{1}{n(j,k-1)} + \frac{\sum_{i=1}^M \frac{n(i,k-1)}{n(j,k-1)}}{\sum_{i=1}^M \frac{y(i) x(i) [1+n(i,k-1)]}{n(j,k-1)}}$$

$$\text{as } k \rightarrow \infty \quad \frac{n(i,k)}{n(j,k)} \rightarrow 0 \quad i \neq j$$

$$\rightarrow 1 \quad i = j$$

$$\text{and } \frac{\text{constant}}{n(j,k)} \rightarrow 0$$

$$\therefore f(l,k) = \frac{0+1}{0+y(j)x(j)} = \frac{1}{y(j)x(j)}$$

The result used in bottleneck analysis. It follows that

$$\frac{\delta f(l,k)}{\delta x(i)} = 0 \quad i \neq j \quad (\text{non bottleneck case})$$

$$= - \frac{1}{y(x)x(j)} \quad i = j \quad (\text{bottleneck case})$$

$$= - \frac{f(l,k)}{x(j)}$$

as was obtained in sections 5.1.2 and 5.1.3.

5.4 Illustrative Examples

5.4.1 High and Low Loads

We now illustrate the above with examples. Figure 5.7 is a simple closed central server model. Figure 5.8 is a plot of $\frac{1}{y(j)} \frac{\delta T(k)}{\delta x(j)}$ versus k for both $j = \text{CPU}$, the bottleneck device and $j = \text{DK1}$ a non-bottleneck device. It is seen that at low utilization (small k), the intended mean value algorithm and the bottleneck analysis both give a value of 1. At high utilization (large k), the bottleneck analysis predicts a value of k if the device is the bottleneck device, 0 otherwise. It is seen that the values calculated using the extended mean value analysis approaches these values for low and high values of k . Figure 5.9 is a plot of $\frac{1}{f(1,k) y(j)} \frac{\delta f(1,k)}{\delta x(j)}$ versus k for the same model. At low utilization from equation (5.3)

$$\begin{aligned} \frac{1}{f(1,k) y(j)} \frac{\delta f(1,k)}{\delta x(j)} &= - \frac{1}{C} = - \frac{1}{\sum_{i=1}^M y(i) x(i)} \\ &= - \frac{1}{15.99} \\ &= - .0625 \text{ sec}^{-1} \end{aligned}$$

At high loads the bottleneck provides two limits. If j is a non-bottleneck device from equation (5.8)

$$\frac{1}{y(j) f(1,k)} \frac{\delta f(1,k)}{\delta x(j)} = 0$$

Or if j is the bottlenecked device from equation (5.10)

$$\begin{aligned} \frac{1}{f(1,k) y(j)} \frac{\delta f(1,k)}{\delta x(j)} &= - \frac{1}{y(j) x(j)} \\ &= 1.25 \text{ sec}^{-1} \end{aligned}$$

These asymptotes are shown on Figure 5.9. The bottleneck device (CPU) and the non-bottleneck device (DK1) are seen to approach their respective asymptotes. The behaviour of the non-bottleneck device first rises above the asymptote then goes to zero as the bottleneck device saturates

Device	Service Discipline	Mean Service Time Per Visit	Branching Probabilities (device, probability)		No. of Visits Per Inter- action
Ter	infinite service	15 seconds	CPU	1.000	1
CPU	processor sharing	.04 seconds	TER	.050	20
			DK1	.450	
			DK2	.500	
DK1	first come first served	.01 seconds	CPU	1.000	9
DK2	first come first served	.01 seconds	CPU	1.000	10

FIGURE 5.7: EXAMPLE SYSTEM CONSIDERED IN 5.4

FIGURE 5.8: PLOT OF $\frac{1}{y(j)} \frac{\delta T(k)}{\delta x(j)}$ VERSUS k FOR MODEL OF FIGURE 5.7

FIGURE 5.9: RATE OF CHANGE OF THROUGHPUT VERSUS NUMBER OF JOBS FOR EXAMPLE (MODEL OF FIGURE 5.7)

FIGURE 5.10: RATE OF CHANGE OF THROUGHPUT VERSUS NUMBER OF JOBS FOR NON BOTTLENECK DEVICE (DK1 OF EXAMPLE, FIGURE 5.7)

Figure 5.10 shows this in more detail. At small k a small increase in the service time will cause the throughput to be decreased. As the effect of the bottleneck begins to be felt, more jobs are waiting and there is a decreasing effect on throughput.

Figure 5.11 is a table which shows the change at low utilization (5 jobs) and high utilization (40 jobs) predicted using the constraint approach, the extended mean value approach and the values calculated using the basic mean value algorithm with the changed values. In these good results are obtained with both the extended mean value algorithm and the bottleneck method.

5.4.2 Moderately Loaded System

Figure 5.12 summarizes the values obtained when the system of Figure 5.6 has a moderate load of 15 jobs. Both the CPU and DK1 are considered. Figure 5.13 are the results for the same system but with the basic service time of DK1 changed to 0.05 seconds with a load of 15 and 25. The accuracy is not as good here because of the steep slope seen in Figure 5.9.

It can be seen that at moderate loads the sensitivity to error is greater and more difficult to predict. This is the non-linear section of the graph of cycle time versus load of Figure 4.2.

CPU Service Time	CPU Utilization	DK1 Service Time	Bottleneck Prediction ΔT_B	Extended Mean Value Prediction $\Delta T_E \frac{\delta x(j)}{\delta x(j)}$	Basic Mean Value Response Time T	ΔT_{MV} Change in Mean Value Response Time	% Difference $\frac{\Delta T_{MV} - \Delta T_g}{T}$	$\frac{\Delta T_{MV} - \Delta T_E}{T}$
High Load - Change in Non-Bottleneck Device								
.04	.24	.010	-	-	1.1795	-	-	-
.04	.24	.005	-.045	-.0464	1.1336	-.0459	.08	.04
.04	.24	.011	.009	.0093	1.1888	.0093	.03	0.0
.04	.24	.015	.045	.0464	1.2265	.0470	.16	.05
.04	.24	.020	.090	.0929	1.2745	.0950	.39	.16
.04	.24	.040	.270	.2786	1.4774	.2979	1.9	1.3
High Load - Error in Bottleneck Device								
.04	1.00	.010	-	-	17.0003	-	0.0	0.0
.04	1.00	.005	0	0.0000	17.0003	0.0000	0.0	0.0
.04	1.00	.012	0	0.0000	17.0003	0.0000	0.0	0.0
.04	1.00	.015	0	0.0000	17.0003	0.0000	0.0	0.0
.04	1.00	.020	0	.0001	17.0004	0.0001	0.0	0.0
.04	1.00	.040	0	.0002	17.0011	0.0008	-0.0	0.0
High Load - Change in Bottleneck Device								
.04	1.00	.01	-	-	17.0003	-	-	-
.03	1.00	.01	8.0	-7.998	9.0428	7.9575	.53	.51
.05	1.00	.01	8.0	7.998	25.0000	7.9997	.00	.00
.08	1.00	.01	32.0	31.992	49.0000	31.9997	.00	.00
.10	1.00	.01	48.0	47.988	65.0000	47.9997	.00	.00

FIGURE 5.11: SUMMARY OF RESULTS FOR MODEL OF FIGURE 5.7 FOR LOW (5 JOBS) AND HIGH (40 JOBS) LOADS

CPU Time	CPU Utilization	DK1 Service Time	Extended Mean Value Prediction $\Delta T_E = \frac{\delta T}{\delta x(j)} \Delta x(j)$	Basic Mean Value Response Time T	ΔT_{MV} Change in Mean Value Response Time	% Difference $\frac{\Delta T_{MV} - \Delta T_E}{T}$
.04	.70	.01	$\frac{\delta T}{\delta x(j)} = 102.4$	2.2177	-	-
.03	.70	.01	1.0240	1.3724	.8453	13.0%
.05	.70	.01	1.0240	3.4501	1.2324	6.0%
.08	.70	.01	4.096	9.6781	7.4604	34.8%
.04	.70	.01	$\frac{\delta T}{\delta x(j)} = 9.060$	2.2177	-	-
.04	.70	.005	.0453	2.1742	.0435	0.1%
.04	.70	.015	.0453	2.2649	.0472	0.1%
.04	.70	.02	.0906	2.3162	.0985	0.3%
.04	.70	.03	.1812	2.4327	.2150	1.4%

FIGURE 5.12: MODERATE LOADS - SYSTEM OF FIGURE 5.6 (15 JOBS)

CPU Service Time	CPU Util.	DK1 Service Time	DK1 Utilization	Extended Mean Value Prediction $\Delta T_E = \frac{\delta T}{\delta x(j)} \Delta x(j)$	Basic Mean Value Response Time T	ΔT_{MV} Change in Mean Value Response Time	% Difference $\frac{\Delta T_{MV} - \Delta T_E}{T}$
.04	.67	.05	.38	$\frac{\delta T}{\delta x(j)} = 18.17$	2.7375		
	.67	.06	.45	.1817	2.9370	.1995	.61
	.67	.07	.52	.3634	3.1769	.4394	2.39
	.67	.08	.58	.5451	3.4651	.7276	5.27
	.67	.09	.65	.7268	3.8103	1.0728	9.08
.04	.95	.05	.53	$\frac{\delta T}{\delta x(j)} = 17.60$	6.1082		
	.95	.06	.63	.1760	6.3306	.2224	.73
	.95	.07	.73	.3520	6.6827	.5745	3.33
	.95	.08	.81	.5280	7.2341	1.1259	8.27
	.95	.09	.88	.7040	8.0630	1.9548	15.51

FIGURE 5.13: MODERATE LOADS - SYSTEM OF FIGURE 5.6 (15 JOBS AND 25 JOBS)

5.5 Conclusions

Computer systems which have all devices operating at a low utilization have a linear relationship between an error in the estimation of a workload and the resultant error in the response time. The advantage of such systems is they have good and stable responses. Their disadvantage is that they may have expensive equipment idle.

On the otherhand, computer systems which operate at high loads in which one device operates near 100% utilization are very sensitive to errors and fluctuation in the workload on that device. It was seen that the error in response time due to an error in workload is multiplied by a factor approaching the number of jobs times the visits per job ($N \cdot y(i) \cdot \Delta x(i)$). Care must be taken in specifying the response time for such a system because of the variations which can occur. Figure 5.8 illustrated this sensitivity for a specific example. However, the response time tends to be independent of errors in the workload in the non-bottleneck devices of a bottlenecked system. This confirms the importance of identifying bottleneck and potential bottlenecks.

Chapter 6

CONCLUSIONS

The importance of identifying bottleneck devices in predicting performance parameters from program source code and source code estimations was confirmed. It was found that errors in the workload on the bottleneck device have a profound effect on performance parameters. The effect of errors in workload on non-bottleneck devices on the other hand were found to be minimal.

A number of the difficulties in obtaining workloads from estimations of source code were found - data dependency, identification and allowance for intrinsic functions, variations in programming styles. These difficulties can result in large errors in workload predictions and thus performance.

The importance of modern programming techniques particularly modularization in performance prediction was illustrated. Modularization can allow estimates of resource requirements of a program to be refined in stages. As modules are completed the estimates can be improved. This is particularly important on large projects where such stepwise improvements can allow necessary modifications to be made at a minimal cost.

Future work could extend the error propagation analysis to multiclass systems and examine the effects of concurrency and blocking on the accuracy of an analysis. Additional studies could be conducted to develop accurate methods of predicting workloads at early stages in the development of software.

REFERENCES

- [AH074] Aho, Alfred, Hopcroft, John and Ullman, Jeffrey, The Design and Analysis of Computer Algorithms, Addison-Wesley Publishing Company, Don Mills, 1974.
- [BEI78] Beizer, Boris, Micro-Analysis of Computer System Performance, Van Nostrand Reinhold, Toronto, 1978.
- [B0080] Booth, Taylor, and Wiecek, Cheryl, "Performance Abstract Data Types as a Tool in Software Performance Analysis and Design", IEEE Transactions on Software Engineering, Vol.SE-6, No.2, March 1980.
- [BOW81] Bowen, B.A., et al., "Analysis and Evaluation of VLSI/VHSI Technology on High Performance Digital Signal Processors", Progress Report - Phase 1B, Microprocessor Systems Development Laboratory, Carleton University, Ottawa, October 1981.
- [BRU80] Bruell, S.C. and Balbo, G., Computational Algorithms for Closed Queueing Networks, Toronto, 1980.
- [BUZ81] Buzen, J.P., et al., "Predicting Software Performance with Crystal", 3rd Annual International Conference on Computer Capacity Management, Chicago, Illinois, April 1981.
- [COM81] Computer-Special Issue on ADA, Volume 14, Number 6, June 1981.
- [CUR76] Curnow, H.J. and Wickmann, B.A., "A Synthetic Benchmark", The Computer Journal, Vol.19, #1, 1976.
- [DOT80] DOTY Associates, Handbook of Procedures for Estimating Computer System Sizing and Timing Parameters, DOTY Associates, Rockville, Maryland, February 1980.
- [DUN81] Dunsworth, Allen E., "A Model for a University Timesharing Computer", M.Sc. Dissertation, Carleton University, 1981.
- [EDP81] EPD Performance Review, Phillip C. Howard, editor, Applied Computer Research, Phoenix, Arizona, Vol.9, Number 8, August 1981.
- [GEL80] Gelenbe, E. and Mitrani, I., Analysis and Synthesis of Computer Systems, Academic Press, Toronto, 1980.
- [GUT77] Guttag, John, "Abstract Data Types and the Development of Data Structures", Communications of the ACM, June 1977, Volume 29, No.6.

- [HEC77] Hecht, Matthew, Flow Analysis of Computer Programs, North-Holland, New York, 1977.
- [HER79] Herzog, U. and Hoffman, W., "Synchronization Problems In Hierarchically Organized Multiprocessor Computer Systems", in Performance of Computer Systems, ed. by Arato, Butrimenko and Gelenbe, North-Holland Publishing Company, 1979.
- [KEL74] Kelly, John, The Theory of Repetition Networks with Application to Computer Programs, Ph.D. Dissertation 1974, Purdue University.
- [KNU68] Knuth, D.E., Fundamental Algorithms, Addison-Wesley Publishing Company, Don Mills, 1968.
- [KLE76] Kleinrock, L., Queueing Systems Volume II: Computer Applications, John Wiley & Sons, Toronto, 1976.
- [LIA80] Lias, E.J., "Tracking the Elusive KOPS", Datamation, November 1980.
- [LIN80] Ling, R.F. and Roberts, U.H., User's Manual for IDA, McGraw-Hill, Toronto, 1980.
- [MER79] Merrill, Herbert, A Comprehensive Approach to the Performance Measurement and Evaluation of Large Scale Computer Systems, Ph.D. Dissertation, University of Illinois, 1979.
- [PLA81] Plattner, Bernhard and Nievergelt, Jurg, "Monitoring Program Execution: A Survey", IEEE Computer, Volume 14, Number 11, November 1981.
- [KLE75] Kleinrock, L., Queueing Systems Volume I: Theory, John Wiley & Sons, Toronto, 1975.
- [REI80] Reiser, M. and Lavenberg, S., "Mean Value Analysis of Closed Multichain Queueing Networks", Journal of the Association for Computing Machinery, Volume 22, Number 2, April 1980.
- [RID80] Riddle, W.E. and Fairley, R.E., Software Development Tools, Springer-Verlag, Berlin, 1980.
- [SAN78] Sanguinetti, John, "A Formal Technique for Analyzing the Performance of Complex Systems", Computer Performance Evaluation Users Group, NBS Special Publication 500-41, October 1978.

- [SM180] Smith, C.U., The Prediction and Evaluation of the Performance of Software from Extended Design Specification, Ph.D. Dissertation, University of Texas, August 1980.
- [WEG80] Wegner, Peter, Programming with ADA, Prentice-Hall, New Jersey, 1980.
- [YEH77] Yeh, R.T., editor, "Software Specification and Design", Current Trends in Programming Methodology, Volume 1, Prentice-Hall, New Jersey, 1977.
- [YEH78] Yeh, R.T. and Chandy, K.M., editors, "Software Modelling", Current Trends in Programming Methodology, Volume 2, Prentice-Hall; New Jersey, 1978.

APPENDIX 1

PROGRAM GAP

```

*****
*
* This is a program that will read in a graph and analysis
* it to obtain the overall expected value and variance.
* The main program acts as the controller. It loops either
* 100 times or until the operator has processed the desired
* number of graphs. It is possible to enter the values
* interactively or from an existing file. If a file or files
* are to be used, they should be associated with the unit
* number using the IBEX set command.
*   eg. SET unit no. fid ,EXIST=OLDFILE
*   (EXIST=OLDFILE is optional, it allows the
*   use of the same file for a number of graphs.
*
* At each time it is possible to either modify the nodes
* of the existing graph or else to enter a completely new
* graph. It is possible to run a graph, obtain the results
* make changes in the graph, give it a new name and run it
* again. The new name is to avoid confusion and is not
* essential.
*
* Four subroutines are used:
*   Readgraf - which reads data formatted by output from
*               a file
*   Readin - which reads in the data for one node of the graph
*               -which may be a collapsed node
*   Output - which prints out on a specified device,
*               the graph plus
*               the calculated values for the mean time
*               and the variance.
*   Calcu - which performs the actual analysis.
*
* Size of problem which may be analyzed. At present this
* is set at 100 nodes with a maximum of 5 branches from each
* node. These values are represented by MAXNO and SIZE
* respectively. If it is desired to modify these, they must
* be changed in the main program, plus the array sizes in the
* program must be modified.
* IN ADDITION it is necessary to change
* the stack sized used in calculate.
* The program can handle up to 6 resources at one time
*
* GLOBAL VARIABLES:
*   NODENAME - array(1..MAXNC) of graph node names. Each
*               node name consists of 3 characters.
*   NODETYPE - corresponding array(1..MAXNO) of graph
*               node types. Each nodetype consists of two
*               characters.
*               BA - basic node has time and variance.
*               BO - basic cr has time and variance and
*                   to SIZE branches leaving with
*                   probabilities which must sum to 1
*               BE - basic end of repetition loop, has
*                   associated time and variance.
*               BT - basic terminus of branches of or
*                   node, with associated time and
*                   variances.

```

```

*
*      CL - collapsed node, refers to a
*           previously defined and executed
*           graph from which the time and
*           variance are obtained.
*
*      DU, DO, DE, DT - dummy nodes which do
*           not have time and variances
*
*      RE - repetition node - may have
*           time and variance, also has an
*           expected number of repetitions and
*           variance.
*
*      BRANCHNODE - nodes branched to from an or node. 3
*                   characters - must be a node defined at
*                   some point in the input.
*
*      BRANCHPROB - associated probability - real
*
*      RESOURCE - expected resource requirement of a
*                  node.
*
*      VAR - associated variance
*
*      REPETITIONS - expected number of repetitions of a
*                     repetition node.
*
*      REPVAR - associated variance
*
*      MAXNO - maximum number of nodes which can be
*              considered.
*
*      SIZE - maximum number of branches from a given
*              or node.
*
*      TOTALRESOURCE - total expected time to graph -
*                      calculated from input parameters by CALCU
*
*      TOTVAR - associated variance of graph - calculated
*              from input parameters by CALCU
*
*      NORES - number of resources (at present
*              limited to 6)
*
*      MAXNORES - maximum allowable number of resources.
*
*      FILEOUT - unit number for output of calculated
*                graph.
*
*      FILEIN - unit number for input of a graph.
*
*      ANS - a logical variable used in calling
*            OUTPUT; FALSE - only the graph is
*            printed formatted for a CRT;
*            TRUE - everything is printed
*            formatted for a lineprinter.
*
*      NUMBER - number of nodes considered.
*
*      GRAPH(I) - name of graph associated with
*                 collapsed node I.
*
*      GRAFNAME - name of graph being considered.
*
*
*

```

Barry R. Thomas, May 1981

IMPLICIT LOGICAL (A-2)

CHARACTER

```

1  BRANCHNODE(100,5)*3, GRAFNAME*8, GRAPH(100)*8, NODE*2,
2  NODEIN*14, NODENAME(100)*3, NODETYPE(100)*2, TEMP*3

```

REAL

```

1  BRANCHPROB(100,5), REPETITIONS(100), REPVAR(100),
2  RESOURCE(100,6), TOTALRESOURCE(6), TOTVAR(6), VAR(100,6)

```

INTEGER

```

1  COUNT, COUNT2, COUNT3, FILEOUT, GF, G1, I, IND, J, K, L,
2  MAXCOUNT, MAXNO, MAXNORES, MAXREC, NORES, NUMBER, SIZE, TIMES,
3  WAY

```

LOGICAL

```

1  AGAIN, CHANGE, CONTINUE, GO, NOTFOUND, OUT

```

```

PARAMETER (MAXNO=100,SIZE=5,MAXNORES=6,MAXCOUNT=100,
1      MAXREC=10000)
9000 FORMAT(3(1X,A//))
CONTINUE
REPEAT 1080,WHILE CONTINUE
*   main loop which allows one of three courses
*   of action to be taken. The program allows the
*   input and output files of the graphs to be changed.
*   It is also possible to check the input and change
*   one or more of the values entered.
WRITE (108,9010)
9010 FORMAT(1X,'   THERE ARE 3 OPTIONS'//
1      1X,'1. Enter a new graph from an existing file'//
2      1X,'2. Enter a new graph interactively'//
3      1X,'3. Make a change in the existing graph,'//
4      1X,'   including where the output is directed'//
5      1X,'   or adding additional nodes.'//
6      1X,' ENTER 1, 2, or 3',//)
READ (105,'(G)',ERR=1090) WAY
IF (WAY.EQ.1)THEN
*   read the graph from an existing file
CALL READGRAF(GRAFNAME,NODENAME,NODETYPE,BRANCHNODE,
1      BRANCHPROB,REPETITIONS,REPVAR,RESOURCE,VAR,NUMBER,
2      NORES,MAXREC,MAXNO,SIZE,MAXNORES,GRAPH)
ELSEIF (WAY.EQ.2)THEN
*   read the graph interactively
WRITE (108,'(A//)') ' Please give graph name.'
READ (105,'(A)') GRAFNAME
GO=.TRUE.
REPEAT 1000,WHILE GO
WRITE (108,'(A//)') ' Please give number of nodes.'
READ (105,'(G)') NUMBER
WRITE (108,'(A//)') ' Number of resources, max=6'
READ (105,'(G)') NORES
IF (NUMBER.LT.1.OR.NUMBER.GT.MAXNO)THEN
WRITE (108,9000) 'NUMBER OF NODES OUT OF RANGE'
ELSEIF (NORES.LT.1.OR.NORES.GT.MAXNORES)THEN
WRITE (108,9000) 'No. of resources out of range'
ELSE
GO=.FALSE.
ENDIF
1000 CONTINUE
DO 1010 J=1,NORES
TOTALRESOURCE(J)=0.0
TOTVAR(J)=0.0
1010 CONTINUE
*   loop to read in data for each node
I=0
DO 1020 L=1,MAXNO
NODENAME(L)='END'
1020 CONTINUE
GO=.TRUE.
REPEAT 1050,WHILE (I.LE.MAXNO.AND GO)
I=I+1
WRITE (108,9020)
9020 FORMAT(1X,'Give node name', ' Enter ? to',
1      ' obtain a listing of nodes entered'//
2      1X,'END to finish',//)
READ (105,9030) NODEIN
9030 FORMAT(A)

```

```

IF (INDEX(NODEIN,'END').NE.0)THEN
  IF (I-1.NE.NUMBER)THEN
    WRITE (108,9040) (I-1),NUMBER,(I-1)
    FORMAT(1X,'NO. OF NODES ENTERED=',I4,/,
      1X,'NO. EXPECTED=',I4,/,
      1X,'NO. OF NODES SET TO',I4,/)
    NUMBER=I-1
  ENDIF.
  GO=.FALSE.
  ELSEIF (INDEX(NODEIN,'?').NE.0)THEN
    DO 1030 L=1,I-1
      WRITE (108,9050) NODENAME(L)
      FORMAT(1X,5(A3,1X))
    CONTINUE
    I=I-1
  ELSE
    GX=I-1
    DOUBLE=.FALSE.
    DO 1040 J=1,GX
      IF (NODEIN(1:3).EQ.NODENAME(J))THEN
        WRITE (108,9060) NODENAME(J)
        FORMAT(1X,A,' is used twice',/)
        I=I-1
        DOUBLE=.TRUE.
      ENDIF
    CONTINUE
    IF (.NOT.DOUBLE)THEN
      NODENAME(I)=NODEIN(1:3)
      CALL READIN(NODENAME,NODETYPE,BRANCHNCD,
        1 BRANCHPROB,REPETITIONS,RESOURCE,I,MAXNO,SIZE,
        2 VAR,REPVAR,MAXNORES,NORES,GRAPH,MAXREC)
    ENDIF
  ENDIF
  CONTINUE
  ELSEIF (WAY.EQ.3)THEN
    at this point do nothing
  ELSE
    WRITE (108,'(A)') ' 1,2, or 3 not entered, no',
      1 ' action taken - reenter later'
    ENDIF
    WRITE (108,9000) ' COPY OF GRAPH?',
      1 ' yes - enter TRUE(T)',
      2 ' no - FALSE(F)'
    READ (105,'(L)') OUT
    IF (OUT) THEN
      CALL OUTPUT(NODENAME,NODETYPE,BRANCHNODE,BRANCHPROB,
        1 RESOURCE,NUMBER,TOTALRESOURCE,MAXNO,SIZE,REPETITIONS,
        2 VAR,TOTVAR,REPVAR,108,.FALSE.,MAXNORES,GRAFNAME,NORES,
        3 GRAPH)
    ELSE
      ENDIF
    IF (WAY.NE.3)THEN
      WRITE (108,9000) ' Do you wish to make any changes',
        1 ' yes - enter TRUE(T)',
        2 ' no - FALSE(F)'
      READ (105,'(L)') OUT
      IF (OUT) WAY=3
    ENDIF
    IF (WAY.EQ.3)THEN

```

```

GO=.TRUE.
J=0
WRITE (108,'(1X,2A,/2A//)',ERR=1100) 'Graph name= '
1   GRAFNAME,' Enter TRUE(T) to change, otherwise',.FALSE'
READ (105,'(L)',ERR=1090) CHANGE
IF (CHANGE)THEN
    WRITE (108,'(1X,A)',ERR=1100) 'New graph name'
    READ (105,'(A8)',ERR=1090) GRAFNAME
ELSE
    ENDIF
REPEAT 1070,WHILE(GO)
    J=J+1
    IF (J.GE.MAXNO) GO=.FALSE.
    WRITE (108,9070)
    FORMAT(1X,'Name of node to change',
1       3X,'END to finish',/)
    READ (105,9080) NODEIN
    FORMAT(A)
    IF (INDEX(NODEIN,'END').NE.0)THEN
        GO=.FALSE.
    ELSE
        NOTFOUND=.TRUE.
        I=0
        REPEAT 1060,WHILE(NOTFOUND)
            I=I+1
            TEMP=NODENAME(I)
            IF (I.GT.MAXNO)THEN
                WRITE (108,'(1X,A)',ERR=1100) 'MAXIMUM NUMBER',
2               ' OF NODES EXCEEDED'
                NOTFOUND=.FALSE.
            ELSEIF (I.GT.NUMBER)THEN
                WRITE (108,'(1X,A)',ERR=1100) 'NEW NODE'
                NODENAME(I)=NODEIN(1:3)
                This must be a new name, so does not
                have to be checked for duplicates.
                NUMBER=NUMBER+1
                CALL READIN(NODENAME,NODETYPE,BRANCHNODE,
3               BRANCHPROB,REPETITIONS,RESOURCE,I,MAXNO,
                SIZE,VAR,REPVAR,MAXNORES,NORES,GRAPH,
                MAXREC)
                NOTFOUND=.FALSE.
            ELSEIF (TEMP.EQ.NODEIN(1:3))THEN
                NOTFOUND=.FALSE.
                WRITE (108,'(1X,2A//)',ERR=1100) 'Enter the name for',
1               ' node - can be same or a new one.',
2               'If a new one it must be consistent.'
                READ (105,'(A)',ERR=1090) NODENAME(I)
                CALL READIN(NODENAME,NODETYPE,BRANCHNODE,
1               BRANCHPROB,REPETITIONS,RESOURCE,I,MAXNO,
2               SIZE,VAR,REPVAR,MAXNORES,NORES,GRAPH,
3               MAXREC)
            ELSE
                go through again lookin for node.
            ENDIF
        CONTINUE
    1060   WRITE (108,9000) ' Copy of graph?'
        1   //,' yes - enter TRUE(T),//,' no   FALSE(F)'
        READ (105,'(L)',ERR=1090) OUT
        IF (OUT) THEN
            CALL OUTPUT(NODENAME,NODETYPE,BRANCHNODE,

```

```

1          BRANCHPROB,RESOURCE,NUMBER,TOTALRESOURCE,
2          MAXNO,SIZE,REPETITIONS,VAR,TOTVAR,REPVAR,108,
3          FALSE,MAXNORES,GRAFNAME,NORES,GRAPH)
          ELSE
          ENDIF
        ENDIF
1C70      CONTINUE
        ENDIF
*
*      Call the subroutine(s) which will analyze the data.
*
      CALL CALC(NODENAME,NODETYPE,BRANCHNODE,BRANCHPROB,RESOURCE,
1          NUMBER,TOTALRESOURCE,MAXNO,SIZE,REPETITIONS,REPVAR,VAR,
2          TOTVAR,NORES,MAXNORES)
*      Printout results
*
      WRITE (108,9000)'Please give unit number of output file'
      READ (105,'(G)') FILEOUT
      CALL OUTPUT(NODENAME,NODETYPE,BRANCHNODE,BRANCHPROB,RESOURCE,
1          NUMBER,TOTALRESOURCE,MAXNO,SIZE,REPETITIONS,VAR,TOTVAR,
2          REPVAR,FILEOUT,TRUE,MAXNORES,GRAFNAME,NORES,GRAPH)
      WRITE (108,9090)
9090      FORMAT(1X,'RUN AGAIN, TRUE OR FALSE?',/)
      READ (105,'(L)',ERR=1090) CONTINUE
1C80      CONTINUE
      STOP
1090      WRITE (108,'(A)') ' ERROR IN READ'
      STOP
1400      WRITE (108,'(A)') ' ERROR IN WRITE'
      STOP
      END

```



```

SUBROUTINE CALCU(NODENAME, NODETYPE, BRANCHNODE, BRANCHPROB,
1      RESOURCE, NUMBER, TOTALRESOURCE, MAXNO, SIZE, REPETITIONS,
2      REPVAR, VAR, TOTVAR, NORES, MAXNORES)

```

 Where the above global variables are defined in the main program.

Barry R. Thomas, May 1981

This subroutine calculates the mean and variance of a graph with the following restrictions:

1. loops can be nested but they cannot intersect.
 Branches must leave a common node and go to a common node and the sum of their probabilities must equal one.
2. There can be no premature exits from repetition loops or branches.
3. There can be only one start and one final node.
4. Dummy nodes can be used to help fit actual situations to the above restrictions.

FORMULAS USED:

1. series - $ET = \sum (EX(i))$
 - $VarT = \sum (Var(i))$ all i
2. parallel - $ET = \sum (p(i) * EX(i))$
 - $VarT = \sum (p(i) * Var(i)) +$
 $\sum (p(i)p(j)[EX(i) - EX(j)]^2)$
 over all pairs i, j
3. loop - $ET = EXEN$
 - $VarT = ENVarx + EX^2 * VarN$

where E refers to the expected value

T refers to total time

i, j refer to nodes

SUM - sum over all nodes

p - probability of going to node i

METHOD USED

A series of stacks are used to save values until an inner loop or parallel grouping is reached. The values are then calculated outward from this point. When an or-node is reached (branching node), then those values are added to the stack. A separate stack pointer is kept for each group of branching nodes (ie all branches exiting a common node are considered a group.)

LOCAL VARIABLES USED:

STACKS-

- ST - main stack on which branching and looping nodes are kept until required.
- STP - corresponds to main stack and keeps the related probabilities.
- SPX - corresponding stack to keep the expected time for the branch.
- STV - corresponding stack to keep the variance for the branch.
- BND - stack used to keep number of branches from any one node.

```

*      REP - corresponding stack to keep repetitions.
*      RVAR - corresponding stack to keep the variance
*              of the repetitions.
*
*      SP - stack pointer
*      BNP - branch pointer
*      RP - repetition pointer
*      I - number of node being considered
*      J - a particular stack pointer.
*      K, L, R - counters,
*      GO - a boolean used to signal when finished.
*      NOTFOUND - boolean used to indicate that node number
*              of a particular node has not been found.
*      NODE - node type of current node
*      CURRENT - node to be considered, name corresponding to I.
*
* .....
IMPLICIT LOGICAL (A-Z)
global variables
CHARACTER
1  BRANCHNODE(MAXNO,SIZE)*3, NODENAME(MAXNO)*3, NODETYPE(MAXNO)*2
REAL
1  BRANCHPROB(MAXNO,SIZE), REPETITIONS(MAXNO), REPVAR(MAXNO),
2  RESOURCE(MAXNO,MAXNORES), TOTALRESOURCE(MAXNORES),
3  TOTVAR(MAXNORES), VAR(MAXNO,MAXNORES)
INTEGER
1  MAXNO, MAXNORES, NORES, NUMBER, SIZE

*
*      Local variables
*
CHARACTER
1  CURRENT*3, NODE*2, ST(100)*3
REAL
1  REP(100), RVAR(100), STP(100), STV(0:100,6), STX(0:100,6),
2  TEMP(6)
INTEGER
1  BNO(100), BNP, I, J, K, L, M, R, RP, SP(0:100)
LOGICAL
1  GO, NOTFOUND
Initialization
GO=.TRUE.
RP=1=SP(1)=1
J=BNP=SP(0)=0
DO 1010 M=0,100
  DO 1000 R=1,NORES
    STX(M,R)=STV(M,R)=0.0
1000  CONTINUE
1010  CONTINUE

*
*      Main loop, GO becomes false when all nodes have been considered
*      or the next node cannot be found.
*
REPEAT 1180, WHILE GO
  NODE=NODETYPE(I)

*
*      According to type of node do calculations and add or
*      remove from stack.
*      IF (NODE.EQ.'BO'.OR.NODE.EQ.'DO') THEN
*          that is an or-node.

```

```

IF (NODE.EQ.'BO') THEN
  DO 1020 R=1, NORES
    STX(SP(J),R)=STX(SP(J),R)+RESOURCE(I,R)
    STV(SP(J),R)=STV(SP(J),R)+VAR(I,R)
1020  CONTINUE
  ELSE
  ENDIF
  BNP=BNP+1
  K=1; BNO(BNP)=0; J=J+1
  REPEAT 1040, WHILE (BRANCHNODE(I,K).NE.'END')
    BNO(BNP)=BNO(BNP)+1
    DO 1030 R=1, NORES
      STX(SP(J),R)=STV(SP(J),R)=0.0
1030  CONTINUE
      ST(SP(J))=BRANCHNODE(I,K)
      STP(SP(J))=BRANCHPROB(I,K)
      K=K+1
      SP(J)=SP(J) + 1
      IF (K.GT.SIZE) THEN
        WRITE (108,9000)
        GO TO 1050
      ELSE
      ENDIF
1040  CONTINUE
1050  CONTINUE
      BNO(BNP+1)=BNO(BNP)
      BNP=BNP+1
      SP(J+1)=SP(J)
      SP(J)=SP(J)-1
      obtain a new current node
      CURRENT=ST(SP(J))
    ELSEIF (NODE.EQ.'DJ') THEN
      dummy node obtain next node
      CURRENT=BRANCHNODE(I,1)
    ELSEIF (NODE.EQ.'BA'.OR.NODE.EQ.'CL') THEN
      basic node, add values, obtain next node
      collapsed node, same thing
      DO 1060 R=1, NORES
        STX(SP(J),R)=RESOURCE(I,R)+STX(SP(J),R)
        STV(SP(J),R)=VAR(I,R)+STV(SP(J),R)
1060  CONTINUE
      CURRENT=BRANCHNODE(I,1)
    ELSEIF (NODE.EQ.'RE') THEN
      repetition node, add values to stacks
      J=J+1
      REP(SP(J))=REPETITIONS(I)
      RVAR(SP(J))=REPVAR(I)
      SP(J+1)=SP(J)+1
      DO 1070 R=1, NORES
        STX(SP(J),R)=RESOURCE(I,R)
        STV(SP(J),R)=VAR(I,R)
1070  CONTINUE
      CURRENT=BRANCHNODE(I,1)
    ELSEIF (NODE.EQ.'DE'.OR.NODE.EQ.'BE') THEN
      signifies the end of a repetition loop,
      values for this loop are calculated.
      IF (NODE.EQ.'BE') THEN
        DO 1080 R=1, NORES
          STX(SP(J),R)=STX(SP(J),R)+RESOURCE(I,R)
          STV(SP(J),R)=STV(SP(J),R)+VAR(I,R)

```

```

1080      CONTINUE
      ELSE
      ENDIF
      J=J-1
      DO 1090 R=1,NORES
        STX(SP(J),R)=STX(SP(J),R)+S X(SP(J+1),R)*REP(SP(J+1))
        STV(SP(J),R)=STV(SP(J),R)+STV(SP(J+1),R)*REP(SP(J+1))
        +STX(SP(J+1),R)*STX(SP(J+1),R)*RVAR(SP(J+1))
1090      CONTINUE
      CURRENT=BRANCHNODE(I,1)
      ELSEIF (NODE.EG.'DT'.OR.NODE.EG.'BT') THEN
        terminus of parallel set
        BNO(BNP)=BNO(BNP)-1
        IF (BNO(BNP).EQ.0) THEN
          all branches have been considered
          BNP=BNP-1
          DO 1110 L=0,BNO(BNP)-1
            DO 1100 R=1,NORES
              STX(SP(J-1),R)=STX(SP(J-1),R)+STP(SP(J)+L)
              *STX(SP(J)+L,R)
              STV(SP(J-1),R)=STV(SP(J-1),R)+STP(SP(J)+L)
              *STV(SP(J)+L,R)
1100      CONTINUE
1110      CONTINUE
          DO 1120 R=1,NORES
            TEMP(R)=0.0
1120      CONTINUE
          DO 1150 R=1,NORES
            DO 1140 L=0,BNO(BNP)-2
              DO 1130 M=L+1,BNO(BNP)-1
                TEMP(R)=TEMP(R)+STP(SP(J)+L)*STP(SP(J)+M)
                *(STX(SP(J)+L,R)-STX(SP(J)+M,R))*2
1130      CONTINUE
1140      CONTINUE
              STV(SP(J-1),R)=STV(SP(J-1),R)+TEMP(R)
1150      CONTINUE
          BNP=BNP-1
          J=J-1
          CURRENT=BRANCHNODE(I,1)
          DO 1160 R=1,NORES
            STX(SP(J),R)=STX(SP(J),R)+RESOURCE(I,R)
            STV(SP(J),R)=STV(SP(J),R)+VAR(I,R)
            time and variance for this node added.
1160      CONTINUE
          ELSE
            pop next node and branch from stack
            SP(J)=SP(J)-1
            CURRENT=ST(SP(J))
          ENDIF
        ELSE
          the node type does not fit classification and
          therefore an error has probably occurred.
          WRITE (108,9010)
        ENDIF
      *
      * Print test results for debugging
      * DO 3000 R=1,NORES
      * WRITE(6,2999) (CURRENT,I,J,SP(J),STX(SP(J),R),STV(SP(J),R)
      * 2999 FORMAT(1X,'C= ',A,' I=',I3,' J=',I3,' SP(J)=',I4, /
      * : 10X,' STX(SP(J),R)=',G15.4,' STV(SP(J),R)=',G15.4)

```

```

*      IF(J.GT.0)THEN
*          WRITE(6,2997) SP(J-1),STX(SP(J-1),R),STV(SP(J-1),R)
*2997      FORMAT(10X,'SP(J-1)=' ,I4,' STX(SP(J-1),R)=' ,G15.4,
*          :          'STV(SP(J-1),R)=' ,G15.4)
*      ENDIF
*      WRITE(6,2998) BNP,BNO(BNP),BNO(BNP-1)
*2998      FORMAT(5X,'BNP= ' ,I3,' BNO(BNP)= ' ,I3,' BNO(BNP-1)= ' ,I3//)
*3000      CONTINUE
*      find I for currentnode
*
*      I=0
*      NOTFOUND=.TRUE.
*      REPEAT 1170,WHILE NOTFOUND
*          I=I+1
*          IF (NODENAME(I).EQ.CURRENT)THEN
*              NOTFOUND=.FALSE.
*          ELSEIF (I.GE.NUMBER.OR.CURRENT.EQ.'END')THEN
*              NOTFOUND=.FALSE.
*              GO=.FALSE.
*          ELSE
*              ENDIF
*1170      CONTINUE
*1180      CONTINUE
*      DO 1190 R=1,NORES
*          TOTALRESOURCE(R)=STX(0,R)
*          TOTVAR(R)=STV(0,R)
*1190      CONTINUE
*      RETURN
*5000      FORMAT(1X,'ERROR IN CALCU,BRANCHNODE')
*9010      FORMAT(1X,'ERROR IN CALCU,NODETYPE')
*      END

```

END

1 6 0 6 8 3

FIN