A COMPARISON OF SOME PERFORMANCE EVALUATION TECHNIQUES.

by

Sabah M. A. Mohamad.

Submitted for the degree of Master of Science to the Faculty of Science, University of Glasgow, Department of Computing Science.

September 1981.

In memory of my father and with much gratitude to my mother, who taught me the love of knowledge, started and always encouraged me in this pursuit, and thus provided the basis for development. Her efforts and sacrifices can never be fully repaid.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to

Dr. John C. Cavouras

for his time spent in guiding me in this research and for demonstrating that cooperative research can be so enjoyable. Without his unfailing energy and enthusiasm, I would not have been able to complete this thesis.
I wish to thank

Professor D.C. Gilles

Head of the department of computing science for giving me the opportunity to carry out this research in this esteemed department.
I would also like to thank both

Dr. J. Haselgrove and Dr. L. Smith

for their interest in my thesis.

Last, but not least, my thanks are due to the staff of the Iraqi Ministry of Higher Education and Scientific Research, who have helped me financially during this period.

Special thanks to my friends Shirley M. Stewart and Salim A. Al-Salim for their great help in typing this thesis.

Finally, a conductive atmosphere is important for research and all of the people associated with the Department of Computing Science at Glasgow University are to be commended for providing a professional but relaxed environment in which to work.

CONTENTS.

Abstract

In this thesis we look at three approaches to modelling interactive computer systems: Simulation, Operational analysis and Performance-Oriented design.

The simulation approach, presented first, is applied to a general purpose, multiprogrammed, machine independent, virtual memory computer system. The model is used to study the effects of different performance parameters upon important performance indices. It is also used to compare or validate the results produced by the other two methods.

The major drawback of the simulation model (i.e. its relatively high cost) has been overcome by combining regression techniques with simulation, using simple experimental case studies.

Next, operational analysis was reviewed in a hierarchical way (starting by analysing a single-resource queue and ending up by analysing a multi-class customer general interactive system), to study the performance model of general interactive systems. The results of the model were compared with the performance indices produced using the simulation results.

The performance-oriented design technique was the third method used for building system performance models. Here, several optimization design problems have been reviewed to minimize the response time or maximize the system throughput subject to a cost constraint. Again, the model results were compared with the simulation results using different cost constraints.

We suggest finally, that the above methods should be used together to assist the designer in building computer performance models.

CHAPTER 1

INTRODUCTION

1.1. Motivation.

1.2. Outline of the Thesis and Summary.

1.2.1. Chapter 2.

1.2.2. Chapter 3.

1.2.3. Chapter 4.

1.2.4. Chapter 5.

1.2.5. Chapter 6.

1.1. Motivation:

Since the early days of the computer industry, there has been considerable interest in the design and performance analysis of systems. The goal has most often been to obtain better insight into their behaviour and to improve their performance.

During the last decade, we have seen the development of a large number of computer systems. In most cases, these systems have failed to meet the performance objectives predicted during the initial design. During the same period, "the complexity of these systems has increased tremendously with the introduction of multiprogramming, multiprocessing, virtual memories, etc. It has thus become more difficult to understand the behaviour of these systems in a qualitative sense, let alone ((/Muntz 75/)). Hence, the road to understanding the behaviour and predicting the performance of computer systems has been, and still is, arduous. Many people have realised this and have attempted to investigate the problem of designing and analysing the performance of computer systems, and to proceed to develop superior tools. Such a tool can most generally be represented in the schematic diagram of Figure 1.1.:

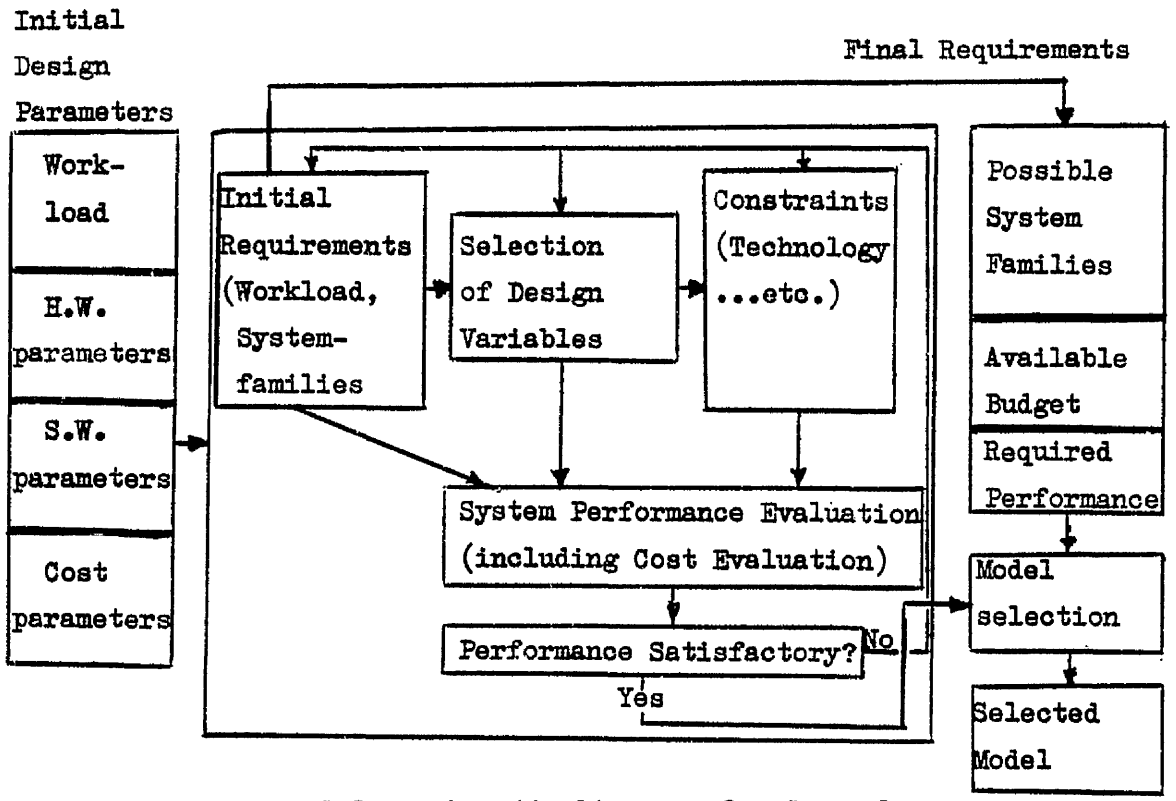


Figure 1.1. Schematic Diagram of a General Tool for Computer System Performance Design and Evaluation.

The design and evaluation diagram shown represents a hierarchical and iterative process. This process starts by selecting the initial design parameters (i.e. input performance parameters) and the initial user requirements and constraints (these may represent the system families, required cost, technology constraints, required system cost, etc.). After that the process takes an iterative shape to select the design variables (these may include many changes in the initial requirements and constraints). In this selection, the evaluation of the system cost and performance plays a critical part. The iterative process will produce several models that satisfy the initial requirements and constraints. The process then enters a decision area to select the 'best' model among the produced models, depending on the final user requirements.

The purpose of such system design and evaluation tools is generally in three parts ((/Lucas 71/)):

* Selection evaluation:
  Selection evaluation plans to include performance as a major criteria in the decision to obtain a particular system from a vendor.
* Performance projection:
  Performance projection is oriented towards designing a new system, either a hardware component or a software package. The goal here is to estimate the performance of a system that does not yet exist.
* Performance monitoring:
  Performance monitoring provides data on the actual performance of an existing system. This data can be used to forecast the impact of changes in the system, such as a reconfiguration of the hardware or an improvement in the frequently executed software modules. Such evaluation may also be concerned with obtaining a profile of the use of a system, in order to make strategic decisions, for example, on the characteristic of a job priority system.

The designed evaluation techniques used for the three purposes are fully discussed in chapter 2. The selection of a particular technique(such as Simulation, Benchmarks, Monitors...etc.) depends

on the suitability of that technique for a given purpose.

The concentration on both design and evaluation techniques is quite important, since it has been proven that "design without evaluation usually is inadequate" ((/Cantrell and Ellison 68/)). This combination always provides better systems, better understanding of the system operations and the effects of each performance factor. It also helps in tracing the performance bugs. Finally, it removes the 'faith' concept in designing a computer system. The problem is a scientific and engineering one only, if it is solved using both performance design and evaluation techniques.

1.2. Outline of the Thesis and Summary:

The aim of this thesis is to show that different performance design and evaluation tools can be combined in such a way as to help the designer in building better computer performance models. This idea is quite important, since there is no single best way to design a computer system.

In this thesis we introduce three modelling techniques which can be combined to construct a more reliable performance model. These techniques are:

* Simulation,
* Operational analysis and
* Performance-oriented design.

The above techniques were selected from many available techniques. The reason for such selection and a brief review of the available techniques are introduced in the second chapter. In the next three chapters we introduce each technique seperately. In the last chapter an implementation of the combined ideas is given.

The following is a brief summary of the contents of the remaining chapters of the thesis.

1.2.1. Chapter 2: "Computer Design and Evaluation Methods".

In this chapter the available computer design and evaluation methods are critically reviewed. These methods are:

* Analytical Methods.
* Simulation Methods.
* Empirical Methods.

According to certain factors a specific set of design and evaluation methods have been chosen to help the designer to solve

future problems with different levels of details and accuracy. The methods chosen are:

* Simulation.
* Operational analysis.
* Performance-Oriented design.

1.2.2. Chapter 3: "The Simulation approach".

An ideal simulator should incorporate the software and the hardware of the system under design. Some researchers call this incorporation "the forth generation computer system concept". A general simulation tool (GST) was presented by Cavouras ((/Cavouras 78/)) to represent this aim. The GST is reviewed in a structured way.

Since Simulation is a very expensive approach for system design and evaluation, we have tried to overcome this by introducing regression analysis techniques to the results of the Simulation in order to produce fast hybrid models. This was done through several case studies and the introduction of an interactive design tool (IDT) is suggested.

1.2.3. Chapter 4: "The operational analysis approach".

In this chapter we aim to represent a similar general interactive computer system as the GST introduced in chapter 3 using the operational analysis technique. For this purpose the operational analysis technique was critically reviewed. It has then been used to represent a general multi-class customer interactive computer system. Many factors have been investigated during the representation process. These include:

* job flow balance.
* load-dependent behaviour.
* homogenous service times.
* decomposition technique.

Finally, we tried in this chapter to concentrate on the representation of the effects of both the hardware and software parameters on the model.

1.2.4. Chapter 5: "The performance-oriented design approach".

In this chapter we have also tried to represent a general interactive computer system similar to the GST model. This was done by reproducing the work of several researchers. Several optimization problems to minimize the response time or maximize the system throughput of the modelled system, subject to a cost constraint, are examined.

1.2.5. Chapter 6: "Comparisons and conclusions".

We conclude our research work by giving an overview of the work presented and give an example to implement the combining of the three discussed modelling techniques. Finally, we suggest several future research ideas and extensions to this work.

CHAPTER 2

COMPUTER SYSTEM DESIGN AND EVALUATION METHODS

2.1. Introduction.
2.2. Analytic Methods.
  2.2.1. Operational Analysis.
  2.2.2. Stochastic Analysis.
  2.2.3. Mean-Value Analysis.
2.3. Simulation Methods.
2.4. Empirical Methods.
2.5. Other Methods.
  2.5.1. Performance-Oriented Design.
  2.5.2. Benchmarking.
2.6. Conclusions.

2.1. Introduction:

For several years immediately following their invention, computers were almost universally monoprogrammed.[1] However, it was soon realised that following more than one program to run concurrently would result in more effective usage of the system's resources, since one program could be using one resource, while another program could be using a different resource. With the advent of operating systems to manage concurrently running programs, multiprogramming became a reality ((/Bouhana 78/)).

In an attempt to understand and quantify resource usage and concomitant delays that result when programs compete for service in a multiprogrammed environment, performance analysts have constructed several representative methods to model computer systems. "Studying these methods are of a vital importance in the system design and evaluation process" ((/DeCegama 72/)).

Grenander and Tsao ((/Grenander and Tsao 72/)) suggest that these quantative methods of design and evaluation of computer systems fall into three categories, namely:

1. Analytical Methods.
2. Simulation Methods.
3. Empirical Methods.

To apply computer performance methods, there are a number of considerations and problems of which the user must be aware. Different techniques are required for different computer systems measurement ((/Goh 76/)). Also, there is no single tool or method which is capable, by itself, of evaluating all elements of a system. The nature of the questions to be answered will influence the choice of a technique, or techniques. The user must develop some criteria for the selection of appropriate performance assessment methods. The criteria to be considered include:

* Understandability.
* Cost.
* Degree of resolution (accuracy).
* Ease of parameter optimization or estimation.
* Breadth of applicability.
* Relevance to actual system.

In order to highlight these methods and their differences, we will try to study them in the following sections.

---

[1] Only one program could be running in the computer, and that program had exclusive use of all the systems' hardware and software resources for the duration of its running time.

2.2. Analytical Methods: (Non-Deterministic Modelling).

Analytical models represent system performance parameters strictly in mathematical terms. Simpifying assumptions may be used to avoid unnecessary complexity and to keep the mathematics tractable, provided that the necessary simplifications of the analytic model still preserve the important characteristics of the computer system which is to be evaluated.

Many computer system analysts prefer this approach, mainly for the following reasons:

* It is an economical method compared to simulation.
* It can be used to optimise the design variables, whereas the the number of simulation runs required to accomplish the same task will be high.
* It is quicker to produce results than by simulation.

This approach, however, may have the following disadvantages:

* Limited in scope.
* Difficult to develop and build.
* Not easy to test the simplification assumptions.

"Queueing theory has been employed widely for the performance evaluation of various classes of computer systems. The models include closed and open queueing networks, the treatment of various customer classes, and approximations which relax some of the restrictions necessary for the application of queueing theory"((/Von Mayhauser 79/)) The queueing network theory has been used by all available analytical methods, namely:

* Operational Analysis.
* Stochastic Analysis.
* Mean-Value Analysis.

Hence, the knowledge of queueing theory is essential in understanding any analytical tool. This theory was previously investigated by many researchers and for further information of this theory the reader is referred to: ((/Kleinrock 75, 76/)) ((/Murdoch 78/)).

2.2.1. Operational Analysis:

"Operational Analysis is a framework for studying the performance of systems during given periods of time. The system may be real or hypothetical, and the time may be past, present or future" ((/Buzen and Denning 80/)).

This kind of analysis was recently invented, about 1976 ((/Buzen 76/)), to construct a precise mathematical tool to meet the following objectives:

1. Relate existing measurement data to other quantities that were not measured but which could, in principle, be empirically determined.
2. Verify the internal consistency of existing sets of measurement data.
3. Predict the effect that certain modifications to the system or the workload would have on measured quantities.
4. Be simple and easy to understand.
5. The tool should be based on testable assumptions.

The general idea of operational analysis (or operational method) can be shown in the following diagram (see Figure 2.1.):

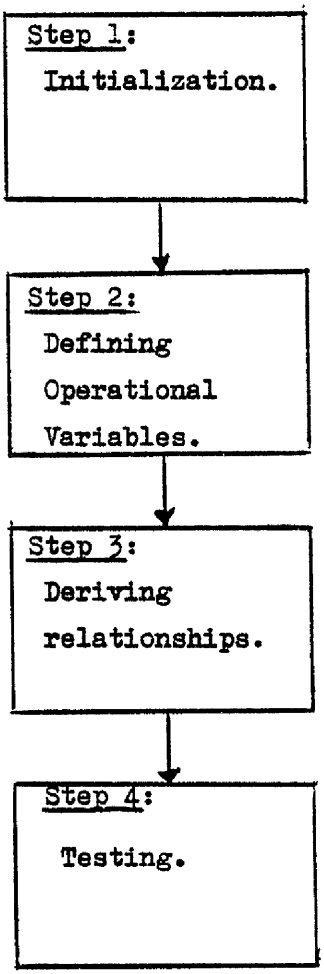


Figure 2.1. Operational Method.

Step 1: Initialization.

In this step an observation interval is obtained: an interval of time during which system behaviour is monitored and measurement data is collected. The measured or computed quantities within the observation interval are called operational variables.

Step 2: Defining Operational Variables.

Defining the operational variables that directly affect the performance indices of interest.

Step 3: Deriving Relationships.

The behaviour of the system is specified in this step by deriving the relationship between the operational variables. These relationships are represented by mathematical equations.

Step 4: Testing.

At this step, the mathematical relationships are tested against the original objectives.

This method is considered by many researchers as equivalent or as an alternative to the traditional method of Stochastic analysis (or Stochastic modelling). ((/Buzen 76/)), ((/Buzen 78/)) ((/Buzen 76a/)) ((/Denning and Buzen 78/)). Other researchers find that this approach has several advantages to the traditional approach. These advantages can be summarised as follows ((/Sevcik and Klawe 79/)):

* Relevance to actual system: The fact that operational analysis is based on observable quantities and testable assumptions makes it easier to relate to system measurements.
* Understandability: Operational analysis can be understood easily , even for large systems.
* Breadth of applicability: Since operational analysis depends on testable assumptions, it has a wide applicability as a modelling technique. Its major application areas are ((/Denning and Buzen 78/)):

a. Performance Calculation:
Operational results can be used to compute quantities which have not been measured.

b. Consistency checking:
A failure of data to verify a theorem or identity reveals an error in the data, a fault in the measurement procedure or a violation of a critical hypothesis.

c. Performance Prediction:

Operational results can be used to estimate performance quantities in a future time (or indeed a past time) for which no directly measured data are available.

* Testability of Assumptions: Most of the assumptions of Stochastic analysis can neither be verified nor disproven in any finite period. While the assumptions of operational analysis can, in principle, be tested in finite time intervals.

However, some researchers do not find this approach suitable for parameter estimation and anticipated design and modification ((/Muntz 79/)) ((/Sevcik and Klawe 79/)). Buzen ((/Buzen 79/)) believes that, "the estimation problem is not really an integral part of either operational analysis or stochastic modelling. It is crucially important but an entirely separate issue". At the same time, Buzen believes that the performance analysis offers major advantages over stochastic modelling in performance prediction.

Operational analysis may use queueing theory, in which case it is called Operational queueing network theory ((/Denning and Buzen 77/)).

The important reason why queueing theory should be used, is the speed with which performance quantities are computed using queueing network formulae. The operational queueing network theory may use some assumptions - e.g. flow balance, one-step behaviour and homogeneity, but these assumptions (as mentioned previously) can be tested for validity in any observation period.

2.2.2. Stochastic Analysis:

This analysis depends on queueing theory: it considers the system as consisting of service centres among which customers circulate. This analysis may also be called stochastic modelling or Probabilistic modelling, since the servicing time of a customer at a servicing centre is taken to be a sample from a specified distribution and the frequency by which the customer will move to another servicing centre is controlled by a specified probability distribution.

Let us now introduce the following:

1. Definition: ((/Ferrari 78/))

A stochastic process X(t) is a function of time t whose values are random variables. The value of X(t) at time t^* represents the state of the stochastic process at t^*. If each random variable may only take on a finite or a countable number of values, we have a discrete-state process or chain. Otherwise, we have a continuous-state stochastic process.

2. Hypothesis:

The behaviour of the real system (or the queueing network model) during a given period of time is characterized by the probability distributions of stochastic process if and only if the following assumptions hold ((/Sevcik and Klawe 79/)):

(a.) Successive service times are independant.

(b.) Successive transitions among service centres are independant.

(c.) The process is ergodic.[1]

(d.) The system reaches equilibrium.

3. Condition 1:

If (a) and (b) was assumed and if service time distribution at each centre is exponential then the system state (number of customers at each centre) is a continuous Markov process.

Note:

Markov modelling is important, because it forms the basis of elementary queueing theory. Readers not familiar with this theory are referred to /Kobayashi 78/.

4. Condition 2:

If (c) and (d) were assumed then the system is at a steady-state equilibrium, and the long term performance measures can be computed.

Based on the above, we can construct a stochastic model. Observable aspects of the real system - e.g. states, parameters, and probability distributions - can be identified with quantities in the stochastic model and equations relating these quantities can be derived. Although formally applicable only to the stochastic process these equations can also be applied to the observable behaviour of the system itself, under suitable limiting conditions ((/Buzen 78/)). The parameters of the stochastic process, representing the operation

---

[1] The system is ergodic means long-term time averages converges to the mean values for stochastic equilibrium.

of the system, must be estimated from observations during a finite time interval. The specific formulae depend on what measurement data is available and on the amount of detail in the queueing network model.

In order to validate the model, the estimated parameter values are plugged into the performance measure formulae, and the results are compared to the corresponding observed values in the observation period. The most common purpose for which models are created is to obtain an indication of how a system will behave in the future, either after its configuration has been altered or its workload has been changed. In order to accomplish this, it is possible to employ the same computational formulae as in the validation of the model, by using modified parameter values in order to reflect the altered circumstances anticipated in the future. Once the future values of the model parameters have been estimated, the obtained formulae are used to calculate the performance measures. These are then interpreted as equilibrium performance measures of a stochastic process.

Stochastic analysis has, however, certain disadvantages ((/Denning and Buzen 78/)):

1. It is impossible to validate the stochastic hypothesis and conditions, hence an analyst can never be certain that an equation derived from a stochastic model can be correctly applied to the observable behaviour of a real system.
2. Stochastic analysis is an <u>inductive</u> mathematical tool:(it estimates unknown values from the projection period from values observed in the baseline period). Thus, one faces the problem of uncertainties in estimation of variables. (<u>Note</u>: this problem is not present in operational analysis, since operational analysis is a <u>deductive</u> mathematical tool).
3. Stochastic analysis can be applied to study a <u>special</u> class of computer systems because the type of assumptions used by this analysis cannot be easily found in real systems (e.g. the assumptions of equilibrium or stochastic independance of successive service times).
4. Stochastic modelling may not be so easy to understand.
5. Stochastic modelling cannot be relevant to a real system. For example, in real systems transactions between devices do not follow Markov chains or processes, and service time distributions are not generally exponential ((/Von Mayhauser 79/)).

On the other hand, Stochastic models bestow certain benefits. Independant and dependant variables can be defined precisely, hypothesis can be stated succinctly and a considerable body of theory can be called on during analysis ((/Denning and Buzen 78/)).

Finally, the relationship between Stochastic analysis and operational analysis is given by figure 2.2. ((/Buzen 78/)).

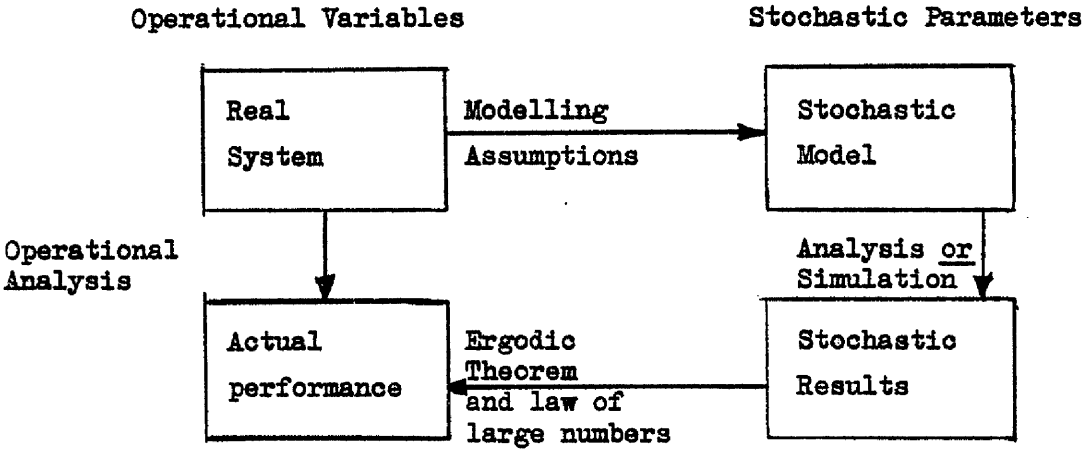


Figure 2.2. The relationship between operational analysis and Stochastic modelling.

2.2.3. Mean-Value Analysis:

This is a new mathematical tool, used to calculate some important performance indices, such as mean response time, throughputs and queue length in closed queueing networks. A primary advantage of mean-value analysis over the traditional approach (i.e. Stochastic Analysis) , is its improved numerical stability ((/Buzen and Denning 80/) This analysis uses the Sevcik & Mitrani ((/Sevcik and Mitrani 78/)) arrival theorem to calculate the mean-value for successively larger loads N.

Reiser ((/Reiser 79/)) found queueing networks with product-form solution[1] remarkably robust[2] with respect to routing and service-time distributions. This robustness leads to the new mathematical explanation called Mean-Value Analysis.

---

[1] Product-form solution: gives the joint queue-size up to a normalization constant. This constant has a simple analytic expression in the case of open queueing networks but is a sum of product terms of closed system.

[2] Robust: Statisticians call a system robust if only the mean enters into the solution.

Mean-Value analysis uses some basic equations which can be applied iteratively for any value of N.

Let i = device number, K = number of devices.

N = number of jobs.

Q_i = overall mean queue length at device i.

Q_{Ai} = mean queue length seen by arriving customer at device i.

$R_i(N)$ = mean response time of device i, i = 1,...,K, given N jobs.

$R_o(N)$ = mean response time of the system given N jobs.

$X_o(N)$ = mean system throughput given N jobs.

V_i = mean number of visits per job to the device i.

$S_i(N)$ = mean time between completions when the state of the system is equal to N.

def

$Q_{Ai}(N) = Q_i(N-1)$.....Sevcik-Mitrani theorem.

Then the basic mean-value equations are

$$\boxed{R_i(N) = S_i\ (1 + Q_i(N-1))} \qquad \text{.....(1)}$$

where i = 1 ,...,K.

and

$$\boxed{X_o(N) = N \Big/ \sum_{i=1}^{k} V_i R_i(N)} \qquad \text{.....(2)}$$

Using the forced flow law, we get

$X_i(N) = V_i X_o(N)$ ....forced flow law.

where $X_i(N)$ = throughput at device i given N

we get

$$\boxed{Q_i(N) = R_i(N) V_i X_o(N)} \qquad \text{.....(3)}$$

where i=1,...,K.

Equations(1),(2) and (3) can be used iteratively, once the values V_i and S_i are given. The iteration begins with N=1 and the boundary condition $Q_i(o)=0$.

It is clear that this type of analysis uses no normalization constant to calculate the important performance indices, and hence the formulae have a simple mathematical structure. This criteria is not available in the two previous analytical methods, i.e. Operational analysis and Stochastic analysis.

Some ideas of extending Mean-Value analysis were given by Buzen and Denning ((/Buzen and Denning 80/)) and by Riser and Lavenberg ((/Riser 81/)) ((/Riser and Lavenberg 80/)), which the reader is referred to for further information.

2.3. Simulation:

"Simulation has been defined as an evaluation and design technique which represents, by a model, the behaviour of a system in the time domain. The observation of the behaviour in time of the systems model, under stimuli generated by a model of the system's inputs, produces numerical results which may be used in evaluation studies. A model suitable for this purpose is called a Simulation model or simulator" ((/Ferrari 78/)). Simulation is applicable whenever we have a certain degree of understanding of the process to be simulated. The ideal simulator should meet specific requirements ((see/Cavouras and Davis 81/)).

Simulators can be classified as shown in Figure 2.3.

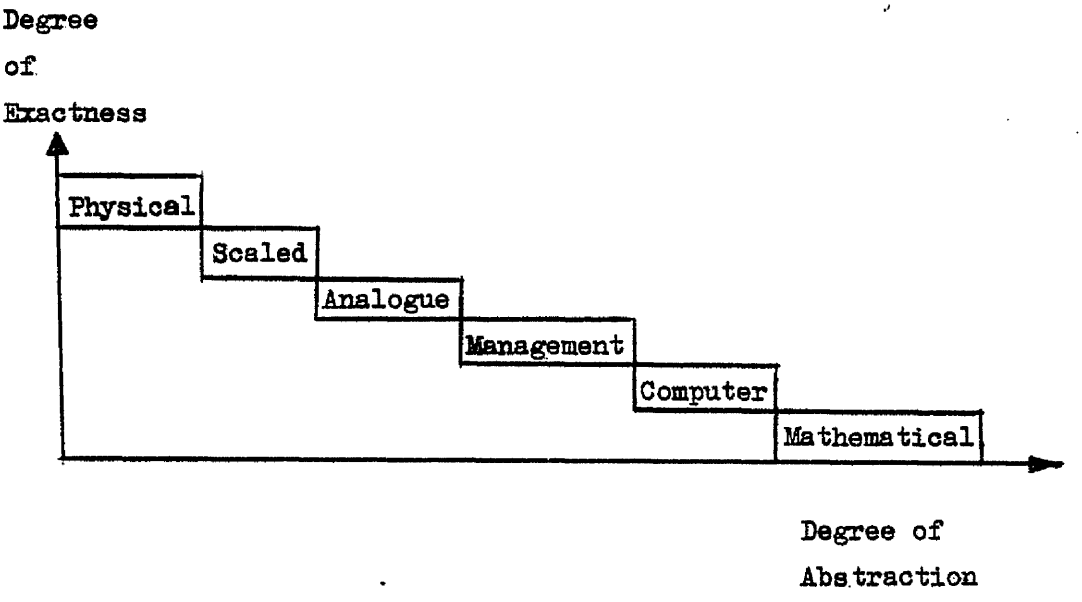


Figure 2.3. Classification of Simulation Models.

Simulation models can be thought of(and hence classified) in a continuous spectrum, starting with exact models of reality and proceeding to completely abstract mathematical models ((/Shannon 75/)).

Although Simulation is an excellent method, many analysts avoid it because the development of a good simulation model is often expensive and time consuming. Many researchers tried to overcome this problem, using approaches such as the following:

1. General Simulation Model: ((/Goh 76/))
   Here the idea is to design an "extensible" simulation model as a general simulation model, which can then simulate any specific models easily.
2. Structured Approach:
   This approach takes the view that the process of developing a simulation model should pass through the following stages (see Figure 2.4.) ((/Mirham 72/)):

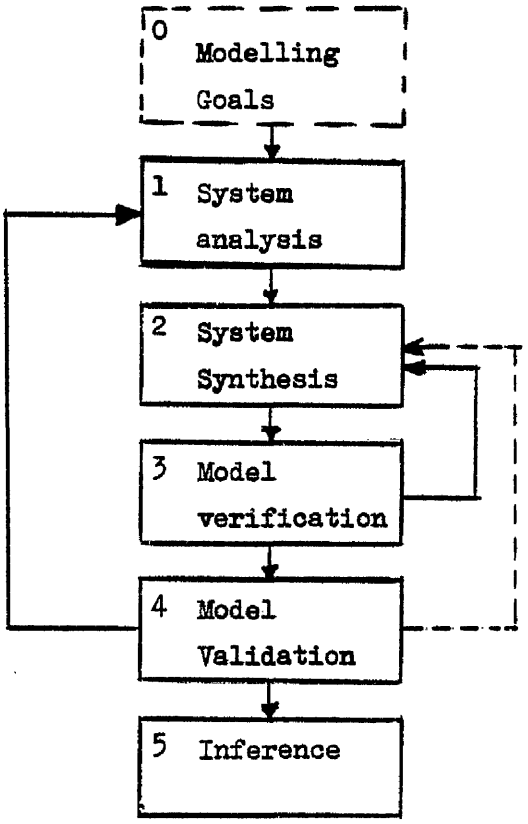


Figure 2.4. Simulation Model Development Stages.

Stage 1: System Analysis.

The initial stage of development, during which the salient components, interactions, relationships and dynamic behaviour mechanisms of a system are isolated.

Stage 2: System Synthesis.

The stage of development, during which the model of the system behaviour is organized in accordance with the findings of the proceeding system analysis stage, and during which appropriate data is delineated and collected.

Stage 3: Verification.

The third stage of development, during which the

model's responses are compared with those which would have been anticipated if the model's structure was prepared as intended.

Stage 4: Validation.

The stage of development during which the responses emanating from the verified model are compared with corresponding observations of, and measurements from, the actual system in order to establish the verisimilitude of the model and the modelled system.

Stage 5: Inference.

The final stage of development, concerned with the definition of experiments with, and comparison of the responses from the verified and validated model.

The structured approach represents a modelling method, which again requires a 'good' design methodology. Cavouras ((/Cavouras 78/)) argued that his Simulation modelling methodology (or approach) is more realistic than the available methodologies. Cavouras' approach is mainly based on the fundamental requirement that a simulation tool should have the same logical structure as the software being modelled, and the method proposed was to embed the supervision of a computer operating system in a simulation of its environment, so that the overall system performance can be measured by direct experimentation ((/Cavouras and Davis 81/)).

Simulation provides an accurate model, but it may require an inordinate amount of time for the determination of the system performance. In the same sense, simulation is very expensive, especially when we want to use it to optimise the future behaviour of a system. There is, in fact, very few models which tried to overcome the optimization problem, for example the SCERT (System and Computer Evaluation and Review Technique) simulator ((/Ihrer 67/)).

2.4. Empirical Methods:

These methods represent an alternative to the modelling techniques described by the last two sections. These methods are appropriate when performance data of (an) actual system(s) are available. Statistical methods use these data to forecast future performance.

Empirical performance results can be obtained through measurements.Measurements may be from an actual system or from a model of a system. The collection of these measurements can be performed with hardware monitors, software monitors and

accounting packages.

The need for performance measurements can arise out of different situations. Lucas ((/Lucas 71/)) suggested three general reasons for undertaking performance evaluation (i.e.modelling and measurements), namely:

1. Selection Evaluation - choosing from a set of new possible alternatives - which system best meets a user's cost / performance specifications.
2. Performance Projection - estimating the performance of a system which is not yet implimented, i.e.an aid in the system design.
3. Monitoring - forecasts the impact of possible changes of the software components or the user load applied to the system, i,e. system tuning or balancing.

Approximately, the same reasons were presented by Grenander and Tsao ((/Grenander and Tsao 72/)).

The major applications of performance measurements are summarized as follows ((/Buzen 77/)):

a. Accounting:
   Since the changes for running a program are typically based on the resources used by that program, sub-routines for measuring CPU time, I/O operations, memory requirements, and so on, are an integral part of most accounting packages. In addition, using measurements as inputs to changing algorithms, accounting packages often make basic measurements of data available in row form for other purposes.
b. Trend Analysis;
   Many data processing centres maintain graphs or tables of performance measurement data which has been aggregated on a daily, weekly or monthly basis. Data of this kind can be of great value to managers and planners who wish to examine trends in workload growth, identify peak periods and cycles, and attempt to determine when a system is 'running out of capacity', (in the sense of requiring an upgrade in order to maintain acceptable levels of service).

c. Tuning:

Careful examination of measurement data often leads to the discovery of imbalances and inefficiences within a system. Frequently, these problems can be readily corrected, and a dramatic improvement in overall performance can be obtained. This is generally the case when gross imbalances are found in the loads on different I/O devices or channels, and when inefficient search algorithms (e.g. linear search) are replaced by more efficient algorithms.

d. Evaluation of Changes:

The use of measurement data for trend analysis and tuning leads, naturally, to a desire to use measurement data for the evaluation of various changes to a systems hardware, software or workload. For example, managers and planners often need to determine the performance impact of changes such as installing a higher performance CPU, more main memory, or larger discs. Similarly, system programmers involved in tuning may be interested in the performance impact of a new swapping algorithm, a change in the ammount of memory allocated to the operating system, or reassignment of priority levels among various classes of work.

However, it is often difficult to obtain accurate measurements of a particular quantity of interest due to inadequate system instrumentation, or due to gross interference caused by the measurement technique((/Adams 78/)).

It has been suggested that the best way to use measurement as a system evaluation technique is to connect both measurement and evaluating models (simulator or analytic model) in one process. This has been employed by Noetzel ((/Noetzel 71/)) in his meta-system (see Figure 2.5). (next page)

The reader interested in measurement techniques is referred to ((/Goodman 72/)) ((/Brad 71/)) ((/Chouinard 76/)) ((/Calingaert 67/)) ((/Kimbleton 72/)) ((/Williams 72/)).

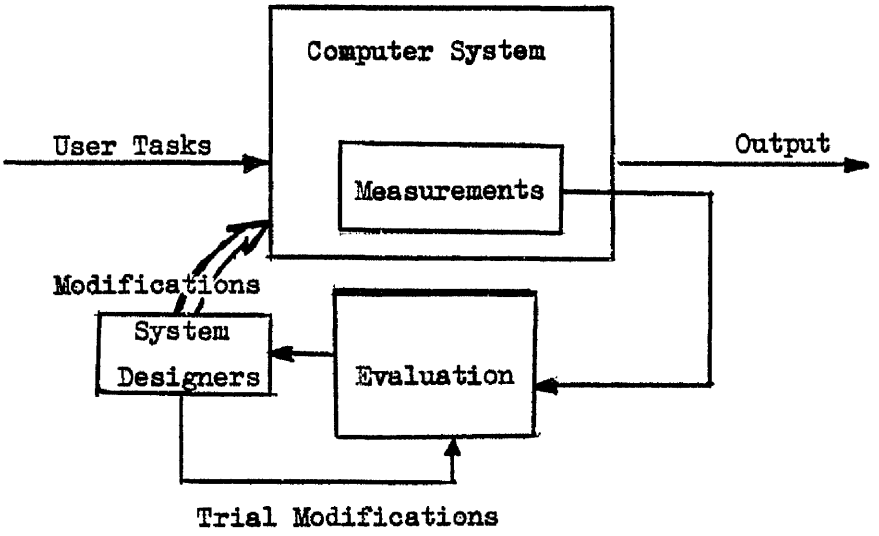


Figure 2.5. The Meta-System.

2.5. Other Methods:

To complete the list of the techniques used in design, measurement and evaluation we have to include:

2.5.1. Performance- Oriented Design Technique.

2.5.2. Benchmarking Technique.

2.5.1. Performance-Oriented Design Technique:

This has been summarized by Sigmon ((/Sigmon 79/)) as follows: This method can be used to aid computer science designers, by generating initial system designs for an iterative design process or by allowing the investigation of many different system configurations, quickly and inexpensively. The design models are based on queueing networks upon which an optimization problem has been superimposed. The objective of these optimization problems is to optimize a system performance index, such as throughput, subject to a cost constraint.

In fact this method is not quite new, since the idea of using optimization in system design has been used by several researchers ((/Decegama 70/)) ((/Irani and Uppal 72/)).

2.5.2. Benchmarking Technique:

Benchmarking represents another alternative to modelling which has been in use since the earliest days of computing.

Benchmarking can be regarded as a performance calculation proceedure in which the system itself performs

the calculation by actually processing the workload on the hardware under the control of the software. The reader is referred to ((/Benwell 75/)) and ((/Sime 73/)) for more details.

Performance-Oriented design is a good way to estimate the future design according to many given constraints, such as cost, workload, technology...etc. The only problem of using this method is that we have to choose a limited number of design variables to keep the optimization problem mathematically tractable.

Benchmarking has major difficulties ((/Buzen 77/)) and it is considered impractical.

The following diagram (figure 2.6) lists the available design and evaluation tools:

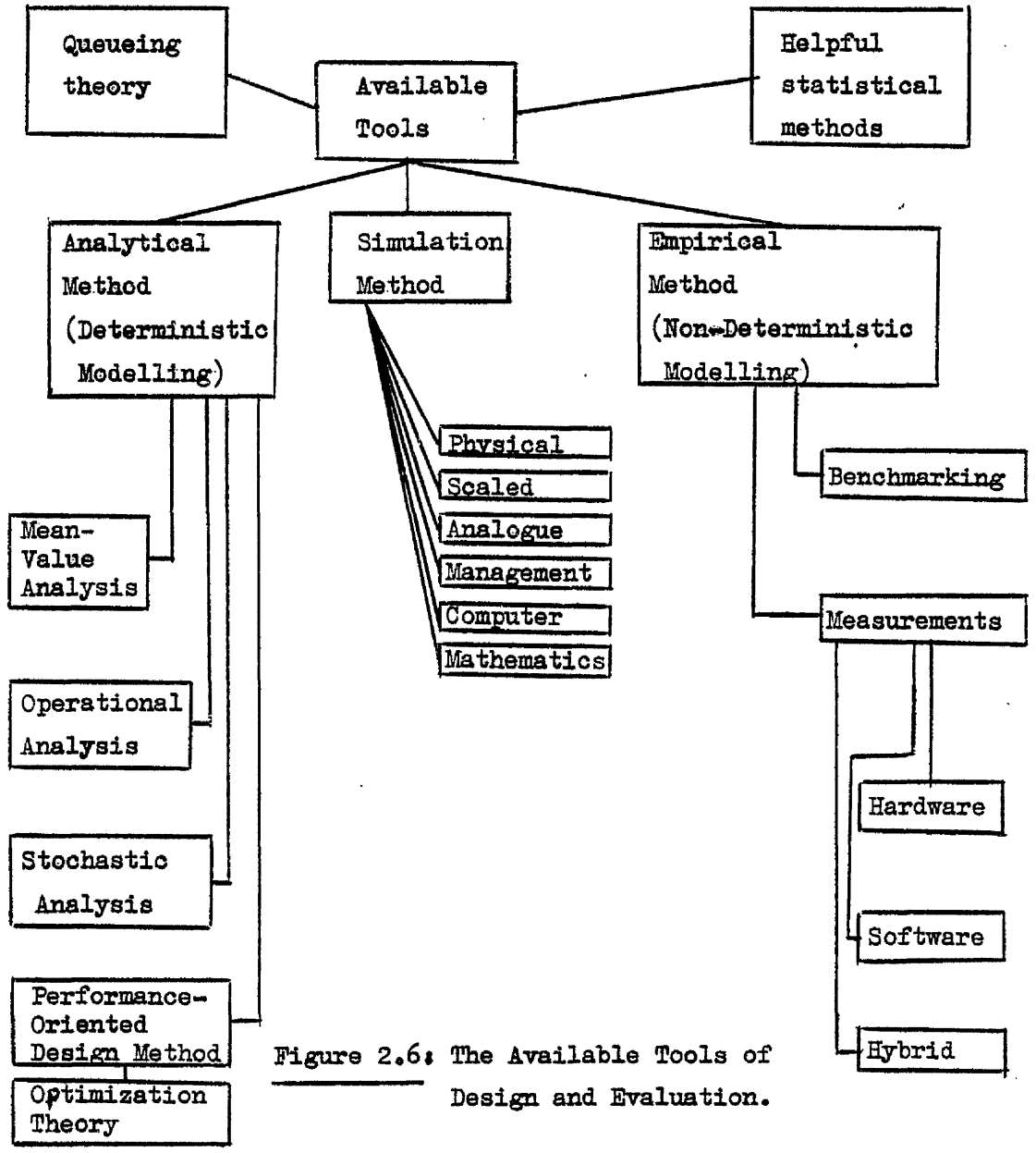


Figure 2.6: The Available Tools of Design and Evaluation.

2.6. Conclusion:

In this chapter several design and evaluation methods were introduced, and it was argued that each method has its own characteristics and advantages, therefore there is no single best way to design and evaluate a computer system. In fact, a designer should deal creatively with the problems he faces.

Due to the enormous task facing a system designer, it was decided to concentrate on a few important design and evaluation tools that can help the designer to attack the problem, on any level of detail and accuracy he wishes.

The selected design and evaluation methods are:

* Simulation.
* Operational Analysis.
* Performance-Oriented Design.

The selection has been made according to the criterion and factors mentioned in Section 2.1., except for the cost factor of the simulation technique which is quite an expensive method. This problem will be overcome in the third chapter, by combining the simulation with the regression analysis to produce lower cost models.

The selected methods will be studied in the following chapters.

CHAPTER 3

THE SIMULATION APPROACH

3.1. Introduction:

Broadly speaking, the performance of a computer system is determined by its hardware (speed, capacity,...etc.), the demand (job types, arrival patterns,...etc.) placed on it and by scheduling strategy employed (the order in which jobs are executed) ((/Coffman and Matrani 75/)). These three important performance elements represent the characteristics of the system hardware and software. Thus, any ideal system design and evaluation tool should include these characteristics.

Many of the existing tools, however, do not represent the software and hardware system characteristics. As an example, the conventional analytic approach which views computer systems as configurations of static hardware resources (CPU's, memories, I/O channels and devices) and user jobs or tasks as dynamic entities that flow through these configurations. This approach ignores the important characteristics and effects, especially those of the software. Moreover, some analytic researchers ((/Kumar and Gonsalves 79/)) try to solve the problem of the software representation by reversing the conventional approach — they view a computer system as a configuration of static software modules, and the processors that execute this software the dynamic entities that flow through this configuration. But again, this idea cannot lead us to construct an ideal tool. What we actually need is a tool which can incorporate the software and the hardware of the computer system at the same time. Some other researchers go further and represent the incorporation level as a fourth generation computer system concept ((/Rakoczi 69/)).

The incorporation level has been solved using simulation techniques which can represent the same logical structure of the software being modelled, and its hardware. Simulation offers a way to evaluate a system with relative accuracy prior to its development. By varying design parameters, the system designer can hope to identify potential bottlenecks, avoid costly design mistakes and estimate some of the guess work of identifying the most suitable system solution. Many researchers ((/Von Mayrhauser 79/)) ((/Ferrari 78/)) find simulation a very expensive approach if used as a tool for system modifications and evaluation. Thus we will try in this chapter to overcome this problem by combining the regression analysis techniques with the simulation to produce simple hybrid models.

Moreover, designing a very detailed simulator which satisfies the requirements of an ideal design and evaluation tool is not an easy task. We find it is more convenient for the purpose of this research work, to use an available ideal simulator. In fact, this type of simulator was presented by Cavouras ((/Cavouras 78/)) in which the supervisor of a computer system was embedded in a simulation of its environment so that the overall system performance can be measured by direct experimentation. Hence, we can consider this simulator as a model to a fourth generation computer system, since it incorporates the software and the hardware of the system being modelled. Besides that, it has been argued that this simulator satisfies the ideal and evaluation tool requirements ((/Cavouras and Davis 81/)).

For the above reasons we have chosen Cavouras' simulation tool as a basis to study the simulation design and evaluation techniques. This simulator will be called, throughout our research work, General Simulation Tool (GST).

The next section of this chapter requires a knowledge in operating systems, and for this purpose the reader is referred to ((/Hansen 73/)) ((/Bayer, Gratam and Seegmüller 78/)) ((/Watson 70/)).

3.2. The GST Simulation Method: ((/Cavouras 78/))

3.2.1. Generalities:

The GST represents a method of constructing a tool for general purpose,multiprogrammed, virtual memory computer system. The GST consists of a two level simulation; a simulation within a simulation. The inner simulation models the execution of the user processes. The outer simulation is partly driven by the former and partly by itself (includes the interrupts and the system processes) to model the overall behaviour of the system (see figure 3.1.).

GST Overall Structure

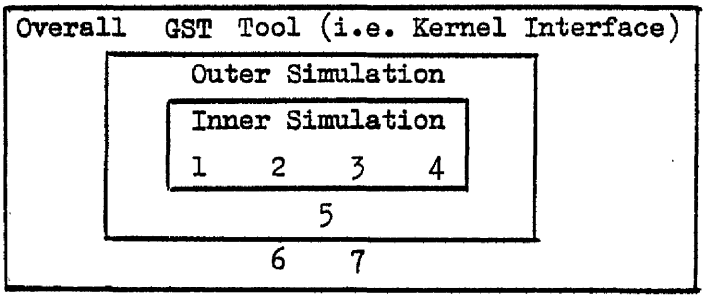


Figure 3.1. The GST Design and Evaluation Tool.

GST Routines:

1: Remove and insert in event list.

2: Remove and insert in an interrupt list.

3: Interrupt event routines.

4: Dispatcher.

5: A re-entrant coroutine program for all user process with its own "remove and insert in event list(s)" routines.

6: One coroutine for each system process (some of these coroutines are exact replicas of the corresponding system processes).

7: A routine which traps primitive calls and schedules events.

In general, GST is based on the concept of replicating the real system supervisor by embedding it in a simulation of its environment so that the overall system performance can be tested by direct experimentation.

Briefly, GST consists of the following modules:

1. An uninterrupted kernel interface which has one-to-one correspondance with the real system events (traps and interrupts).
2. A re-entrant coroutine program which independently models the execution of all user processes.
3. A set of coroutines, one for each system (supervisor) process in the system.
4. Routine which accepts (traps) primitives (supervisor) calls from the above coroutines and converts them into scheduled events before returning to the overall system simulation in 1. above.

The above four modules, in fact, represent the software part of GST (i.e. the operating system), which consists mainly of a kernel and a set of processes (supervisor and user), communicating and synchronised by message passing. The kernel and supervisor processes are asynchronous monitors ((/Wettstein and Merbeth 80/)) in charge of particular resources. The hardware part was represented in GST by selecting typical computer system hardware configurations. In the meantime, GST provides a very structured scheduling system in which the processes can compete for resources in a highly organized environment. Scheduling is a very vital subject, and it represents one of the important parts of the software components which is usually missing in conventional approaches. Hence, this subject deserves more

attention and we will try to highlight on the scheduling facilities available in GST.

3.2.2. Model Components:

The complete description of a system model consists of the following types of information ((/Hellerman and Conroy 75/)):

1. Workload Description.
2. System Structure.
3. Scheduling.
4. Performance Indices.

The workload description states how jobs are to be characterized, for example, by arrival and required execution times. By system structure, it is meant the individual resources and the paths by which jobs may be moved into, out of, and within the system. The scheduling rules specify how jobs are selected for movement within the system. Common examples include FCFS (First come, first served) and SXFS (Shortest execution first served). Performance-Indices define one or more ways in which the 'goodness' of the system is to be measured or judged. Hence, we will try to follow this classification in describing the GST model.

3.2.3. The Selected System of the GST:

The general system to be studied and modelled supports a number of terminals. Each of these place a non-trivial computational load and many I/O operations. This is a virtual storage system which could be paged or segmented. Demand paging is used to move required portions of a user's address space into main storage. A page fault occurs if the referenced page is not in main storage.

The paging is done from drum and disc. The system also maintains some waiting queues that are available to each system device. The degree of multiprogramming is limited or affected by the working sets of processes. This is done to avoid thrashing. The jobs in the system may have different priorities, which can be classified in the system input parameters. Also, the jobs may have different sizes. The ratio of the response time to think time is assumed to be small since we assume the system is fast and powerful.

The overall hardware configuration consists of typical disc, drum, operator console, terminal interface, channels, main store and CPU (with up-to-date facilities). As mentioned previously, our overall software consists of a kernel and a set of processes (i.e.

supervisor and user). The system configuration is shown in figure 3.2. and some device characteristics are given by table 3.1.

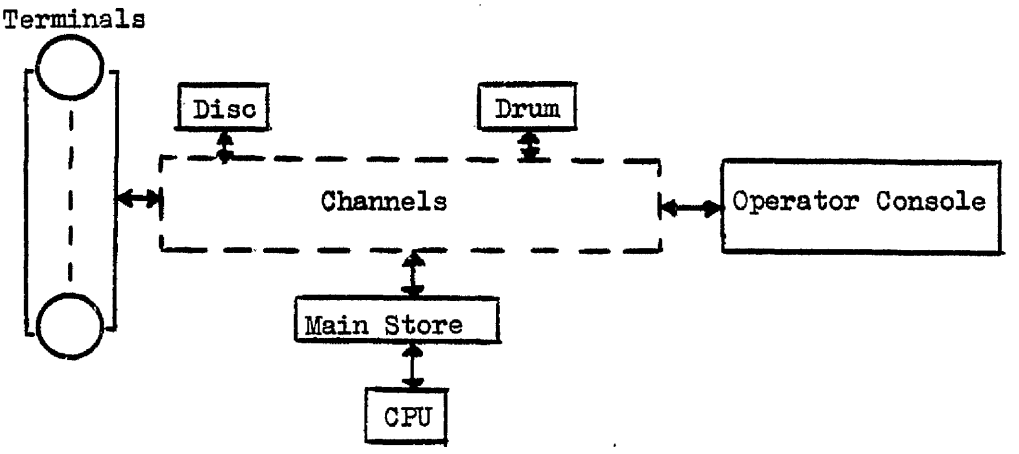


Figure 3.2. The Selected Interactive GST Hardware Configuration

Device Name	Transfer Time (m secs./byte)	Seek Time (m secs.)	Latency Time (m secs.)	Record Size (Bytes)
Drum	8.333e-4	0.000	8.000	32767
Disc	3.333e-3	7.500e1	1.200e1	32767

Table 3.1. Characteristics of Devices.

3.2.4. The GST Components:

3.2.4.1. As mentioned in section 3.2.2. , the GST consists of the following components:

The Workload:

The GST treats a computer system in a heavy workload situation. This means that there are always jobs waiting outside the system. Whenever a program finishes, a new job arrives. It has been assumed that a typical program alternates between service at the CPU and at one of the I/O devices. After a service at the CPU, the program is either finished or it requires service at I/O device. The input jobs stream is considered as poisson distribution. The mean-interarrival time is a variable of the tool.

The workload is specified by many parameters, such as job size, number of interactions, scheduling parameters...etc. Hence, it

The supervisor processes in GST were allocated relative priorities as given in table 3.2.:

Layer no.	Priority no.	Name
0	-1	Kernel
1	0	Store Manager
2	1	CPU "
	2	Operator Counsol Manager
	3	Terminal System Manager
	4	Drum Manager
	5	Disc Manager
	6	L/P Manager
	7	C/R Manager
3	8	Process Creator
	9	File System Monitor
4	10	Output Spooler
	11	Input Spooler
5	12	Job Scheduler
6	12	User Processes (includes editors, compilers and other utility programs)

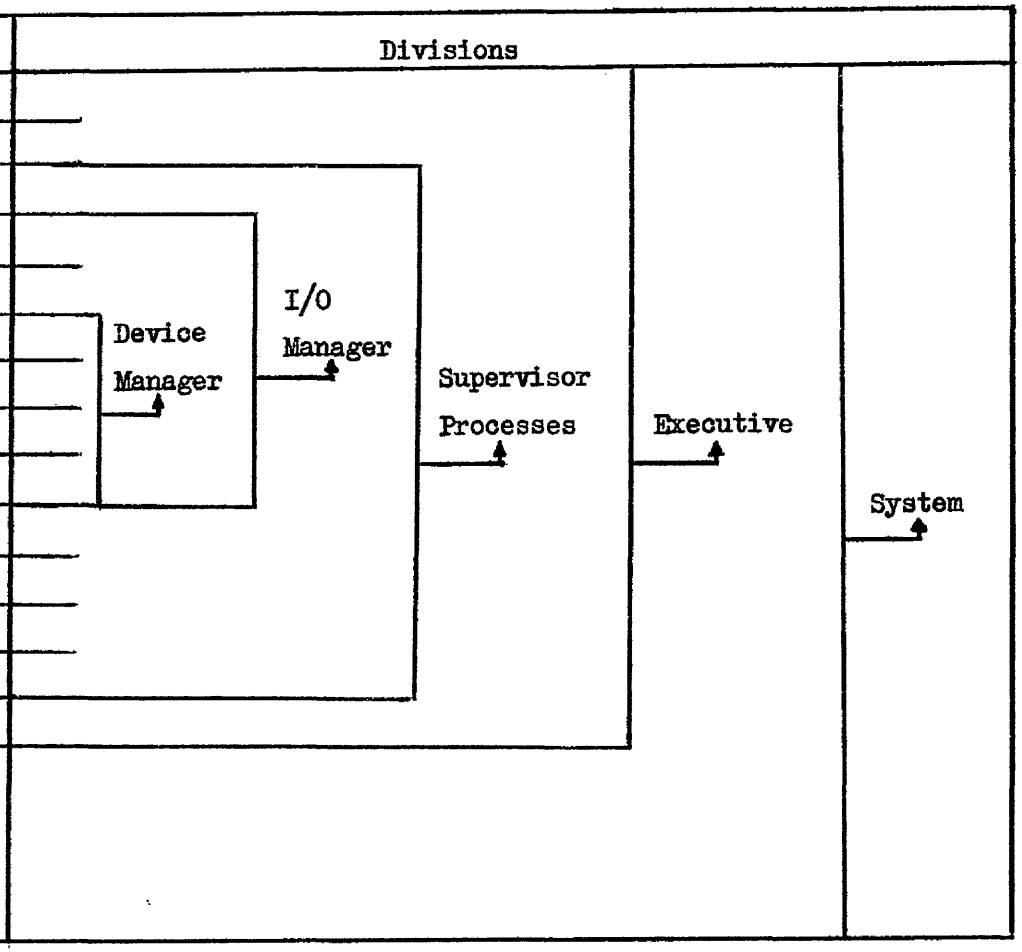


Table 3.2. GST Relative Priorities of Processes.

is difficult to specify the performance indices that affect the workload.

Number of jobs is considered to be a very large integer, otherwise the simulation program will consider it as a termination factor instead of the given simulation period.

3.2.4.2. System Structure:

GST has the same software and hardware structure as the system being modelled. A typical system software is the operating system. Hence, GST operating system consists of a kernel and a set of processes (supervisor and user) communicating and synchronised by message passing.

The supervisor is constructed in a hierarchical way. It consists of several layers; each layer contains one or more of such supervisor processes and each level implements a more convenient virtual machine for higher level processes. A process at a particular level operates in terms of virtual resources at lower levels and is not aware of what other processes and virtual resources exist at the same or higher layers. In other words, a process at one level is restricted to call upon processes at lower levels only. Figure 3.3. illustrates the layout of the various layers.

The flow of processes inside the system is determined by the operating system structure and the relative priorities of these processes. Several queues are available in the tool (mainly for each supervisor process) which acts as another source for organising the flow of the processes inside the system. From figure 3.3. the GST consists of the following layers and processes:

a. The Kernel:

The major interface between the basic machine hardware and the operating system is provided by the kernel which is the innermost layer of the nucleus of the executive. The kernel is not just a monitor — it is the only resident program which runs in Priveleged mode with interrupts inhibited ((/Lister 75/)). The kernel provides the following functions to the rest of the supervisor — short-term scheduling, virtual processors, protection (capabilities), interrupt handling, I/O scheduling and control, synchronisation and communication primitives and dispatching.

b. The Executive:

The executive consists of a set of monitors which have

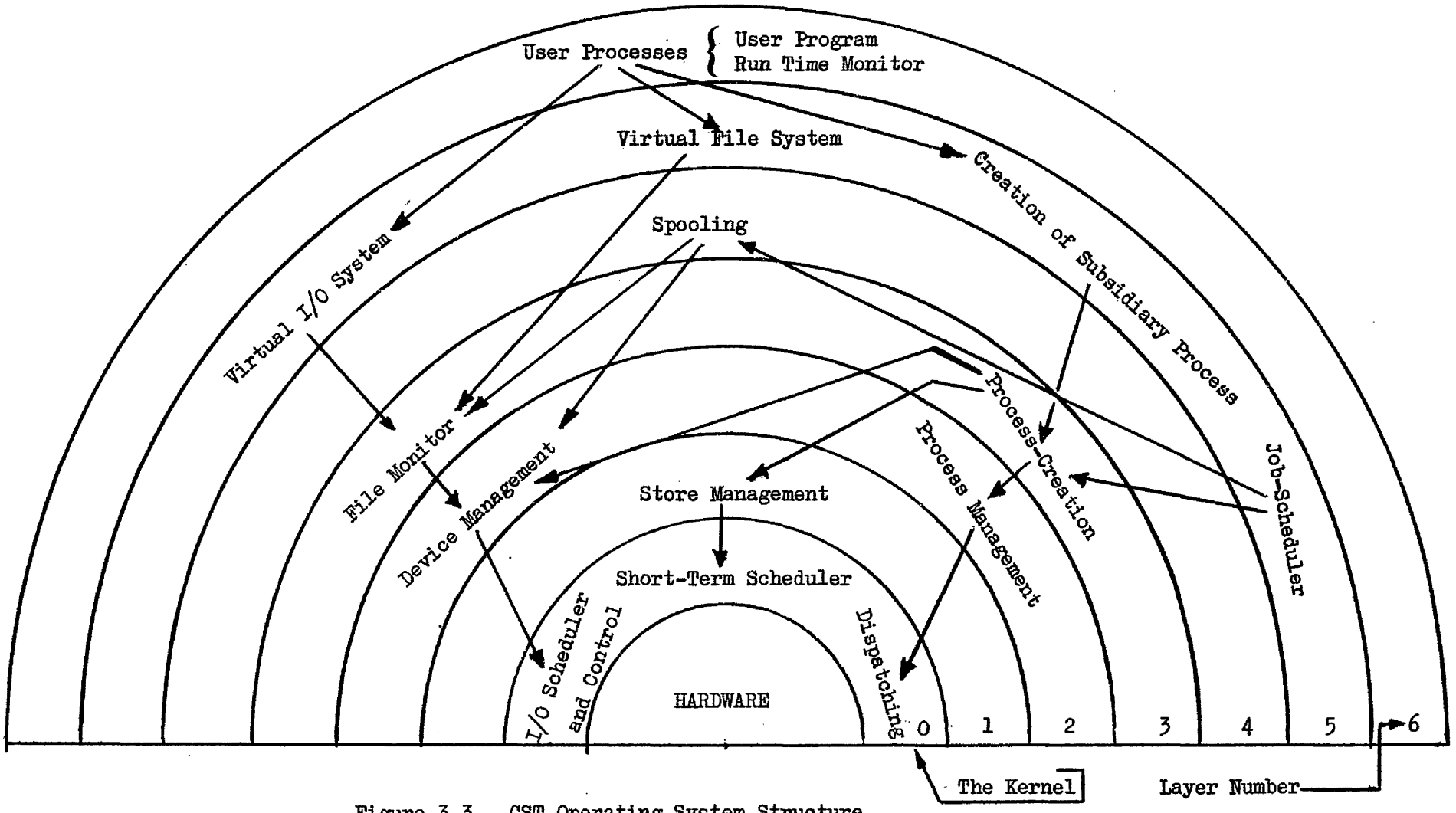


Figure 3.3. GST Operating System Structure.

been structured in a uniform way. The monitors provide the service offered by the executive. These monitors are:

1. Store manager:

   This monitor constructs a one-level virtual store from the two-level physical store. The store manager is invoked by the process creator and the CPU manager, when a process is created and deleted respectively. It is also invoked when a page or segment fault occurs. The store manager invokes the disc and the drum managers.

2. Process management: (process scheduler or medium-term scheduler or CPU manager)

   It is responsible for allocating the processor between the active processes. This process is invoked by the process creator to allocate or delete process descriptors when a process is created or deleted. It is also invoked to perform medium-term scheduling (i.e. prepare the dispatcher list). Also, it is invoked by the job (long-term) scheduler to report on resource utilization and system response.

3. The device managers:

   These monitors perform the actual input and output, thus hiding the actual physical I/O devices from processes. There is one supervisor process corresponding to each of the physical I/O devices and each such process synchronises its activities with the interrupts from its associated device, which are converted to messages sent to it by the kernel. There is also a terminal system manager which provides the interface for processes to communicate with the outside world through terminals. All I/O requests have to go through the appropriate device manager.

4. File monitor:

   The file system monitor is concerned with the physical features of the file system, namely,

   * Auxiliary storage management.
   * Physical organization and access methods.
   * Access control verification.
   * Basic file system, and
   * Symbolic file system.

   The file system is modelled in a very simple fashion. Its function is to accept requests from other processes

and to send messages to the Disc Manager to satisfy these requests.

5. Process creator:

This simple monitor provides the facilities to create and destroy processes. It is invoked by the job scheduler and by any other parent process. This process invokes the CPU manager, store manager and disc manager.

6. Job scheduler:

This monitor provides the long-term scheduling function, job initialization, accounting and preallocation of some system resources such as files. The job scheduler invokes the operator's console, the process creator and CPU manager. It is in turn invoked when a multiaccess user arrives at the system.

7. The CPU manager:

This monitor (also known as ALP manager or medium-term scheduler) calculates the priorities of the eligible processes.

c. User Processes:

This layer includes the user program and their run-time monitors. Each user process works in synchronisation with the lower level processes that it invokes.

3.2.4.3. Scheduling:

Scheduling strategies are evolved to solve problems of selection in operating systems. There are many examples of scheduling in all operating systems, some of them being ((/Landly 71/)):

* Allocation of time to processes.
* Allocation of peripherals to processes.
* Allocation of core store to processes.
* Choosing the next task for an output device.
* Choosing the next job to be run.
* Allocation disc space to users.

The GSF scheduling system consists of the following four levels:

1. Low-level scheduler (the dispatcher).
2. Short-term scheduler.
3. Medium-term scheduler.
4. Long-term scheduler.

These represent a multi-level scheduler system. The first two schedulers are embedded within the kernel. This kernel, in fact, consists of three modules ((/Lister 75/)):

* The first-level interrupt handler, which performs the

initial handling of all interrupts.

* Procedures which implement the inter-process communication functions and other primitives. These procedures are invoked via primitive calls in the processes concerned.
* The dispatcher, which switches the central processor(s) between processes.

The kernel is automatically entered in any of the following circumstances:

i. An interrupt occurs.

ii. A process issues a permitive to execute some function or requesting the use of a priveliged instruction.

The permitives available in GST kernel are:

* SEND-MESSAGE.
* RECEIVE-EVENT.
* RECEIVE-MESSAGE.
* START-PROCESS.
* CREATE-PATH.
* CHANGE-ELIGIBLE-SET-NUMBER.
* INITIATE-IO.
* ACTIVE-PROCESS.
* ABORT.
* HALT.

Also, the following subroutines (procedures) are used by the kernel to collect some important measurements,(e.g. interaction time, time blocked, time ready...etc.).

* IN-STATISTICS (QUEUE-NUMBER).
* OUT-STATISTICS (QUEUE-NUMBER, TIME-IN).

Now, to give a brief idea of the scheduling environment we present Figure 3.4. which represents a model of the system operations in terms of flow of jobs and processes.

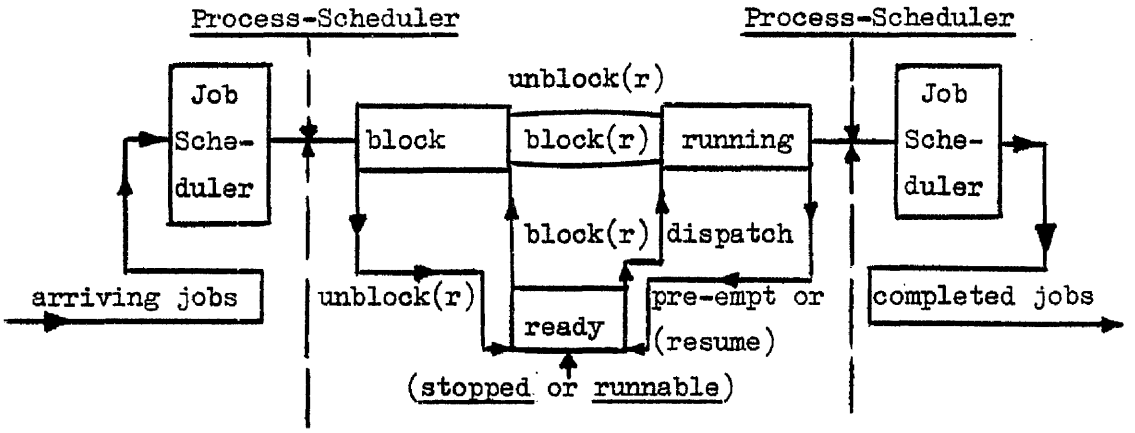


Figure 3.4. Schematic Diagram of the Scheduling Environment in GST.

Finally, the important data structures available in GST are:

1. The process descriptor and the process descriptor table,
2. Message buffers and queues,
3. Job descriptors and the job descriptor table,

as well as several data structures used for memory management. In the meantime, we will continue to discuss the GST multi-level scheduler in considerable detail:

3.2.4.3.1. The Low-Level Scheduler (The Dispatcher):

The dispatcher is involved after the handling of an interrupt has been completed, and to allocate the central processor among the various processes in the system. Its function is limited to choosing the next process to be executed from the processor queue (the queue of eligible processes). The dispatch process can be summarized by the following algorithm in a Pascal-like notation:

```
P: = dispatcher-list-head;
if   P  <> NIL        (* if there is a ready process*) then
begin
    if  P <>  current-process (* the process which was executing
                                 just before the kernel was invoked*)
                                      then
    begin (* CONTEXT SWITCHING, privileged instructions*)
        RESTORE the context of process P;
        current-process : = P
    end;
    ENTER (* the current-process, another privileged instruction*)
end;
(* you might want to save the current-time at this point, for latter
calculation of the CPU-idle-time*)
(* idle loop*)
While true do (* or you can use PAUSE (or WAIT) , a privileged
                 instruction*)
      (* Alternatively, a null or waiter process can always be the
         last one (the one pointed to by dispatcher-list-tail) on
         the dispatcher list, in which case P will (or should be)
         always nonNIL, and it will be entered. In this case, the
         null process's run time will be equal to the CPU-idle-time
         (non-productive rather than idle)*).
```

3.2.4.3.2. The Short-Term Scheduler:

The functions of this scheduler can be summarized as follows:

1. To allocate resources to processes as soon as they become available. Scheduling decisions taken at this level

determine the rate at which the system is able to respond to real time events.

2. The short-term scheduler simulates a virtual machine for each process and implements the set of primitives which enable concurrent processes to achieve mutual excusion, synchronization and communication with one another.

Since this scheduler is invoked whenever an interrupt (internal or external) occurs, its function should be confined to the examination/modification of the state of processes and the collection of measurements for use by medium and long term schedulers.

3.2.4.3.3. The Medium-Term Scheduler: (CPU Manager)

This scheduler performs the following functions:

* Prepares the dispatcher list.
* Processes the events collected by the short-term scheduler.
* Calculates the resource allotment of the processes.
* Reports to long-term scheduler.

The medium-term scheduler is entered whenever a scheduling event occurs. The primary scheduling events are:

* Process is created and/or deleted.
* At fixed time intervals.

The above functions are designed mainly to provide high resource utilization, low response time, high throughput and low overhead times. The CPU-Manager process can be summarized in the following algorithm:

```
declarations;
begin
    INITIAL-ENTRY; (* The initialization (or setting up) of several
                      important variables, such as free list pointer,
                      dispatcher list pointer...etc., is performed*)
    MAIN-ENTRY; (* In this procedure the following are performed:
                   A RECEIVE-EVENT permitive is issued specifying
                   the permanant ports to the job scheduler, store
                   manager and the process creator process
                   If activation is received then
                   MANAGER (* This procedure performs the medium-
                              term scheduling function*)
                   else
                   case message-command of
                   CREATE: begin
                           (* This entry is invoked whenever the
                              process creator process requests a
                              descriptor to be reserved for a process
                              to be created*)
```

```
                end;
        LOADED: begin
                (*This entry is invoked whenever a
                  process has been created*)
                end;
        LOAD-FAILED: begin
                (*This entry is invoked following
                  an unsuccessful attempt to complete
                  the creation of the process
                  specified in the message*)
                end;
        REPORT: begin
                (*This entry is invoked whenever the job
                  scheduler inquires on the system load
                  in order to decide whether or not to
                  admit a new multiaccess job into the
                  active job mix*)
                end;
        DELETE: begin
                (*This entry is invoked when the process
                  creator requests the deletion of the
                  process specified in the message
                  identifier*)
                end
        end (*case*) *)
end (* CPU-Manager *)
```

3.2.4.3.3.1. The Design of CPU Manager Procedure:

The policy of the CPU Manager procedure chosen in the GST is a deadline scheduling one. In this policy the process priority is basically dependent upon an estimate of how long a process will take and how long it has run. In particular, this deadline scheduling policy is a policy driven one in which processes are ordered according to increasing times of their (current) interactions.

The policy functions used in GST are:

$$t_c(R) = \begin{cases} R.m_H & \forall \quad 0 \leqslant R < t_H \\ R.m_{INF} + t_H(m_H - m_{INF}) & \forall \quad t_H \leqslant R < t_{INF} \\ t_{INF}.m_{INF} + t_{INF}(m_H - m_{INF}) & \forall \quad R \geqslant t_{INF} \end{cases}$$

where R, t_H, t_{INF} are all expressed in time units, m_H, m_{INF} are dimensionless and R represents the amount of service received by the process. The shape of these policy functions can be shown in figure 3.5.

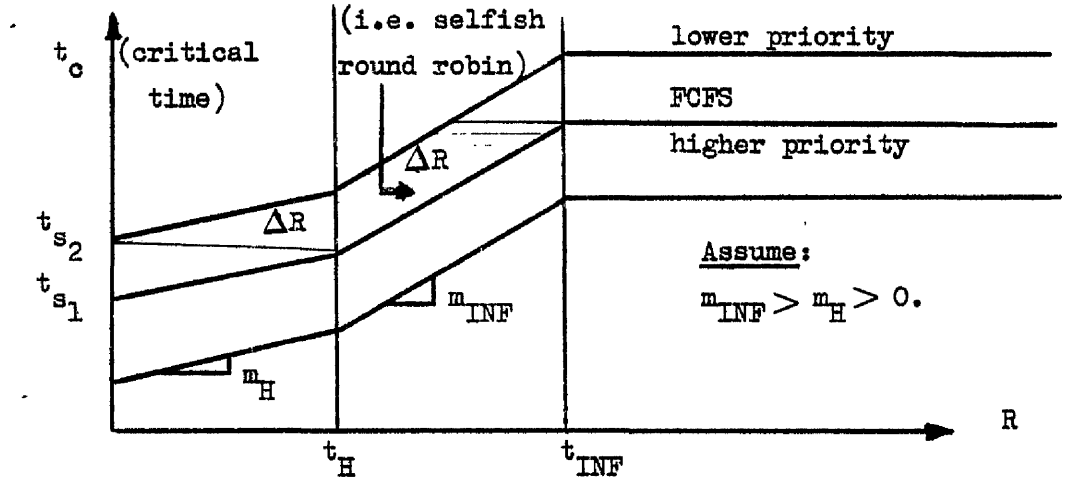


Figure 3.5. Shape of Policy Function and Critical Times.

For more information about these policy functions refer to ((/Cavouras 78/)). The scheduling is implemented as a dynamic balance scheduling system (i.e. the load is adjusted according to the existing equipment or configurations). Again, for further information the reader is referred to ((/Denning 69/)) and ((/Gotlieb and Schonbach 80/)).

3.2.4.3.4. The Long-Term Scheduler (The Job Scheduler):
The functions of this scheduler can be summarized as follows:

* To allocate virtual machines to users (jobs) according to rules laid down by the instalation management.
* Establishment of the identity and authority of users, the input and analysis of their requests.
* The initiation and control of users computations.
* Accounting users' resource usage.

The long-term scheduler is invoked when a job enters or leaves the system. The GST policy of the multi-access long-term scheduling can be summarized as follows:

> The job scheduler used here is simple. There is a specific maximum number of terminals on the system at any time. This number, n say, is fixed or the operator of the computer system can set a limit on the total number of users allowed to dial into the system.

3.2.4.4. Performance Indices:

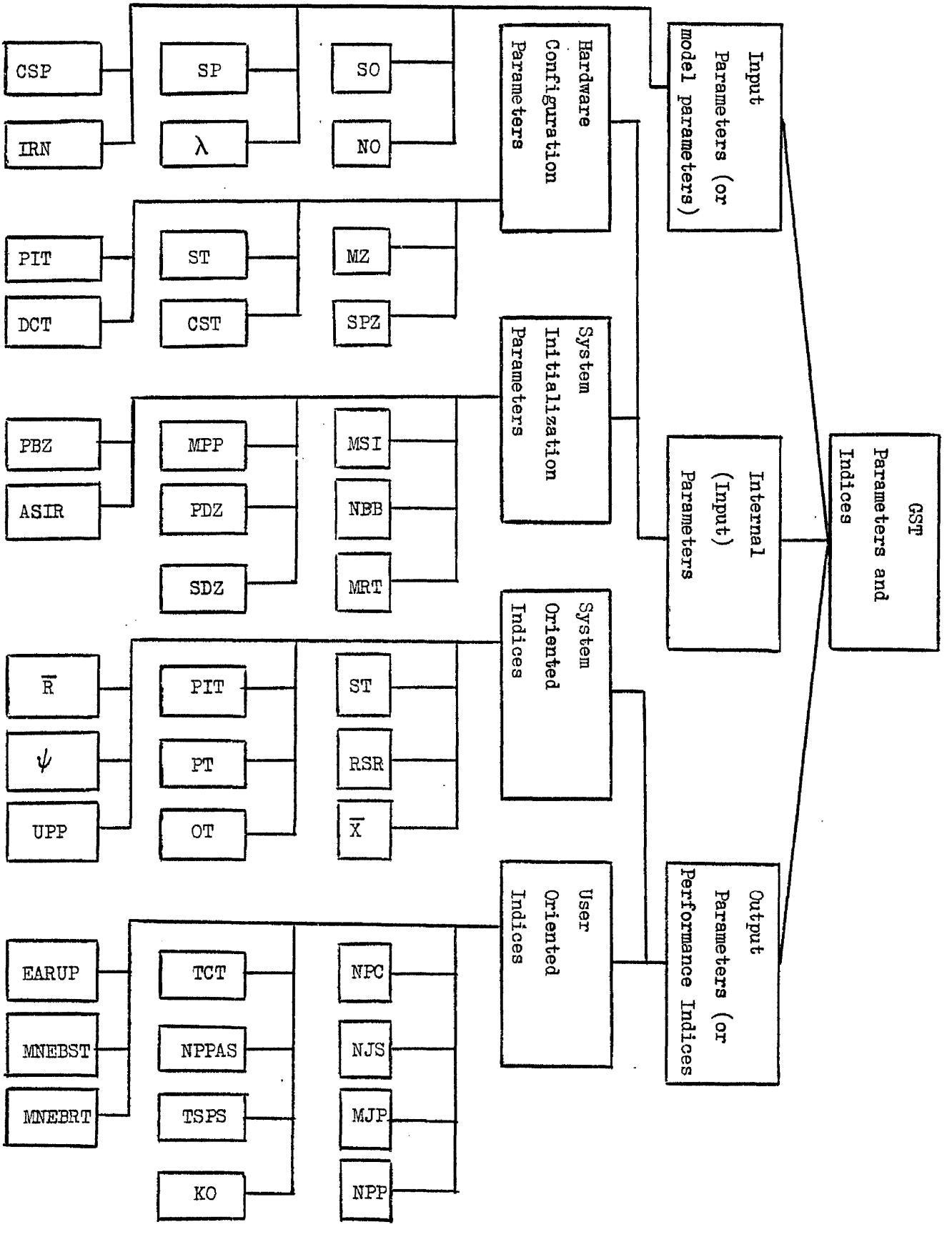
A performance index is a descriptor which is used to represent

a system's performance or some of its aspects ((/Ferrari 78/)). In GST we can identify three kinds of performance indices (or parameters). The input parameters corresponding (in general) to the workload, internal parameters corresponding to system hardware variables and output parameters corresponding to the user-oriented and system-oriented performance indices of interest to evaluators.

The GST performance indices can be classified as in figure 3.6. In the following the reader is assumed familiar with ((/Bytheway 80/)).

Figure 3.6* The GST Performance Parameters and Indices.

* The abbreviation used in the relation matrix can be found in abbreviation appendix A.2.



3.3. Experimentations:

The aim of this section is to use the GST as an aid to generate an analytical performance prediction tool. This will be attempted using regression analysis techniques in which the results of several tests on the GST are modelled, using the regression analysis techniques to produce several hybrid models. The hybrid models (i.e. equations) represent the result of combining the simulation technique (using the GST) and the regression analysis techniques. Hence, we may call the analytical performance prediction tool as Simulation/Regression tool. This tool can offer a number of advantages — in particular the following benefits:

* By combining simulation and regression analysis techniques, the advantages of both modelling techniques may be exploited.
* It enables the analyst to obtain approximate solutions.
* It requires substantially less main memory space and central processor time than the GST. Hence, it is less expensive.
* It can predict the effects of the changes over the limited simulation time given by the GST.

The number of tests (and hence the number of hybrid models) necessary will depend on the number of variables (i.e. performance parameters and indices) and on whether the analyst decides to use a classical approach or a fully factorial consideration of all the variables. For the purpose of our research experiments, the classical approach has been used. All the variables are properly controlled, no two variables being allowed to change in any one test. For the purpose of the second approach, the reader is referred to ((/Baird 62/)) ((/Cox 58/)) ((/Kempthorne 52/)).

The reader should also note that the variables used in the simulation/regression tool is a subset of those used in GST. The subset chosen represents the variables of an interactive computer system.

The tests will be performed using several case studies (see the following sections). All cases have been analysed under heavily loaded system conditions. A heavily loaded condition is likely to happen after a long run of the GST. To achieve this, we have selected the simulation time to be long enough (specifically, simulation time = 35 min.).

The case studies presented in this section are:

* Case 1: Analysis of response time vs. no. of users,

CPU busy time vs. no. of users and
interactive throughput vs. no. of users.

* Case 2: Analysis of response time vs. no. of tasks/user interaction,
CPU busy time vs. no. of tasks/user interaction,
and interactive throughput vs. no. of tasks/user interaction.

* Case 3: Analysis of response time vs. average think time,
CPU busy time vs. average think time and
interactive throughput vs, average think time.

* Case 4:Analysis of response time vs, mean inter-arrival time,
CPU busy time vs. mean inter-arrival time and
interacive throughput vs. mean inter-arrival time.

* Case 5: Analysis of degree of multiprogramming vs. no. of users,
drum utilization vs. no. of users,
disc utilization vs. on. of users,
terminal connect time vs. no. of users,
no. of multiaccess jobs processed vs. no. of users and
ratio of simulation time to real time vs. no. of users.

* Case 6: Analysis of degree of multiprogramming vs. no. of tasks/ user interaction,
drum utilization vs. no. of tasks/ user interaction,
disc utilization vs. no. of tasks/ user interaction,
terminal connect time vs. no. of tasks/ user interaction,
no. of multiaccess jobs processed vs. no.of tasks/ user interaction,
and ratio of simulation time to real time vs. no. of tasks/ user interaction.

* Case 7: Analysis of degree of multiprogramming vs. average think time,
disc utilization vs. average think time
drum utilization vs. average think time
terminal connect time vs. average think time
no. of multiaccess jobs processed vs. average think time
and ratio of simulation time to real time vs. average think time.

* Case 8: Analysis of degree of multiprogramming vs. mean interarrival time,
disc utilization vs. mean interarrival time,

drum utilization vs. mean interarrival time,

terminal connect time vs. mean interarrival time,

no. of multiaccess jobs processed vs. mean interarrival time,

and ratio of simulation to real time vs. meam interarrival time.

The simulation/regression tool should be constructed using the method followed in the above mentioned case studies. For further information the reader is referred to section 3.4.

The reader is also referred to Appendix A.1. for further information on the regression analysis and some other helpful statistical methods ((/Sprangins 79/)) ((/Rehmann and Gangwere 68/)) ((/Gomaa 76/)).

3.3.1. Case 1: Effects of performance parameter: number of active users.

The run of the GST was made in this case varying the workload from 16 to 48 active users in steps of 8. Refer to table C1.1 and graphs C1.1, C1.2, C1.3, C1.4, C1.5 and C1.6.

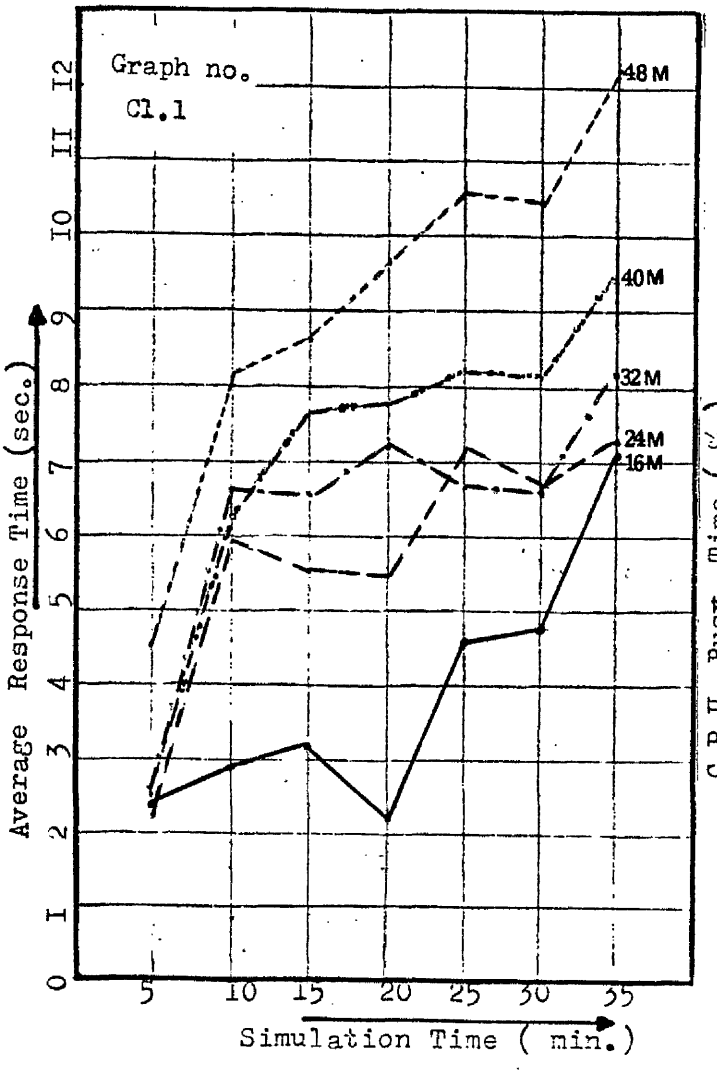
The relationship between the parameter, number of active users (M) and the selected performance indices (i.e. average response time ($\overline{R}$), CPU busy time (PBT) and interactive throughput ($\overline{X}$)) can be constructed from the direct graphs (C1.1, C1.2, and C1.3) using regression analysis (curve fitting) and under heavy loaded system condition (i.e. simulation time = 35 mins.) These relationships represent simple hybrid models which can be expressed using the relational graphs (C1.4, C1.5 and C1.6) and table C1.2.

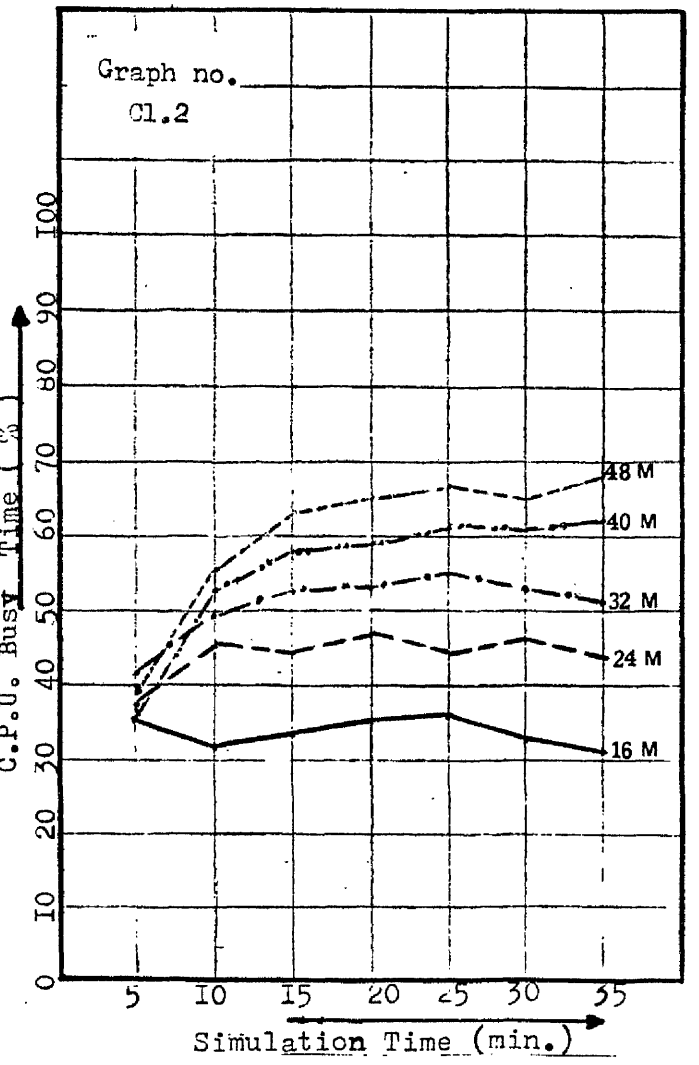
Performance Index	Relationship Equation	Regression Constant Values	Graph No.	Equation Number
Average Response Time (sec.)	$\overline{R} = ae^{bM}$	a = 5.00 b = 0.02	C1.4	E1,1
CPU Busy Time (%)	$\overline{PBT}=a+b\ln M$	a = -5.7 b = 0.32	C1.5	E1,2
Interactive Throughput (Processes/ min:)	$\overline{X}=a+b\ln M$	a = 0.91 b = 0.41	C1.6	E1,3

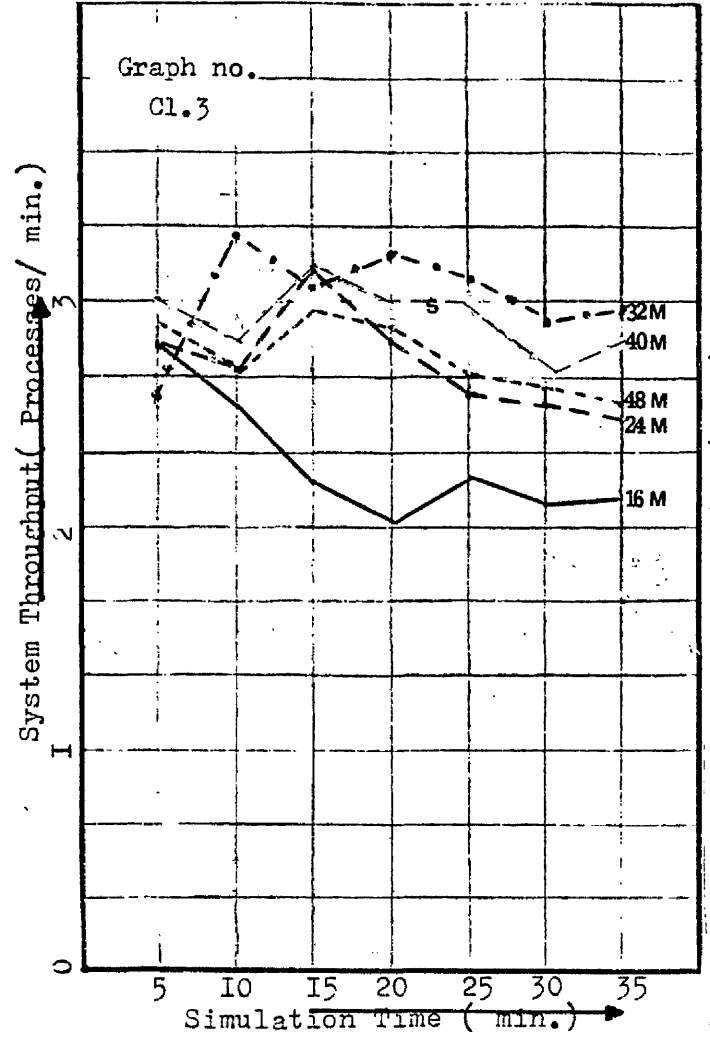
Table No. C1.2

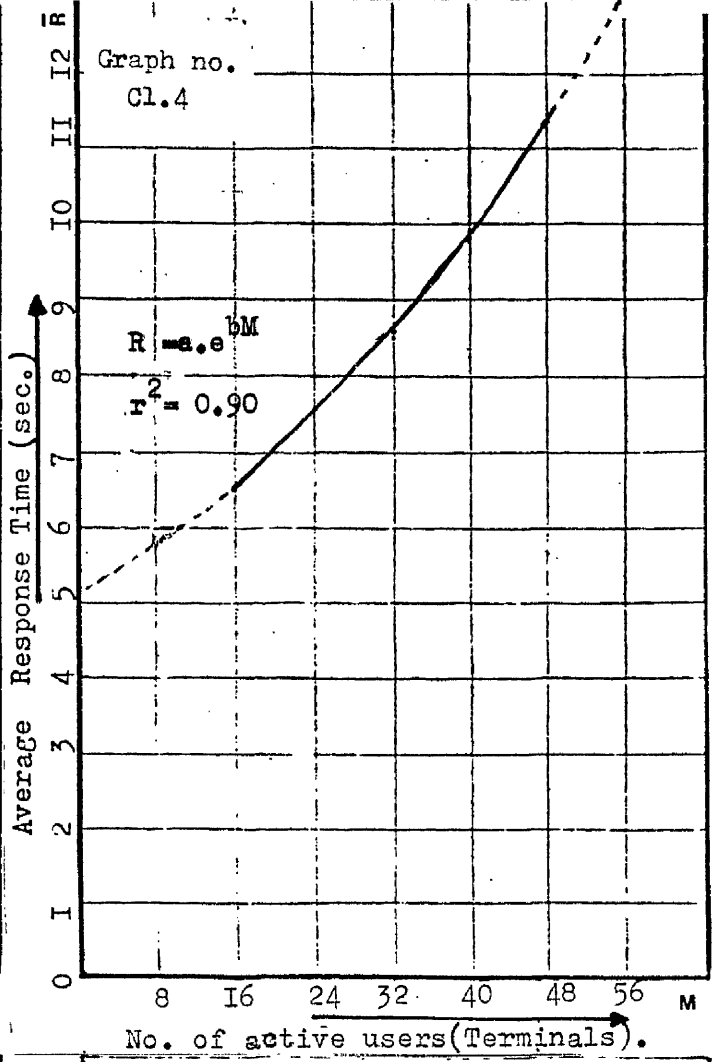
PERFORMANCE INDEX	Simulation Time (min.)	PERFORMANCE PARAMETER: No. of Active Users.									
		16		24		32		40		48	
Average Response Time (Sec.)	5	3.38		3.30		3.42		3.39		4.55	
	10	3.89		5.94		6.69		6.19		8.19	
	15	4.13		5.62		6.68		7.65		8.69	
	20	3.28		5.55		7.23		7.76		9.68	
	25	4.70		7.18		6.68		8.13		10.50	
	30	4.81		6.70		6.61		8.05		10.40	
	35	7.10		7.11		8.92		9.40		12.20	
C.P.U. Busy Time (O.T. + P.T.) (%)		O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.
	5	13.74	22.36	16.35	22.46	16.82	23.72	16.04	22.74	16.07	22.70
	10	11.27	21.06	19.18	27.70	19.98	29.89	22.48	29.67	24.75	30.39
	15	10.53	22.48	18.26	27.72	21.37	31.09	26.45	31.32	31.38	31.39
	20	9.97	24.73	17.11	29.43	21.63	30.99	27.66	31.08	35.28	29.82
	25	9.59	25.50	15.63	29.16	21.14	32.46	29.38	31.29	36.69	29.70
	30	8.78	23.35	14.98	30.99	21.00	31.78	30.58	30.38	37.58	28.70
	35	8.10	22.36	14.44	29.11	20.64	30.72	31.61	29.79	39.47	29.05
Interactive Throughput (Processes /min)	5	2.80		2.80		2.60		3.00		2.90	
	10	2.60		2.70		3.30		2.90		2.70	
	15	2.26		3.13		3.06		3.13		2.97	
	20	2.05		2.80		3.20		3.00		2.89	
	25	2.28		2.68		3.12		3.00		2.93	
	30	2.10		2.66		2.93		2.76		2.95	
	35	2.11		2.50		2.97		2.82		2.59	

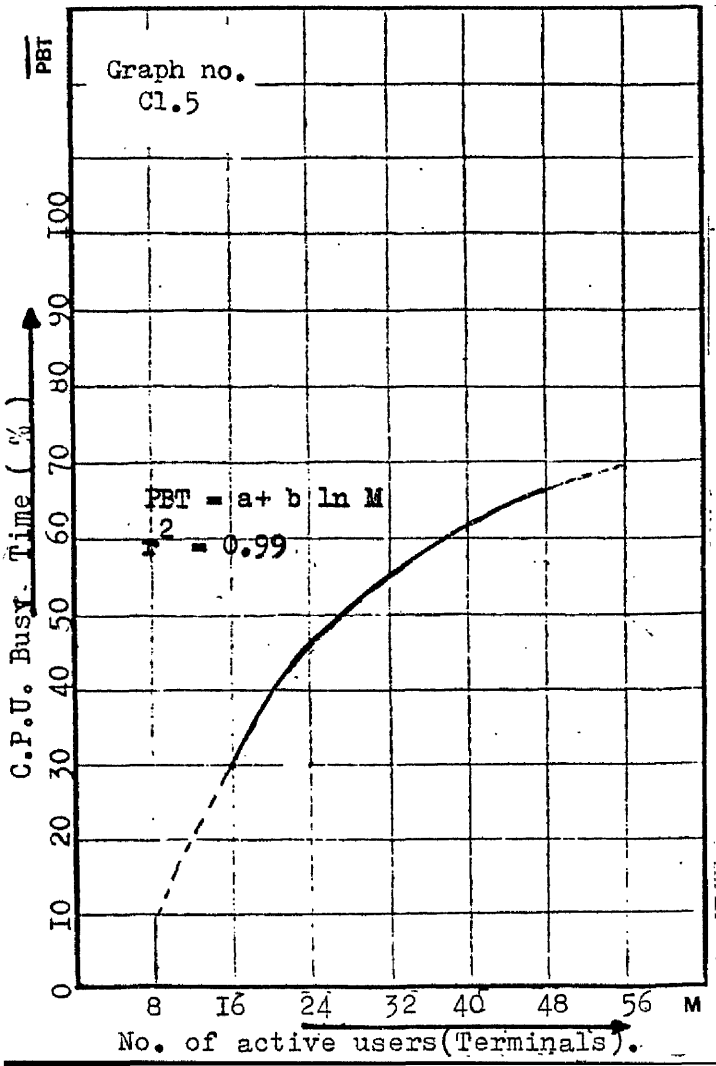
TABLE No. C1.1

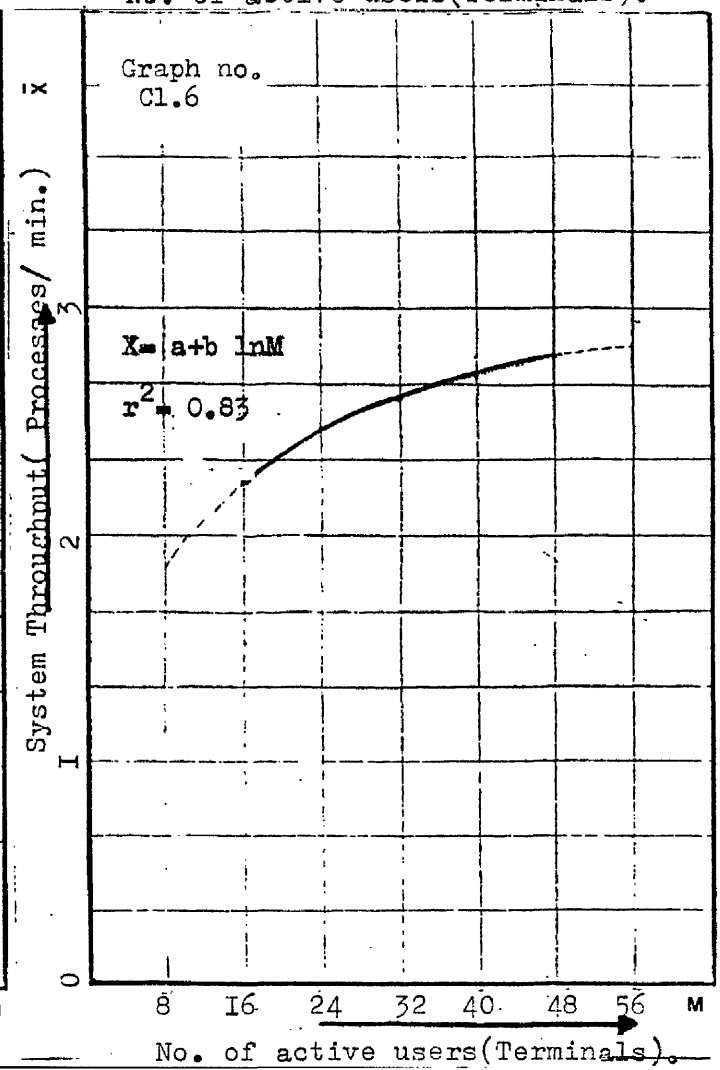





Graph no.
Cl.3
System Throughput(Processes/ min.)
0
I
2
3
32M
40M
48 M
24 M
16 M
5 I0 I5 20 25 30 35
Simulation Time (min.)


Graph no.
Cl.4
R
Average Response Time (sec.)
0 I 2 3 4 5 6 7 8 9 I0 II I2
R =a.e^bM
r^2 = 0.90
8 I6 24 32 40 48 56 M
No. of active users(Terminals).


Graph no.
Cl.5
PBT
C.P.U. Busy Time (%)
0 I0 20 30 40 50 60 70 80 90 I00
PBT = a+ b ln M
r^2 = 0.99
8 I6 24 32 40 48 56 M
No. of active users(Terminals).


Graph no.
Cl.6
x̄
System Throughput(Processes/ min.)
0 I 2 3
X= a+b lnM
r^2 = 0.83
8 I6 24 32 40 48 56 M
No. of active users(Terminals).

Conclusions:

* The effects of the parameter,number of active users on the selected performance indices are modelled. The models can be used to predict the future changes.
* For 'good' average response time keep the number of users below 44 active user.
* The CPU is not over-utilized even for 48 active user.
* The average interactive throughput shows a slight increase with the increase of active users in the system.

3.3.2. Case 2: Effects of performance parameter: No. of tasks per multi-access job.

The second case study addresses the effects of varying the average number of tasks per multiaccess job ($\overline{TSM}$) on the selected performance indices (i.e. average response time ($\overline{R}$), CPU busy time ($\overline{PBT}$) and interactive throughput ($\overline{X}$)). The run of the GST was made in this case varying the workload from 1 to 4 tasks per multiaccess job in step of 1. The number of users has been fixed to 32 users. Refer to table C2.1 and graphs C2.1, C2.2, C2.3, C2.4, C2.5 and C2.6.

The relationships between the parameter, no. of tasks per multiaccess job and the selected performance indices can be constructed from the direct graphs (C2.1, C2.2 and C2.3) using regression analysis and under heavy loaded system condition. These relationships represent simple hybrid models which can be expressed using the relational graphs (C2.4, C2.5 and C2.6) and table C2.2.

Performance Index	Relationship Equation	Regression Constant Values	Graph No.	Equation No.
Average Response Time (sec.)	$\overline{R}=ae^{b\,\overline{TSM}}$	a = 4.83 b = 0.20	C2.4	E2,1
CPU busy time (%)	$\overline{PBT} = k$	k = 97.83	C2.5	E2,2
Interactive Throughput (processes/ min.)	$\overline{X}=a+b\ln\overline{TSM}$	a = 2.04 b = 0.64	C2.6	E2,3

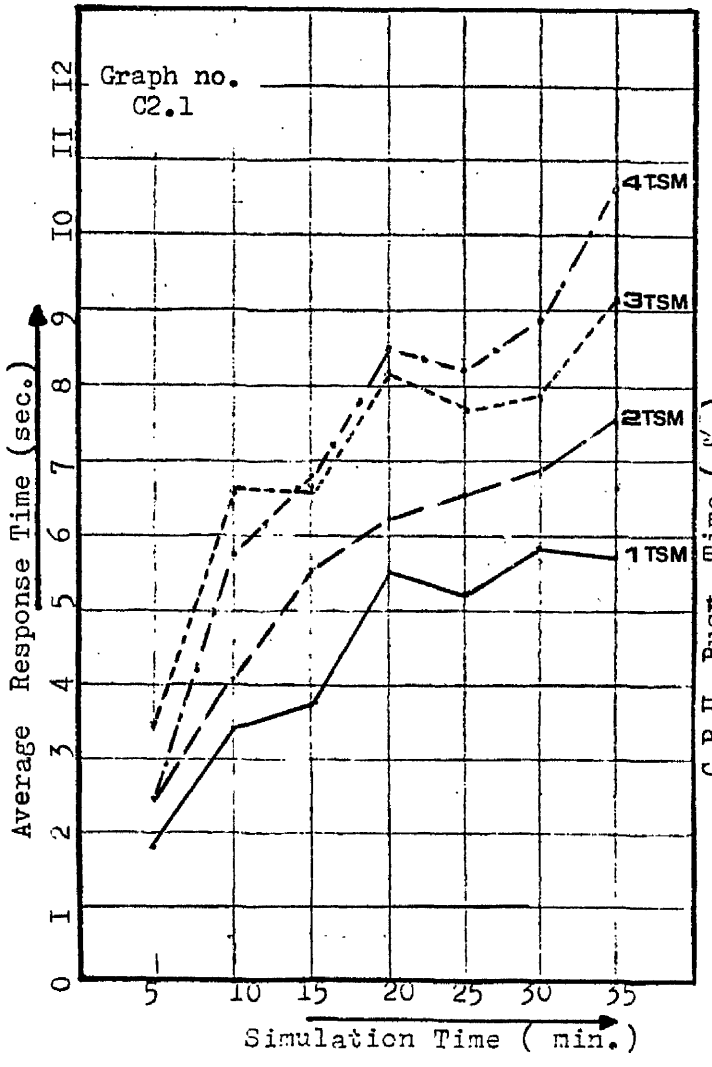
Table no. C2.2

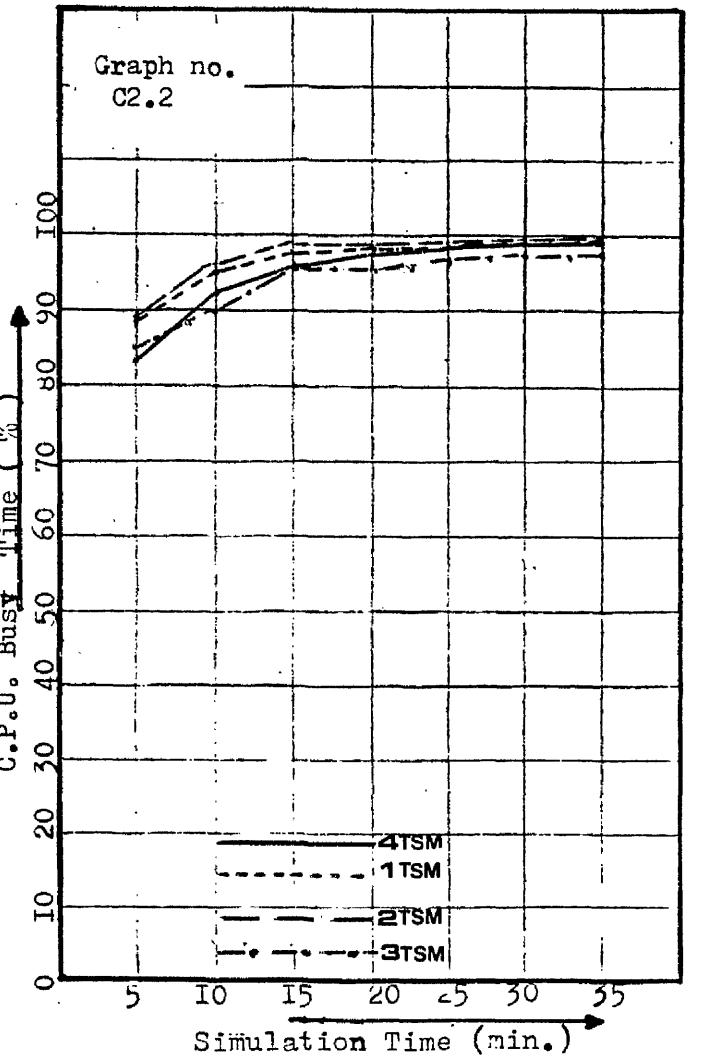
Conclusions:

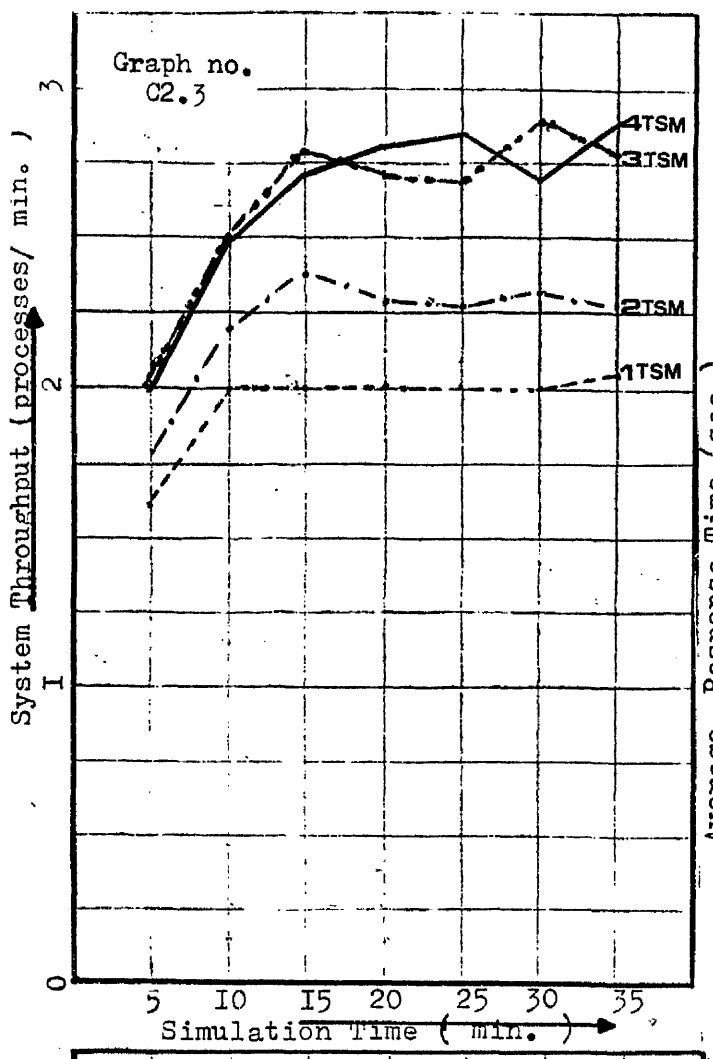
* The effects of the parameter, number of tasks per multiaccess job on the selected performance indices are modelled. The models can be used to predict the future changes.
* The average response time is increased sharply by the increasing no. of tasks per unit of time in a multiaccess job.
* The number of tasks depends on what the user wants to do. For our experiments, we will keep the number of tasks per multiaccess job equal to 3.
* The average number of tasks per multiaccess job has no effect on the CPU busy time.
* The average interactive throughput is affected slightly by the no. of tasks per multiaccess job.

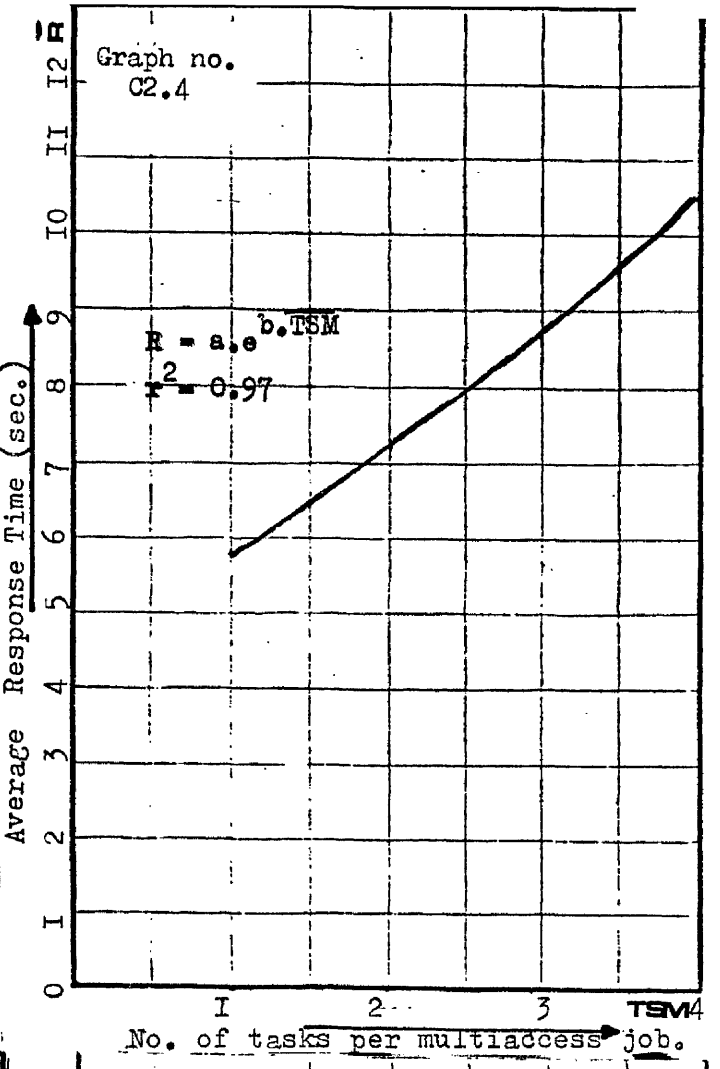
PERFORMANCE INDEX	Simulation Time (min.)	PERFORMANCE PARAMETER: No. of Tasks Per Multiaccess Job.									
		1		2		3		4			
Average Response Time (Sec.)	5	1.88		2.44		3.42		2.42			
	10	3.37		4.14		6.69		5.74			
	15	3.76		5.56		6.68		6.82			
	20	5.50		6.21		7.23		7.56			
	25	5.18		6.47		6.68		7.20			
	30	5.83		6.68		6.61		7.98			
	35	5.76		7.56		8.22		10.73			
C.P.U. Busy Time (O.T. + P.T.) (%)		O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.
	5	36.73	52.15	34.19	54.99	29.10	55.39	39.38	44.46		
	10	43.41	50.42	41.62	52.48	30.30	59.78	44.64	46.89		
	15	41.61	54.28	40.86	55.21	30.83	62.51	43.11	51.24		
	20	40.69	56.23	42.28	54.77	31.17	63.83	43.97	51.78		
	25	40.02	57.51	41.80	55.84	31.34	64.66	43.53	53.09		
	30	39.99	57.95	41.69	56.33	31.36	65.31	42.61	54.55		
	35	39.75	58.48	40.42	57.89	31.14	66.00	42.18	55.39		
Intermediate Throughput (Processes /min)	5	1.60		1.80		2.00		2.00			
	10	2.00		2.20		2.50		2.50			
	15	2.00		2.40		2.86		2.78			
	20	2.00		2.30		2.75		2.80			
	25	2.00		2.28		2.72		2.80			
	30	2.00		2.30		2.90		2.70			
	35	2.08		2.37		2.80		2.94			

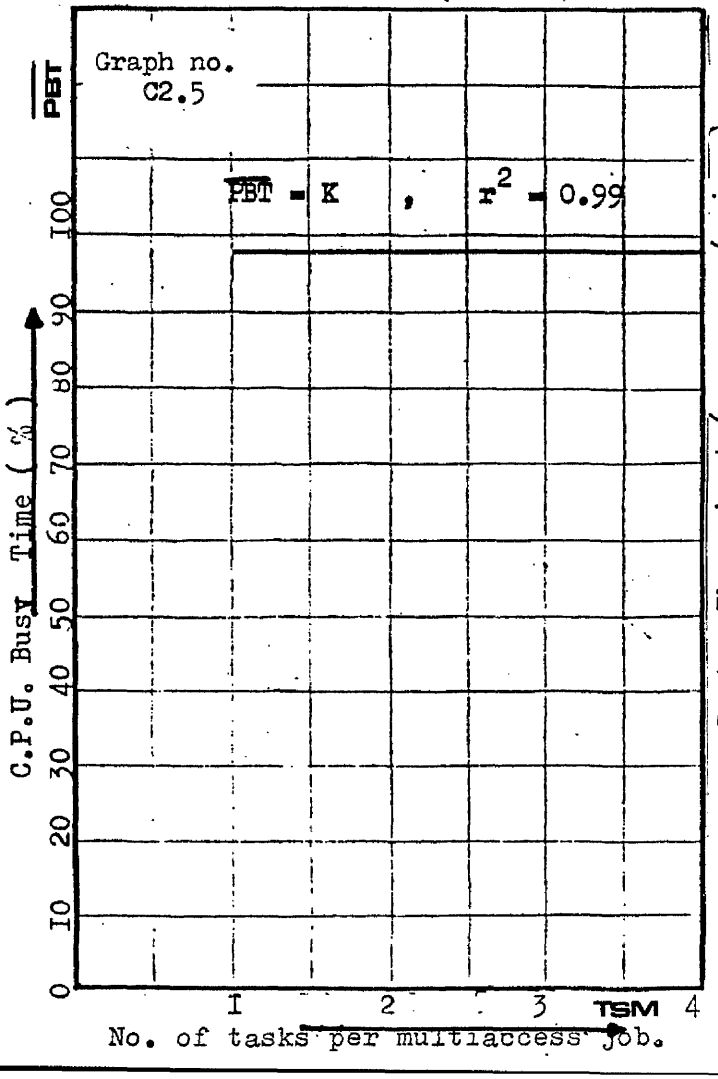
TABLE No. C2-1

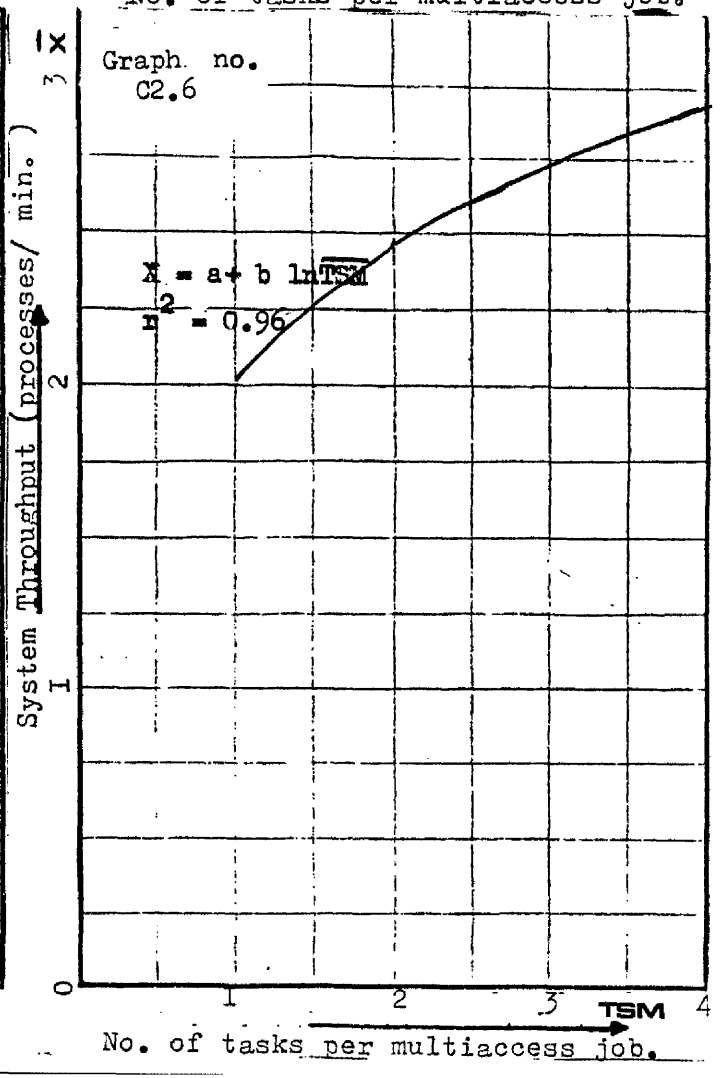





Graph no.
C2.3
4TSM
3TSM
2TSM
1TSM
System Throughput (processes/ min.)
Simulation Time (min.)
0
I
2
3
5
I0
I5
20
25
30
35


Graph no.
C2.4
R = a.e^b.TSM
r^2 = 0.97
Average Response Time (sec.)
No. of tasks per multiaccess job.
TSM
0
I
2
3
4
5
6
7
8
9
I0
II
I2


Graph no.
C2.5
PBT = K , r^2 = 0.99
C.P.U. Busy Time (%)
No. of tasks per multiaccess job.
TSM
0
I0
20
30
40
50
60
70
80
90
I00


Graph no.
C2.6
X = a+ b lnTSM
r^2 = 0.96
System Throughput (processes/ min.)
No. of tasks per multiaccess job.
TSM
0
I
2
3

3.3.3. Case 3: Effects of performance parameter: Average think time.

The third case study addressed the problem of effects of varying the average think time ($\overline{TH}$) on the selected performance indices (i.e. Average response time ($\overline{R}$), CPU busy time ($\overline{PBT}$) and interactive throughput ($\overline{X}$)). The run of the GST was made in this case varying the workload (i.e. using $\overline{TH}$) from 10 to 40 secs. in step of 10. Refer to table C3.1 and graphs C3.1, C3.2, C3.3, C3.4, C3.5 and C3.6.

The relationships between the parameter, average think time and the selected performance indices can be constructed from the direct graphs (C3.1, C3.2 and C3.3) using regression analysis and under heavy loaded system condition. These relationships represent simple hybrid models which can be expressed using the relational graphs (C3.4, C3.5 and C3.6) and table C3.2.

Performance Index	Relationship Equation	Regression Constant Values	Graph No.	Equation No.
Average Response Time (sec.)	$\overline{R}=a+b\ \overline{TH}$	a = 10.35 b = -0.09	C3.4	E3,1
CPU busy time (%)	$\overline{PBT}=a+b\ \overline{TH}$	a = 67.09 b = -0.49	C3.5	E3,2
Interactive Throughput (Processes/ min.)	$\overline{X}=a+b\ \overline{TH}$	a = 3.41 b = -0.02	C3.6	E3,3

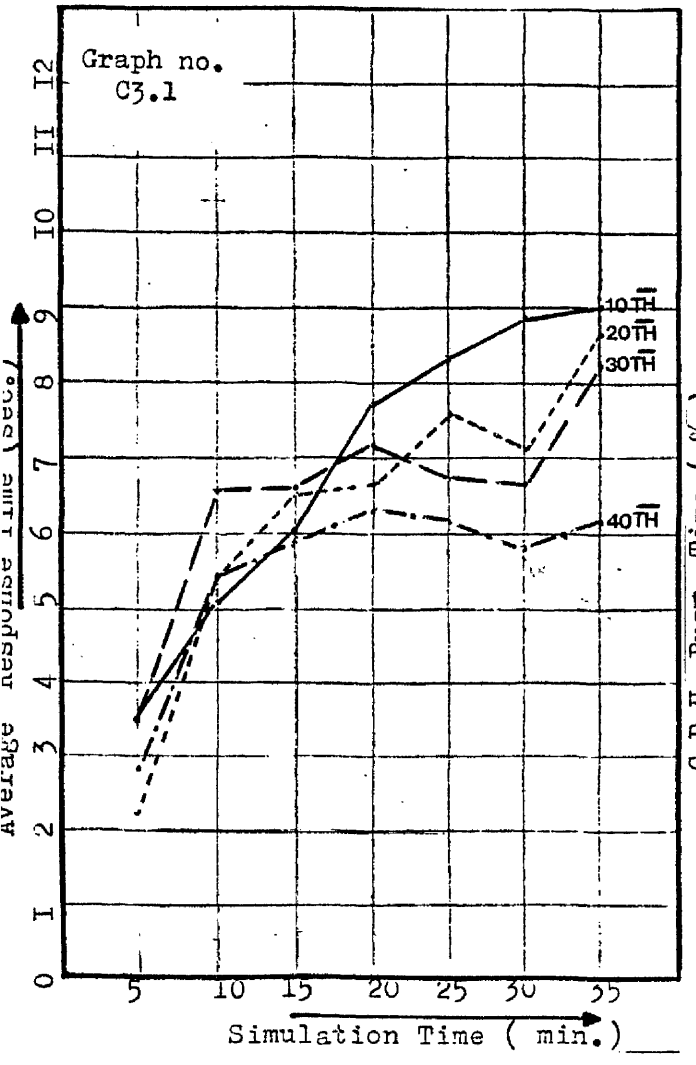
Table no. C3.2

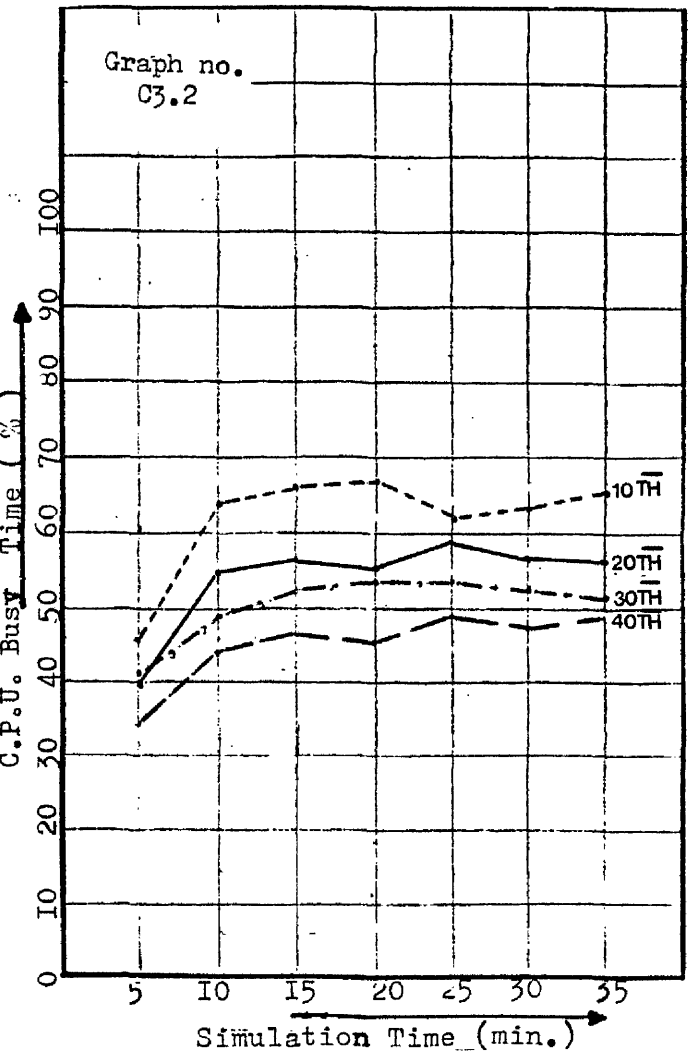
Conclusions:

* The increase in average think time gives us a 'good' response time but reduces the average interactive throughput. Barber ((/Barber 79/)) in his research work proves that the average think time can be kept within a certain effective average. This can be done by decreasing the faulty transactions (i.e. increasing operator productivity).
* For the purpose of our experiments we will fix the average think time at 30 secs.
* The increase of average think time decreases the CPU busy time. In particular, it increases the processor ideal time. This could be of benefit when the processor is bottlenecked.
* The effects of the parameter, average think time on the selected performance indices are modelled. The models can be used to predict the future changes.

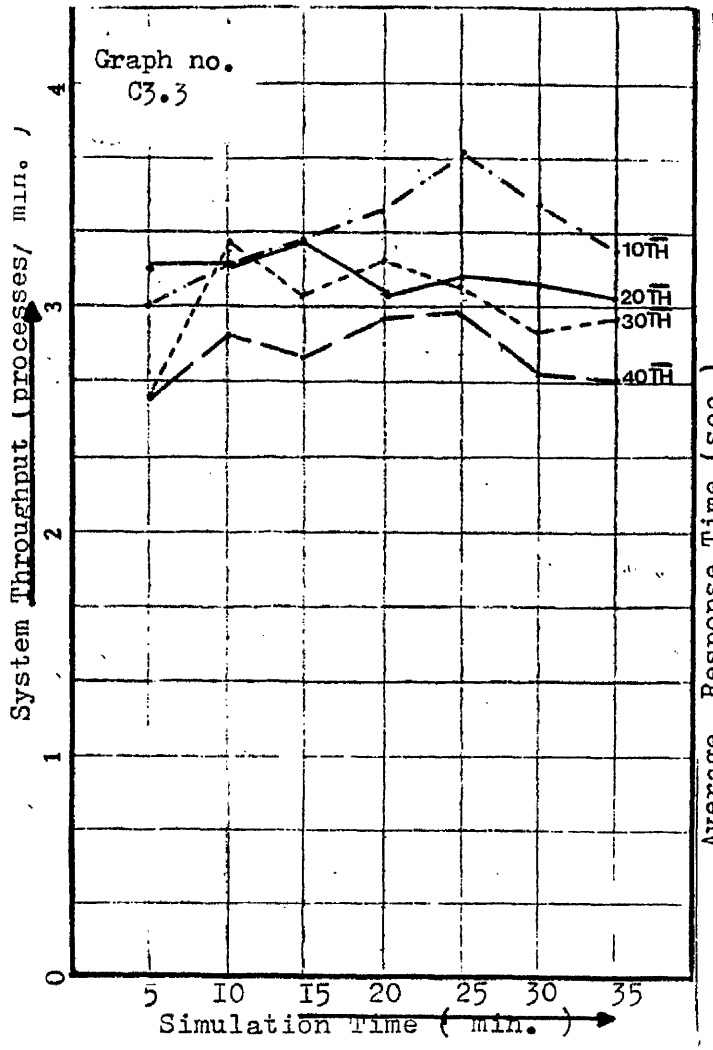
PERFORMANCE INDEX	Simulation Time (min.)	PERFORMANCE PARAMETER: The Average Think Time.									
		10 sec.		20 sec.		30 sec.		40 sec.			
Average Response Time (Sec.)	5	3.47		2.25		3.42		2.84			
	10	5.15		5.42		6.69		5.45			
	15	6.05		6.58		6.68		6.03			
	20	7.78		6.74		7.23		6.49			
	25	8.25		7.61		6.68		6.20			
	30	8.81		7.16		6.61		5.82			
	35	9.03		8.78		8.22		6.15			
C.P.U. Busy Time (O.T. + P.T.) (%)		O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.
	5	53.32	17.10	60.46	15.14	59.44	16.82	14.48	18.95		
	10	39.60	24.56	44.98	21.66	50.11	19.98	18.89	25.82		
	15	37.43	27.64	44.04	24.43	47.52	21.37	20.41	27.13		
	20	36.76	26.26	44.86	22.72	47.36	21.63	20.35	26.77		
	25	37.95	25.24	42.44	22.77	46.38	21.14	19.30	30.52		
	30	37.05	28.63	42.89	24.23	47.21	21.00	19.92	28.81		
	35	37.01	31.17	43.51	25.14	48.63	20.64	20.35	27.94		
Interactive Throughput (Processes /min)	5	3.00		3.20		2.60		2.60			
	10	3.20		3.20		3.30		2.90			
	15	3.32		3.33		3.06		2.80			
	20	3.45		3.05		3.20		2.95			
	25	3.60		3.16		3.12		2.96			
	30	3.46		3.13		2.93		2.73			
	35	3.22		3.08		2.97		2.71			

TABLE No. C3-1

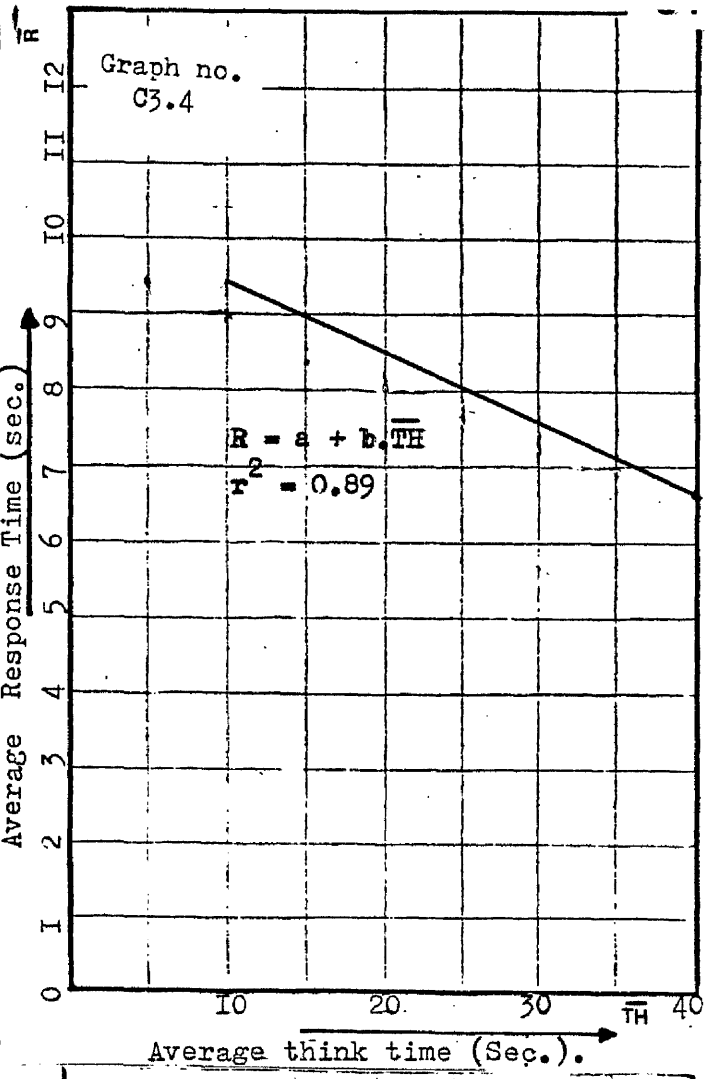




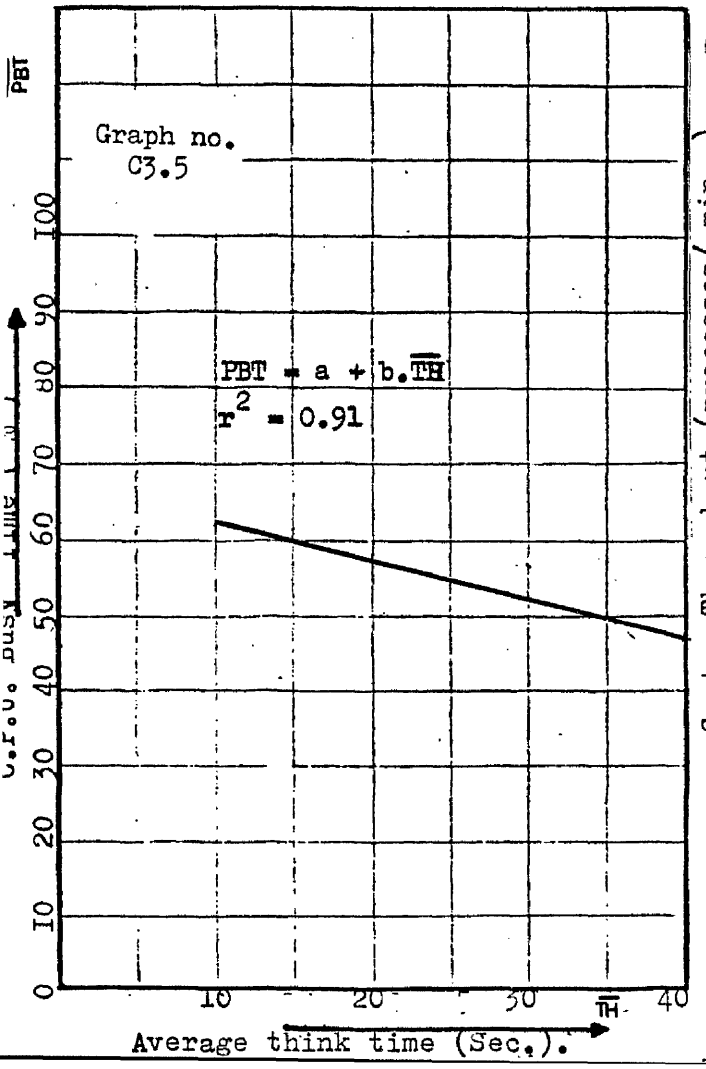
Graph no.
C3.3



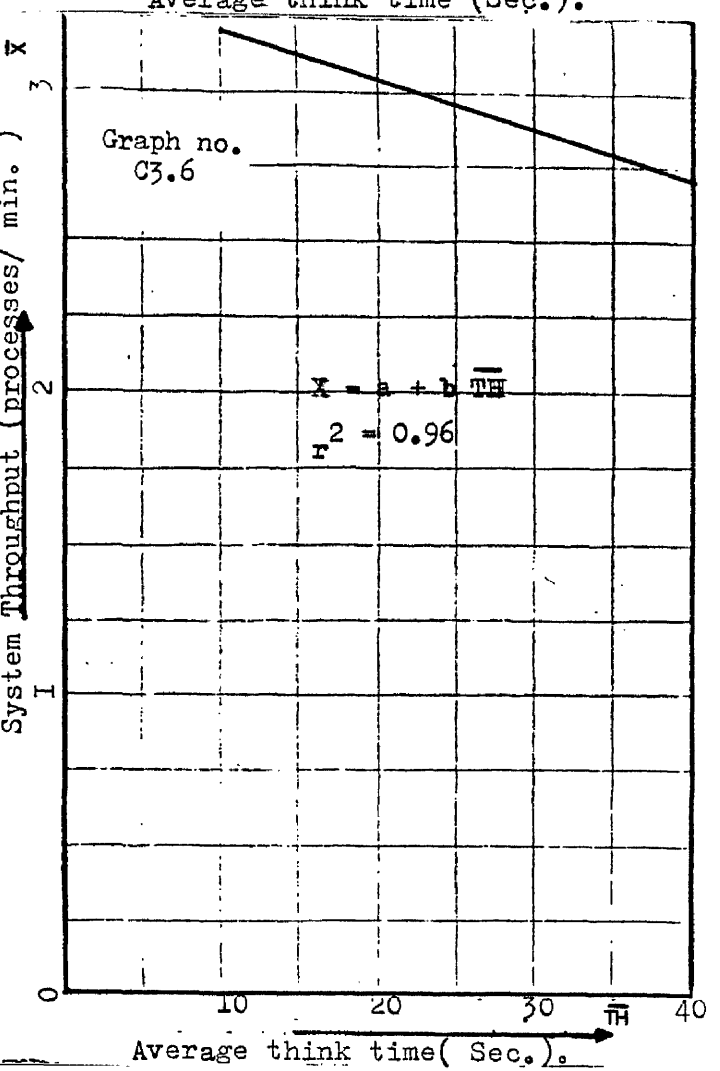
Graph no.
C3.4



Graph no.
C3.5



Graph no.
C3.6



3.3.4. Case 4: Effects of performance parameter: Mean interarrival time.

This case addresses the effects of varying the mean interarrival time (λ) on the selected performance indices (i.e. average response time ($\overline{R}$), CPU busy time ($\overline{PBT}$) and interactive throughput ($\overline{X}$)). The run of the GST was made in this case varying the workload (i.e. using λ) from 15 to 60 in step of 15. Refer to table C4.1 and graphs C4.1, C4.2, C4.3, C4.4, C4.5 and C4.6.

The relationships between the parameter, mean interarrival time and the selected performance indices can be constructed from the direct graphs (C4.1, C4.2 and C4.3) using regression analysis and under heavy loaded system condition. These relationships represent simple hybrid models which can be expressed using the relational graphs (C4.4, C4.5 and C4.6) and table C4.2.

Performance Index	Relationship Equation	Regression Constant Values	Graph No.	Equation No.
Average Response Time (sec)	$\overline{R} = ae^{b\lambda}$	a = 13.48 b = -1.87	C4.4	E4,1
CPU busy Time (%)	$\overline{PBT} = a+b\lambda$	a = 55.58 b = -0.26	C4.5	E4,2
Interactive Throughput (Processes/ min.)	$\overline{X} = ae^{b\lambda}$	a = 3.75 b = -0.09	C4.6	E4,3

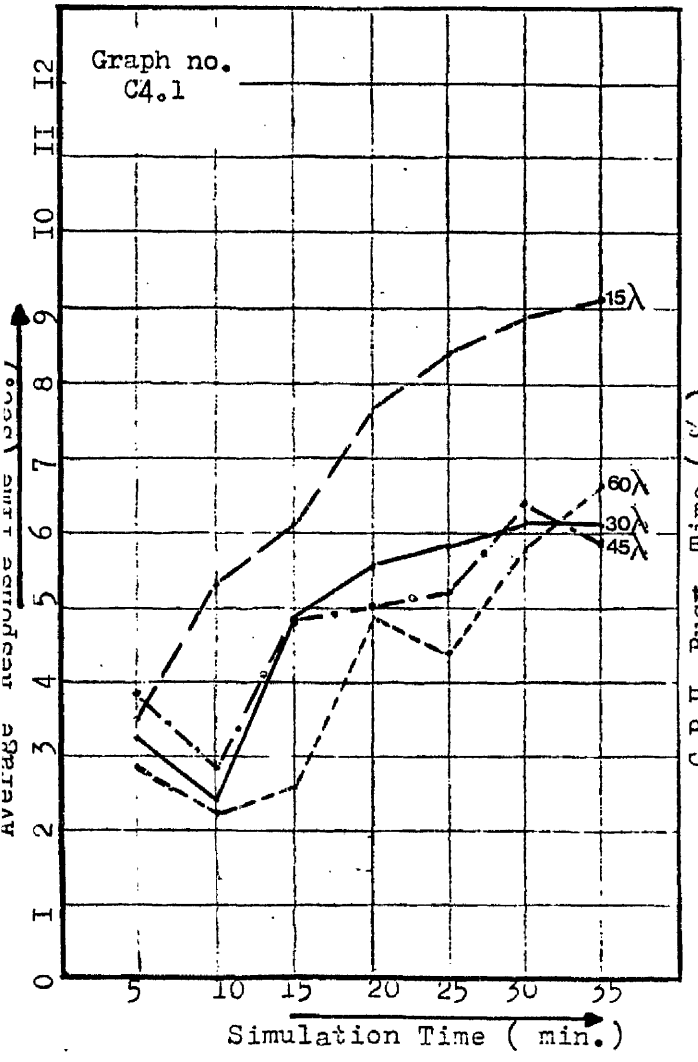
Table No. C4.2

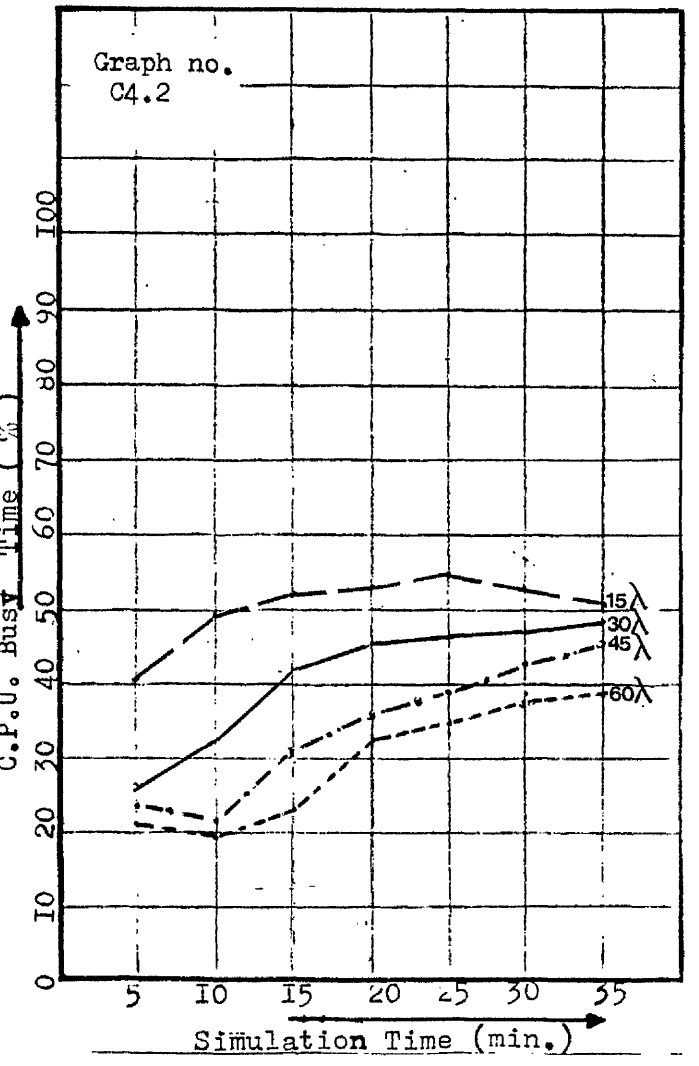
Conclusions:

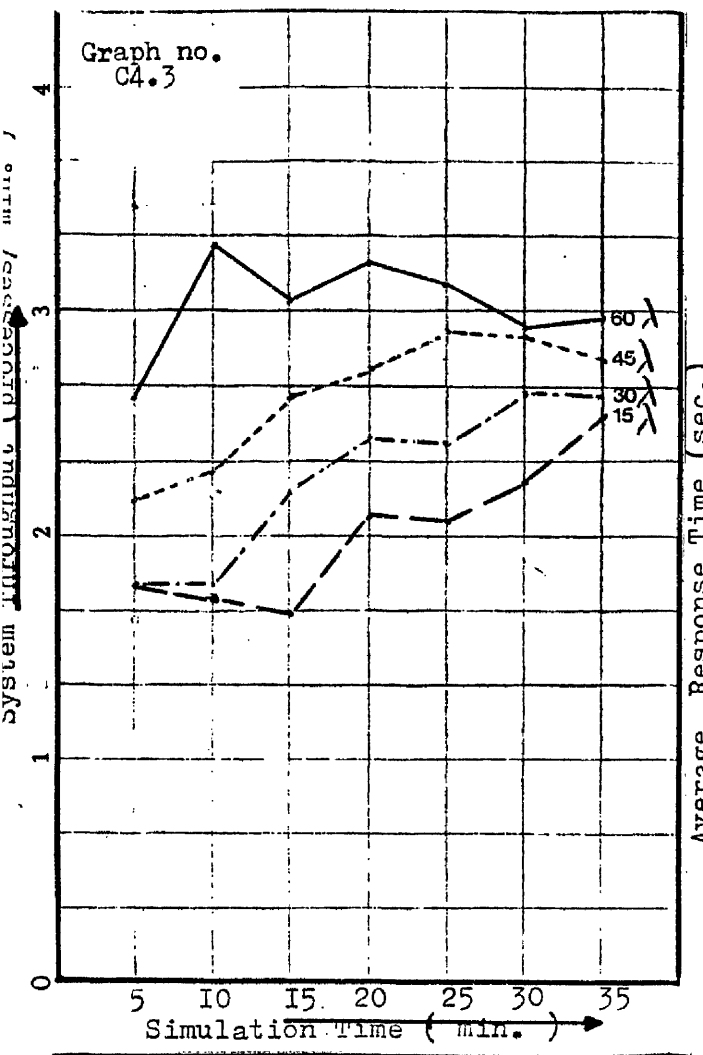
* The effects of the parameter, mean interarrival time on the selected performance indices are modelled. The models can be used to predict the future changes.
* The increase of the mean interarrival time decreases the average response time and decreases the average interactive throughput.
* Increasing the mean interarrival time decreases the CPU busy time. This may be of benefit to the system if the CPU was bottlenecked.
* For the purpose of our experiments we will fix the average mean interarrival time to 15 secs.

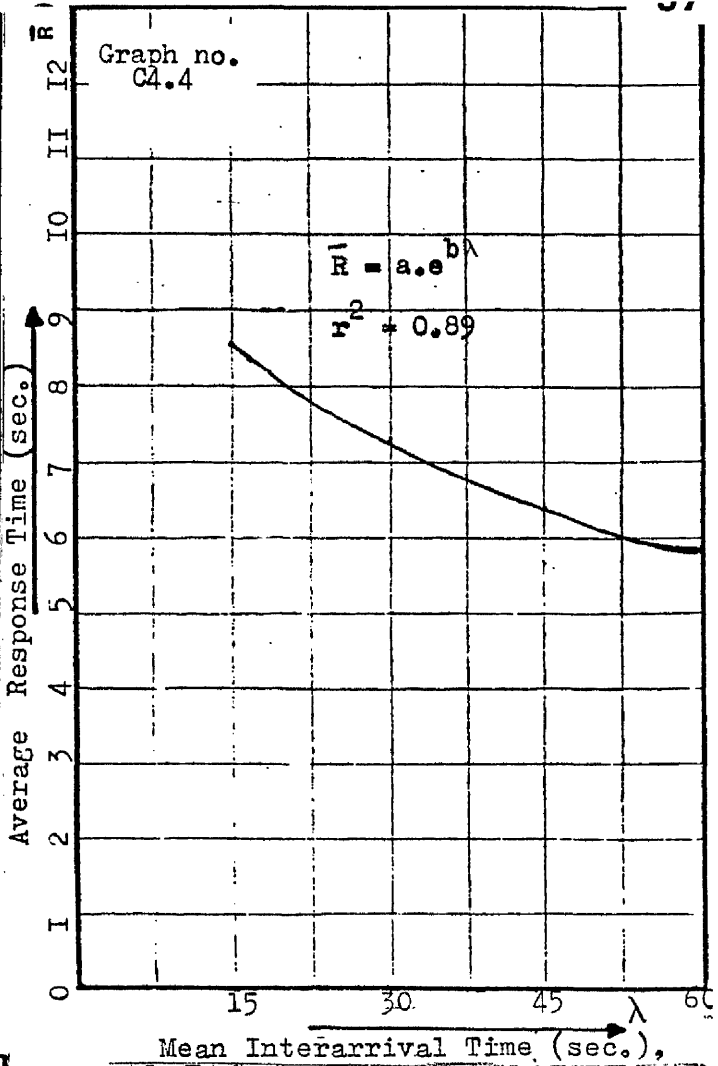
PERFORMANCE INDEX	Simulation Time (min.)	PERFORMANCE PARAMETER: Mean Interarrival Time.									
		15		30		45		60			
Average Response Time (Sec.)	5	3.47		3.24		3.87		2.82			
	10	5.15		2.38		2.76		2.21			
	15	6.05		4.85		4.85		2.65			
	20	7.78		5.54		5.03		4.90			
	25	8.25		5.85		5.19		4.37			
	30	8.81		6.17		6.40		5.88			
	35	9.03		6.09		5.97		6.68			
C.P.U. Busy Time (O.T. + P.T.) (%)		O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.	O.T.	P.T.
	5	53.32	17.10	7.62	18.78	6.76	15.84	79.74	5.94		
	10	39.60	24.56	10.85	20.95	6.72	14.96	80.51	5.71		
	15	37.43	27.64	14.87	26.51	9.25	21.59	77.86	6.50		
	20	36.76	26.26	16.55	27.02	11.60	25.03	68.75	8.50		
	25	37.95	25.24	16.61	28.29	12.72	26.40	66.05	9.95		
	30	37.05	28.63	16.64	29.30	13.22	28.77	63.86	10.95		
	35	37.01	31.17	18.76	29.08	14.97	28.07	60.57	12.37		
Interactive Throughput (Processes /min)	5	2.60		2.20		1.80		1.80			
	10	3.30		2.30		1.80		1.70			
	15	3.06		2.60		2.20		1.66			
	20	3.20		2.75		2.45		2.15			
	25	3.12		2.94		1.40		2.12			
	30	2.93		2.86		2.66		2.26			
	35	2.97		2.77		2.65		2.57			

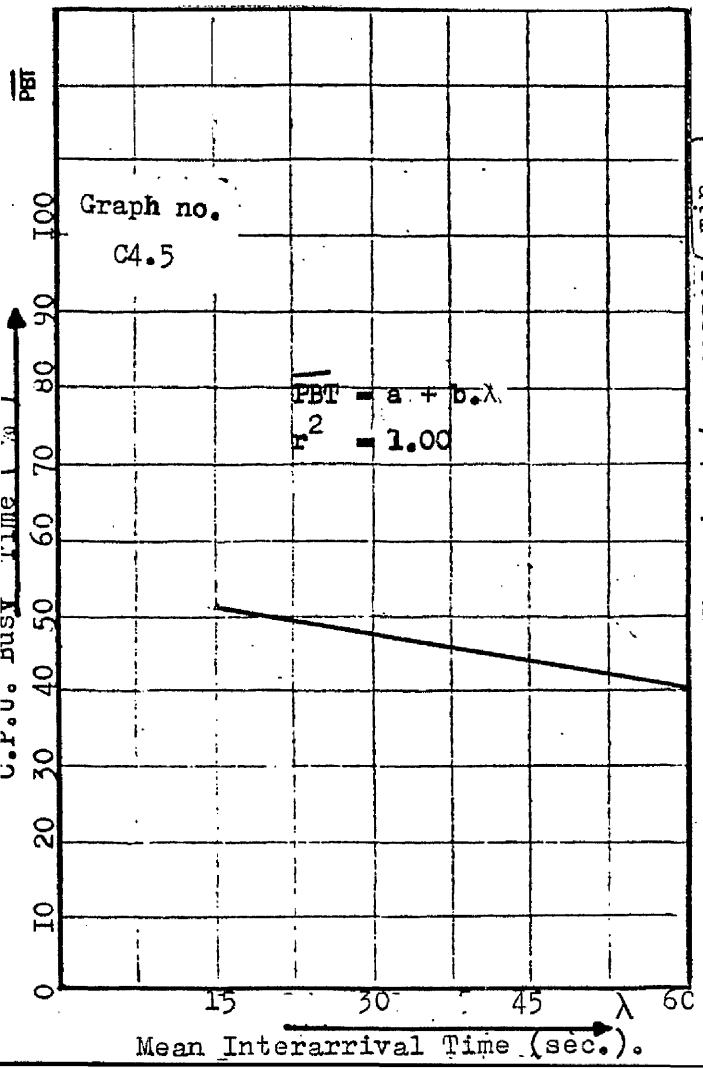
TABLE No.

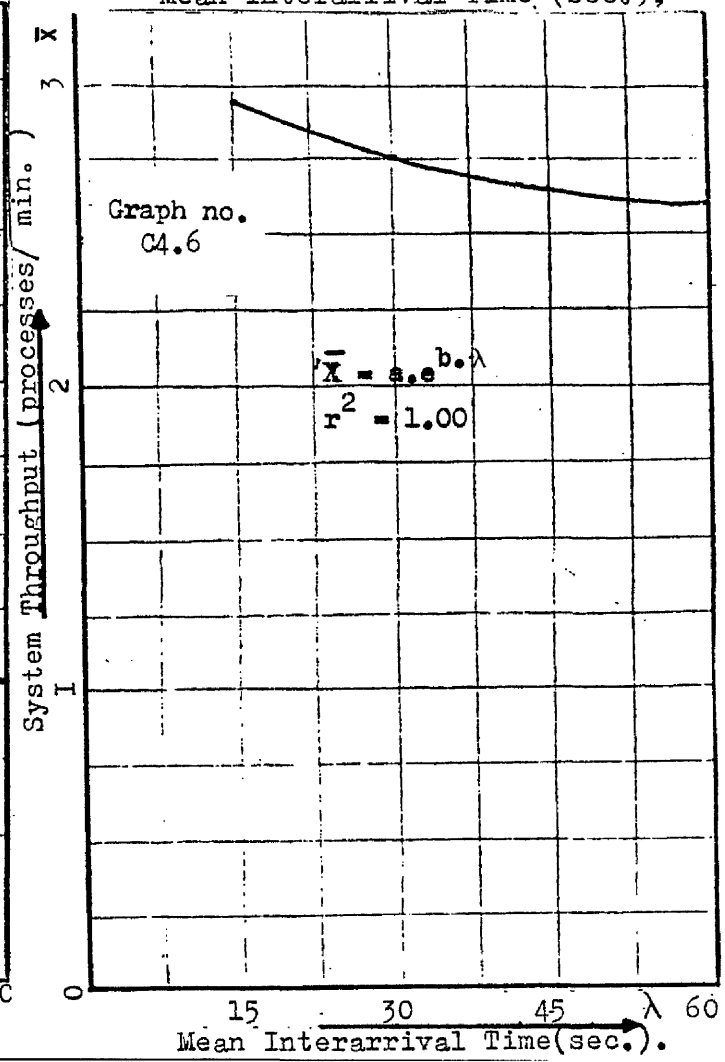





Graph no.
C4.3
System Throughput (processes/ min.)
60 λ
45 λ
30 λ
15 λ
5 10 15 20 25 30 35
Simulation Time (min.)


Graph no.
C4.4
Average Response Time (sec.)
$\bar{R} = a.e^{b\lambda}$
$r^2 = 0.89$
15 30 45 λ 60
Mean Interarrival Time (sec.).


Graph no.
C4.5
C.P.U. Busy Time
$\overline{PBT} = a + b.\lambda$
$r^2 = 1.00$
15 30 45 λ 60
Mean Interarrival Time (sec.).

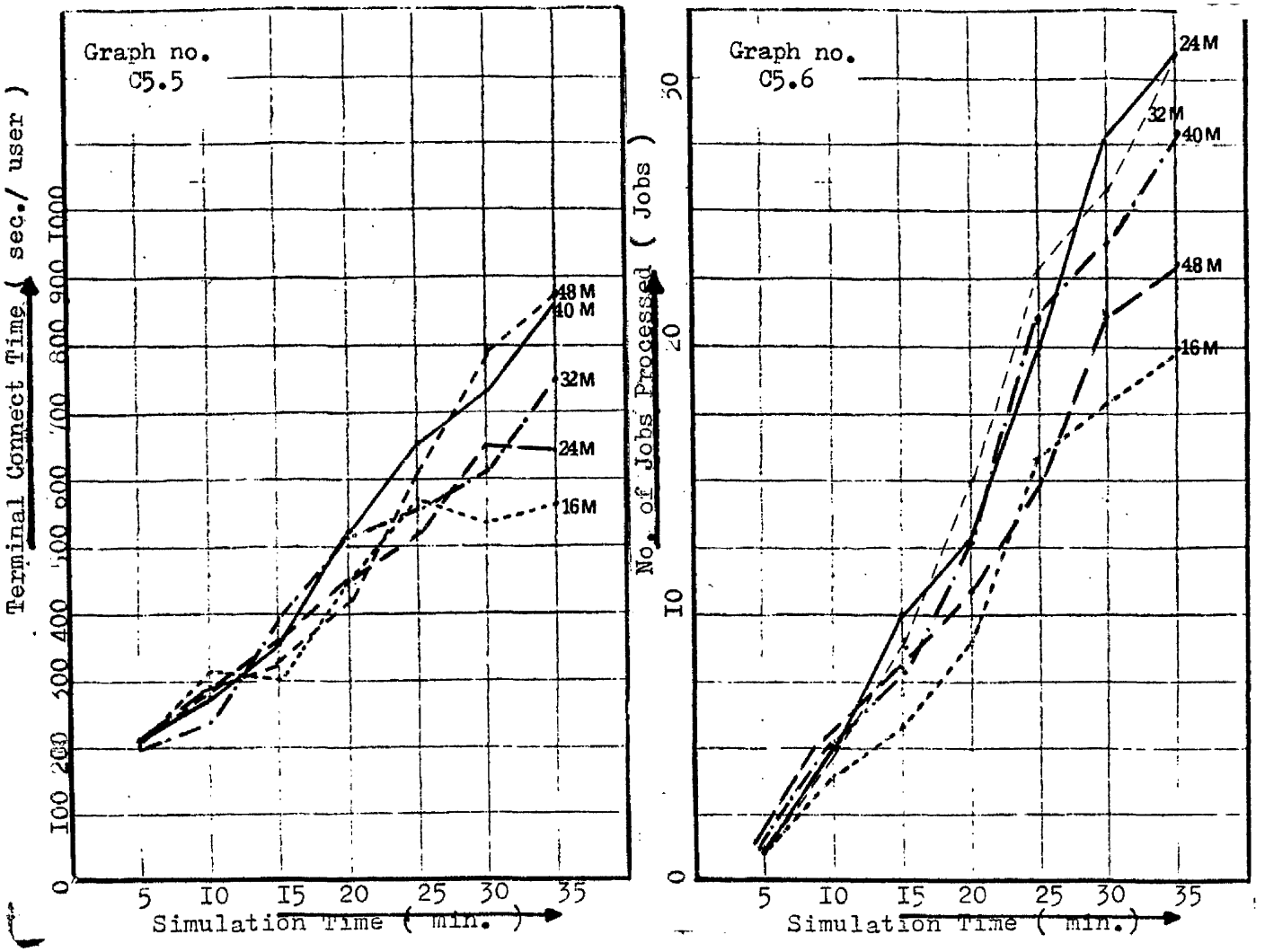

Graph no.
C4.6
System Throughput (processes/ min.)
$\bar{X} = a.e^{b.\lambda}$
$r^2 = 1.00$
15 30 45 λ 60
Mean Interarrival Time(sec.).

3.3.5. Case 5: Other effects of the performance parameter: Active number of users.

In this case, we will try to analyse the effects of the performance parameter, active number of users (M) on new selected performance indices (i.e. effective degree of multiprogramming (ψ), disc utilization (DU), drum utilization (DRU), terminal connect time (TCT), ratio of jobs processed to no. of active users (RJU) and ratio of simulation time to real time (RSR)). This can be achieved by running the GST with a workload from 16 to 48 active user. Refer to table C5.1 and graphs C5.1, C5.2, C5.3, C5.4, C5.5, C5.6, C5.7, C5.8, C5.9, C5.10, C5.11 and C5.12.

Table No. C5.1.

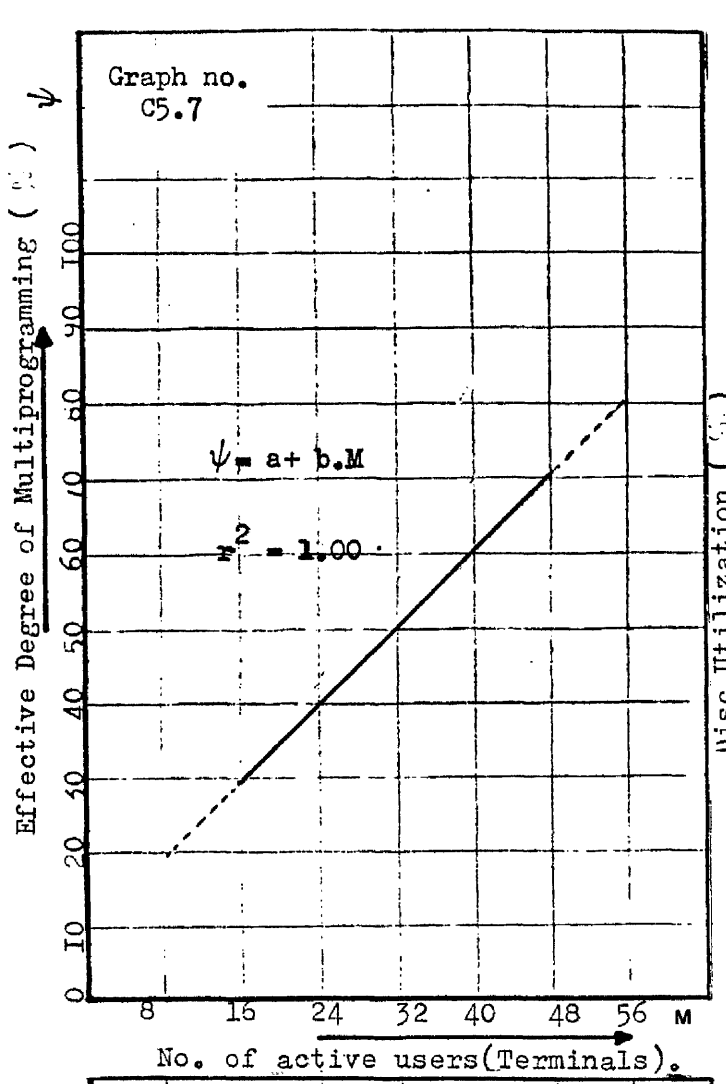
Performance Index	Simulation Level	The Performance Parameter : Number of Active Users.									
		16		24		32		40		48	
Degree of Multi-Programming (%)	5	18.84		21.69		21.26		21.56		21.51	
	10	24.22		31.64		36.97		41.23		42.62	
	15	25.99		34.86		43.15		49.94		54.57	
	20	26.48		36.75		45.81		54.08		61.35	
	25	26.81		37.41		47.15		56.58		64.98	
	30	27.08		37.73		48.39		58.63		67.78	
	35	26.99		37.60		48.81		60.10		70.14	
Device Utilization (%)		Disc	Drum	Disc	Drum	Disc	Drum	Disc	Drum	Disc	Drum
	5	79.06	0.89	78.83	1.42	86.27	4.45	79.11	1.42	79.11	1.39
	10	75.03	0.49	82.81	2.23	88.52	6.50	82.59	3.68	81.24	3.85
	15	72.73	0.40	84.81	1.98	89.28	7.50	83.50	5.18	80.73	5.78
	20	72.81	0.30	84.49	1.69	89.47	7.61	83.77	5.44	79.62	6.66
	25	72.63	0.24	84.61	1.36	89.72	7.04	83.07	5.61	79.19	6.91
	30	67.02	0.20	85.03	1.18	89.83	6.89	82.79	5.85	79.60	7.35
	35	62.41	0.17	83.98	1.05	89.94	6.59	82.41	6.01	78.58	7.61
Terminal Connect Time (Secs per user)	5	201		204		200		204		204	
	10	314		296		244		283		296	
	15	303		373		397		372		330	
	20	451		451		516		520		449	
	25	578		526		570		660		609	
	30	544		649		622		735		798	
	35	576		643		748		860		879	
Number of Multiaccess jobs processed (jobs)	5	1		1		1		1		1	
	10	4		5		5		5		5	
	15	6		10		9		8		8	
	20	9		13		15		13		11	
	25	16		20		23		21		15	
	30	18		28		26		24		21	
	35	20		31		31		28		23	
Ratio of Simulation Time to Real Time	5	2.27		2.16		1.68		2.09		2.16	
	10	2.59		1.96		1.43		1.59		1.59	
	15	2.70		1.97		1.35		1.42		1.37	
	20	2.68		1.99		1.35		1.40		1.33	
	25	2.73		2.06		1.37		1.40		1.31	
	30	2.99		2.07		1.39		1.39		1.29	
	35	3.15		2.14		1.42		1.40		1.29	

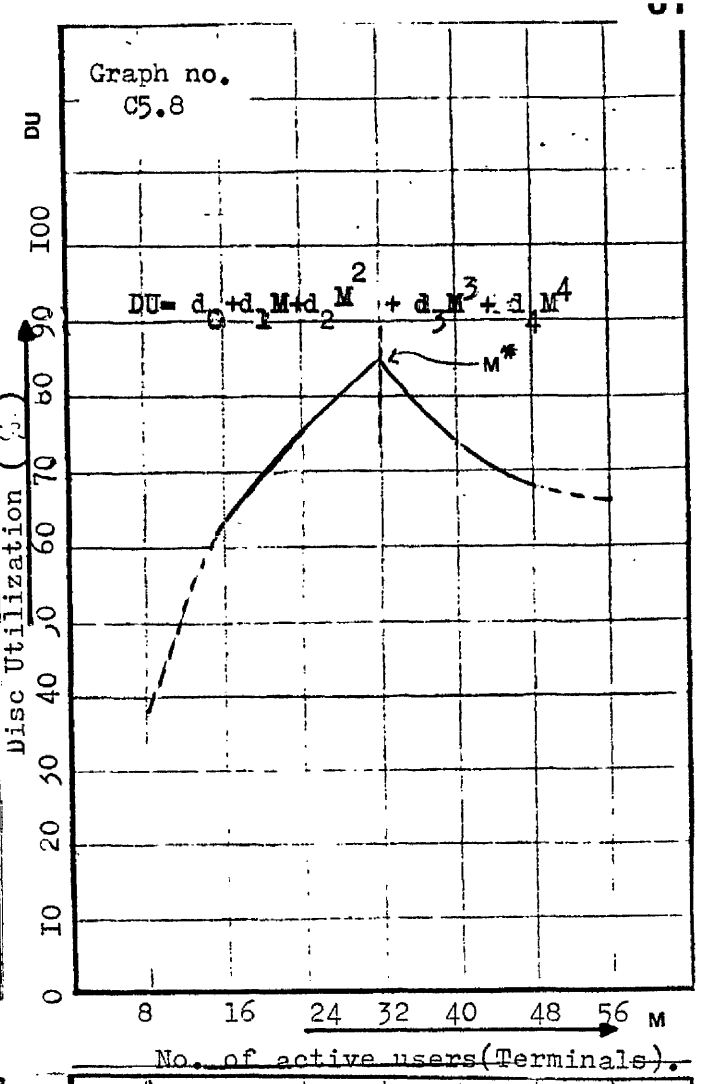


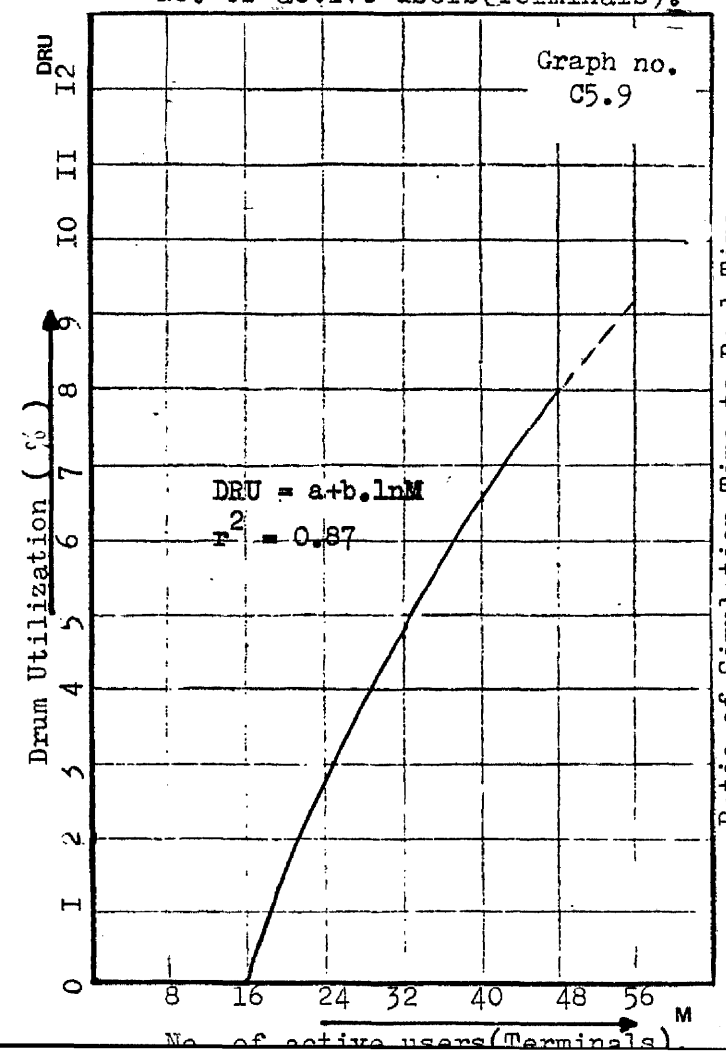
The relationships between the parameter, active no. of users and the new selected performance indices can be constructed from the direct graphs (from C5.1 to C5.6) using regression analysis and under heavy loaded system condition. These relationships represent simple hybrid models which can be expressed using the relational graphs (from C5.7 to C5.12) and table C5.2.

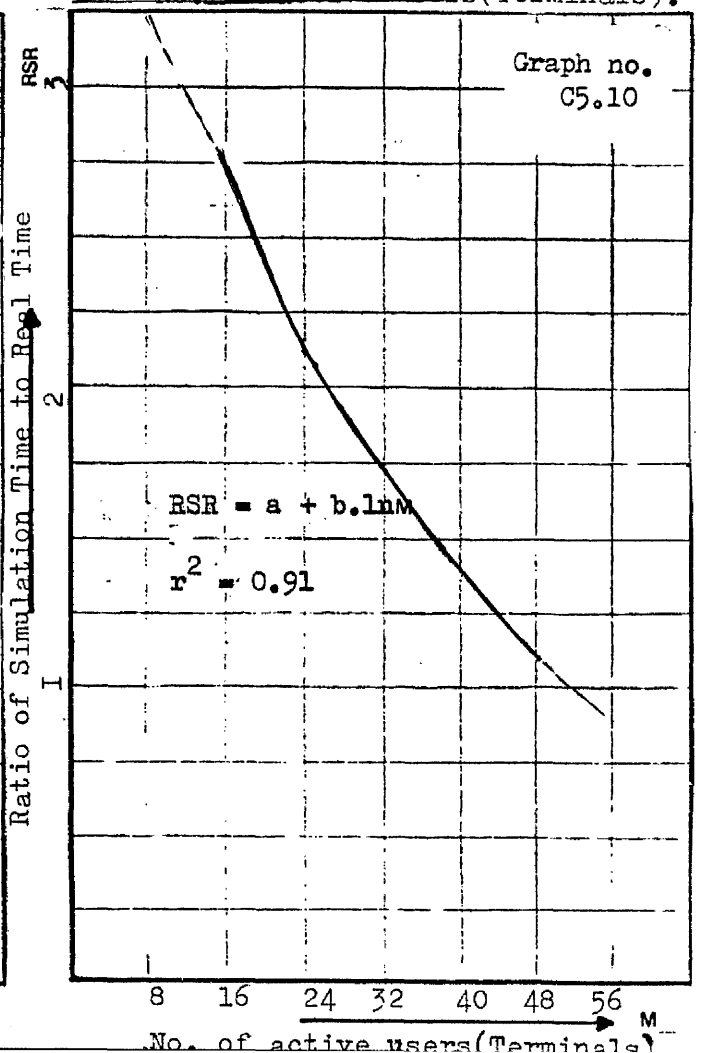
PERFORMANCE INDEX	RELATIONSHIP EQUATION	REGRESSION CONSTANT VALUES	GRAPH NO.	EQUATION NUMBER
DEGREE OF MULTIPROGRA-	$\Psi = a + bM$	$a = 5.12$, $b = 1.36$	C5.7	E1,4
DISC UTILIZATION	$DU = d_0 + d_1M + d_2M^2 + d_3M^3 + d_4M^4$	$d_0 = 31.31$, $d_1 = -3.36$, $d_2 = 0.52$, $d_3 = -0.02$, $d_4 = 0.00015$	C5.8	E1,5
DRUM UTILIZATION	$DRU = a + b \ln M$	$a = -20.60$ $b = 7.33$	C5.9	E1,6
TERMINAL CONNECT TIME	$TCT = a + bM$	$a = 416.20$ $b = 10.11$	C5.10	E1,7
RATIO OF JOBS PROCESSED/No. of active users.	$RJU = a + b \ln M$	$a = 3.45$ $b = -0.73$	C5.11	E1,8
RATIO OF SIMU. TIME TO REAL TIME	$RSR = a + b \ln M$	$a = 7.75$ $b = -1.73$	C5.12	E1,9

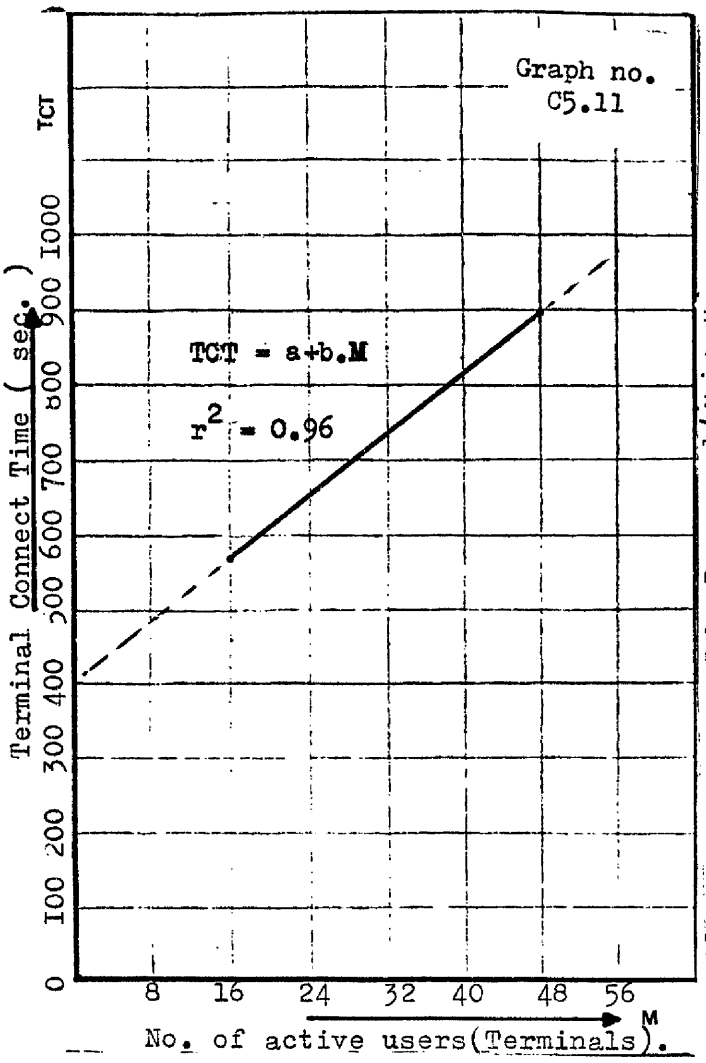
TABLE No. C5-2

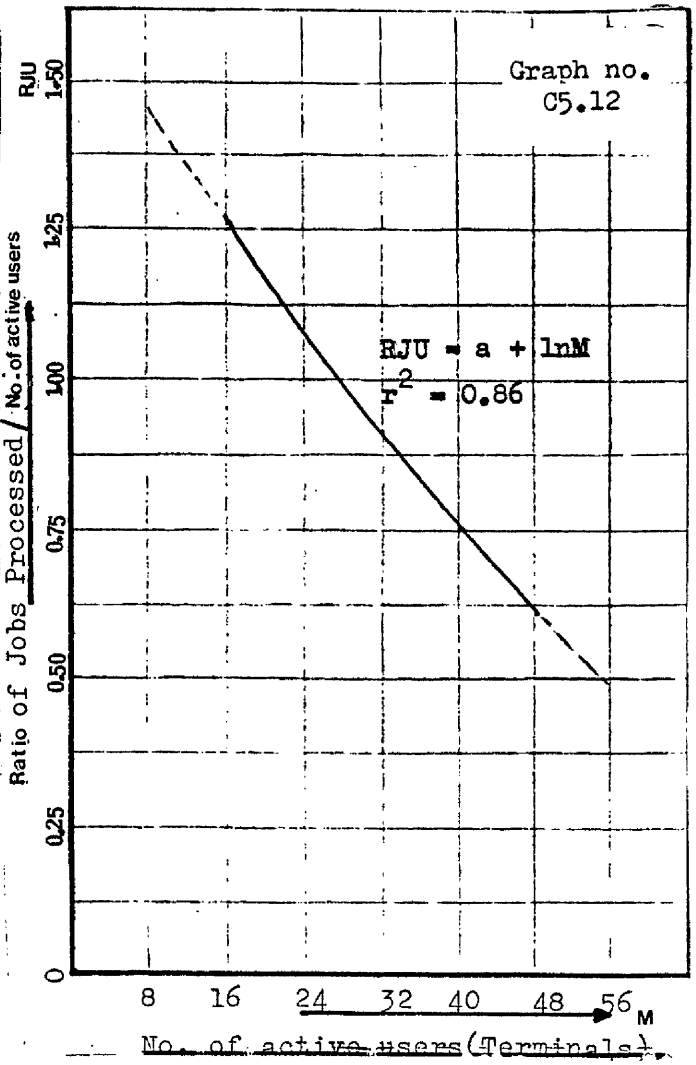

Graph no.
C5.7
Effective Degree of Multiprogramming (%) ψ
ψ = a + b.M
r² = 1.00
0 10 20 30 40 50 60 70 80 90 100
8 16 24 32 40 48 56 M
No. of active users(Terminals).


Graph no.
C5.8
Disc Utilization (%) DU
DU = d0 + d1M + d2M² + d3M³ + d4M⁴
M*
0 10 20 30 40 50 60 70 80 90 100
8 16 24 32 40 48 56 M
No. of active users(Terminals).


Graph no.
C5.9
Drum Utilization (%) DRU
DRU = a + b.lnM
r² = 0.87
0 1 2 3 4 5 6 7 8 9 10 11 12
8 16 24 32 40 48 56 M
No. of active users(Terminals).


Graph no.
C5.10
Ratio of Simulation Time to Real Time RSR
RSR = a + b.lnM
r² = 0.91
1 2 3
8 16 24 32 40 48 56 M
No. of active users(Terminals).





Conclusions:

* Since the effective degree of multiprogramming increased with the increasing number of active users in the system, this means we need to consider two things:
  1. What is the maximum degree of multiprogramming that the available main memory capacity can hold?
  2. What is the best scheduling policy that ensures the system performance will not be degraded with a given degree of multiprogramming (i.e. no thrashing for example)?

  The above considerations will be one of our future research interests.
* It seems in GST the disc is bottlenecked, since it approximately reaches its maximum utilization point. But as soon as it reaches M^* (see graph no. C5.8) , a degradation in the disc utilization occurs, and at the same time the drum started to be utilized more and more. Despite this , the drum is still under-utilized.

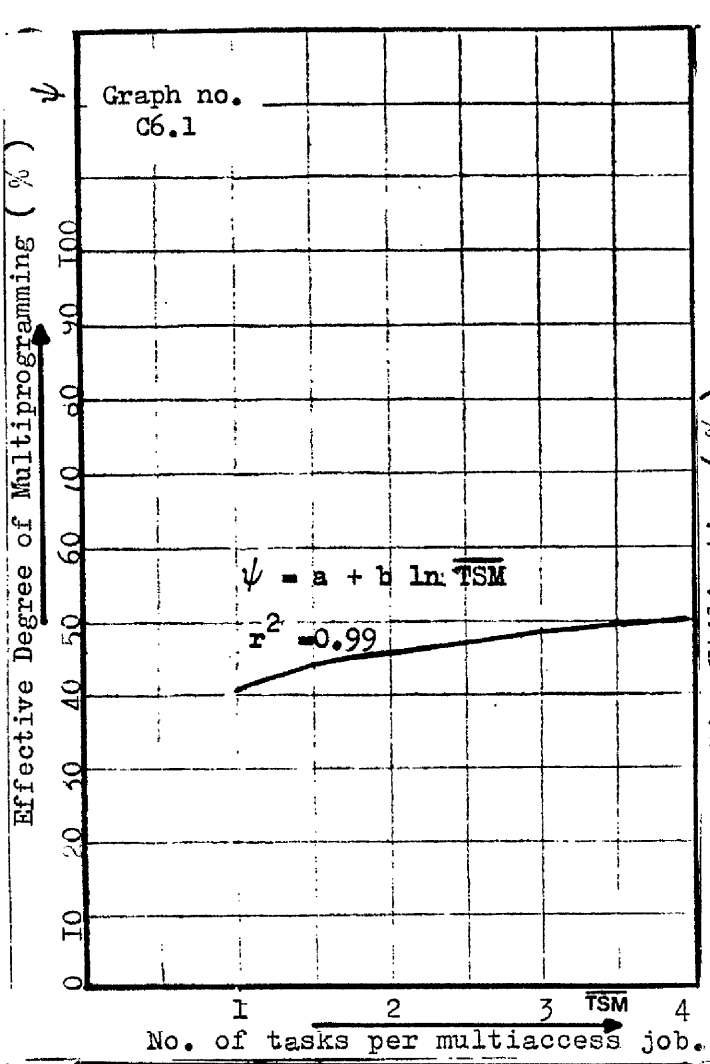
* Ratio of simulation time to real time represents an important cost-performance factor, since by this ratio we can calculate the real time required to perform the actual tasks given to the computer system. Hence, the maximum acceptable ratio should be 1. In GST the ratio = 1 when number of active users equal 50 (see graph C5.10).
* The ratio of jobs processed per no. of active users decreases with the increasing no. of active users in the system. That means the system productivity decreases with the increasing no. of active users in the system.
* The effects of the parameter, number of actve users on the new selected performance indices are modelled. The models can be used to predict the future changes.
* For the purpose of the next experiments, the direct graphs will not be drawn, since it can be directly constructed from the first table in these experiments.

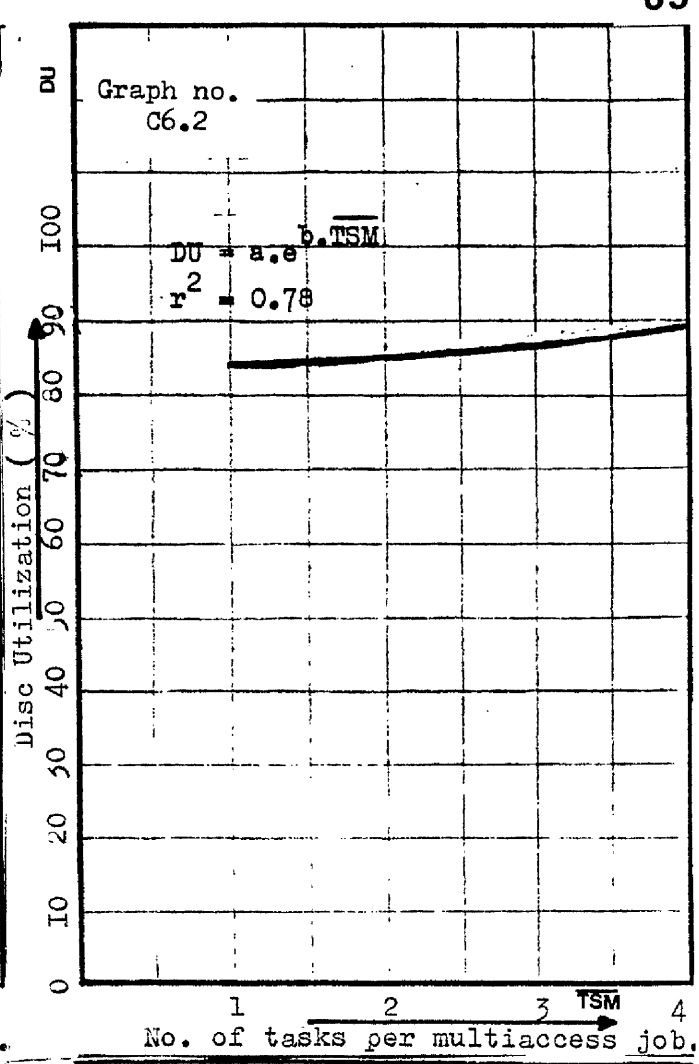
3.3.6. Case 6: Other effects of the performance parameter: No. of tasks per multiaccess job.

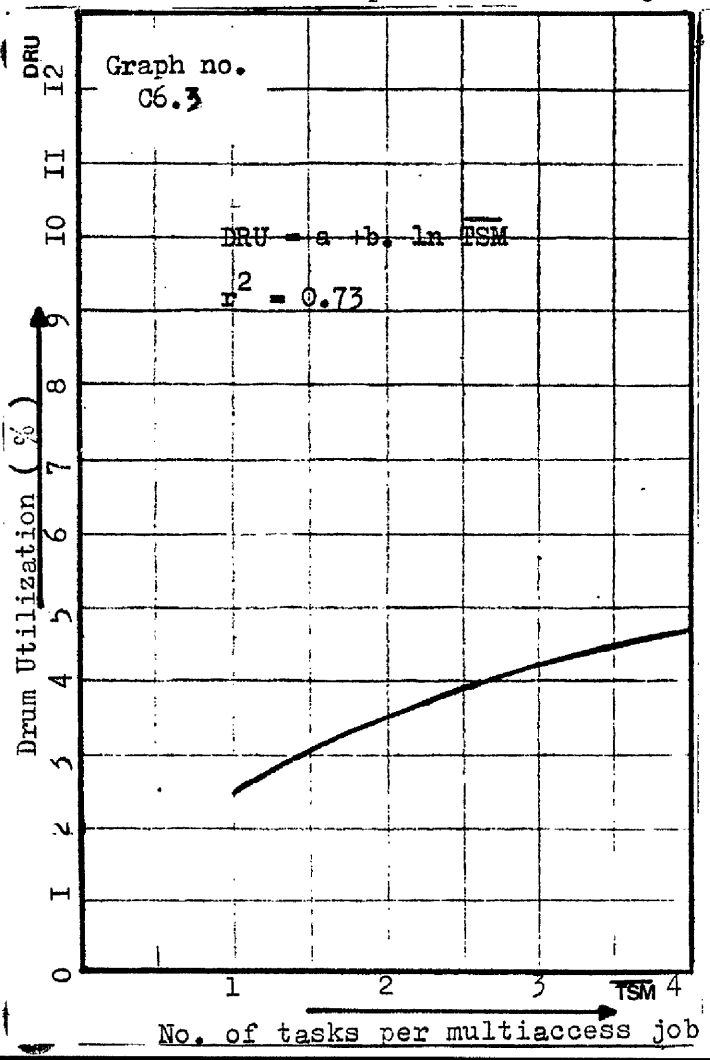
This case study addresses the effects of varying the average no. of tasks per multiaccess job ($\overline{TSM}$) on the new selected performance indices (i.e. effective degree of multiprogramming (ψ), disc utilization (DU), drum utilization (DRU), terminal connect time (TCT), no. of multi-access jobs processed (MJP) and ratio of simulation time to real time (RSR). This can be achieved by running the GST with a workload of average no. of tasks per multiaccess job from 1 to 4 in step of 1. Refer to table C6.1 and graphs C6.1, C6.2, C6.3, C6.4, C6.5 and C6.6.

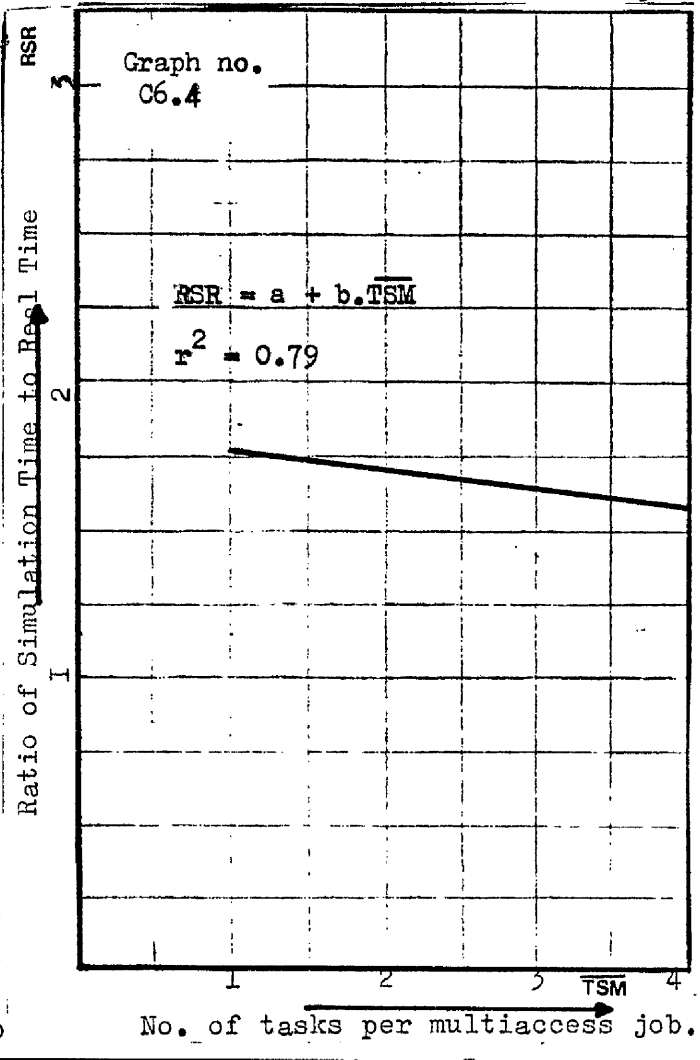
Table No. C6.1

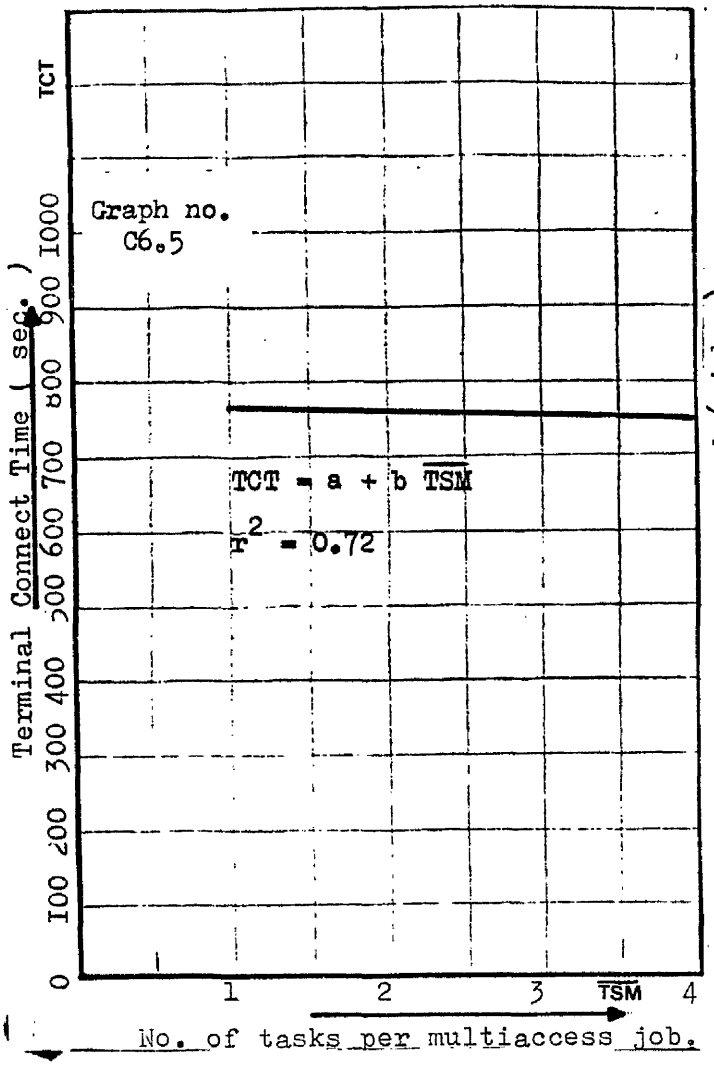
Performance Index	Simulation Time (Mins)	The Performance Parameter : No. of tasks per multiaccess job.							
		1		2		3		4	
Degree of Multi-Programming (%)	5	18.25		19.03		21.26		21.46	
	10	31.75		34.09		36.97		37.78	
	15	36.72		40.16		43.15		44.44	
	20	38.68		42.31		45.81		47.57	
	25	39.77		43.74		47.15		49.27	
	30	40.35		44.64		48.39		50.48	
	35	40.74		45.03		48.81		50.98	
Device Utilization (%)		Disc	Drum	Disc	Drum	Disc	Drum	Disc	Drum
	5	81.08	0.00	85.63	0.261	86.27	4.45	82.89	0.40
	10	83.13	1.49	86.33	1.64	88.52	6.56	84.86	2.35
	15	84.20	2.24	86.41	2.404	89.28	7.50	85.99	3.24
	20	85.35	2.11	87.37	2.21	89.47	7.61	86.97	3.37
	25	86.52	1.74	87.69	1.99	89.72	7.04	87.17	3.13
	30	86.34	2.13	86.77	2.46	89.83	6.89	86.54	3.24
	35	86.20	2.35	86.09	2.74	89.94	6.59	86.86	3.19
Terminal Connect Time (Secs per user)	5	195		203		200		195	
	10	223		249		244		256	
	15	339		397		397		386	
	20	569		539		516		528	
	25	578		592		570		597	
	30	690		645		622		644	
	35	780		767		748		771	
Number of Multiaccess jobs processed (jobs)	5	1		1		1		1	
	10	4		4		5		5	
	15	7		8		9		8	
	20	16		15		15		15	
	25	22		23		23		23	
	30	27		26		26		26	
	35	30		30		31		31	
Ratio of Simulation Time to Real Time	5	2.47		2.20		1.68		2.33	
	10	1.99		1.82		1.43		1.82	
	15	1.82		1.73		1.35		1.70	
	20	1.79		1.74		1.35		1.68	
	25	1.82		1.75		1.37		1.68	
	30	1.82		1.75		1.39		1.70	
	35	1.83		1.76		1.42		1.74	

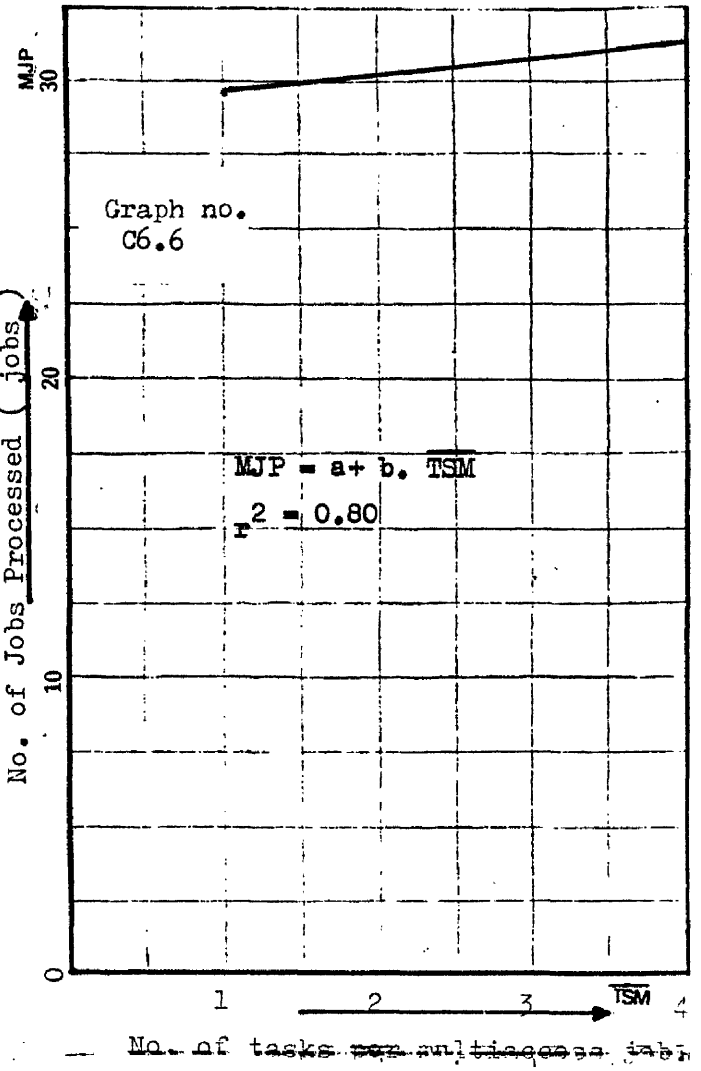

Graph no.
C6.1
Effective Degree of Multiprogramming (%)
ψ
ψ = a + b ln TSM
r² = 0.99
TSM
No. of tasks per multiaccess job.


Graph no.
C6.2
Disc Utilization (%)
DU
DU = a.e^(b.TSM)
r² = 0.78
TSM
No. of tasks per multiaccess job.


Graph no.
C6.3
Drum Utilization (%)
DRU
DRU = a + b. ln TSM
r² = 0.73
TSM
No. of tasks per multiaccess job


Graph no.
C6.4
Ratio of Simulation Time to Real Time
RSR
RSR = a + b.TSM
r² = 0.79
TSM
No. of tasks per multiaccess job.





The relationships betweem the parameter, average no. of tasks per multiaccess job and the new performance indices can be constructed from table C6.1 using regression analysis and under heavy loaded system condition. These relationships represent simple hybrid models which can be expressed using the relational graphs (from C6.1 to C6.6) and table C6.2.

PERFORMANCE INDEX	RELATIONSHIP EQUATION	REGRESSION CONSTANT VALUES	GRAPH NO.	EQUATION NUMBER
DEGREE OF MULTIPROGRA-	$\Psi = a + b \ln \overline{TSM}$	a = 40.47 b = 7.45	C6.1	E2,4
DISC UTILIZATION	$DU = a e^{b \overline{TSM}}$	a = 85.82 b = 0.01	C6.2	E2,5
DRUM UTILIZATION	$DRU = a + b \ln \overline{TSM}$	a = 2.44 b = 1.61	C6.3	E2,6
TERMINAL CONNECT TIME	$TCT = a + b \overline{TSM}$	a = 778 b = -4.60	C6.4	E2,7
No. OF JOBS PROCESSED	$MJP = a + b \overline{TSM}$	a = 29.50 b = 0.40	C6.5	E2,8
RATIO OF SIMU- TIME TO REAL	$RSR = a + b \overline{TSM}$	a = 1.84 b = -0.06	C6.6	E2,9

TABLE No. C6-2

Conclusions:

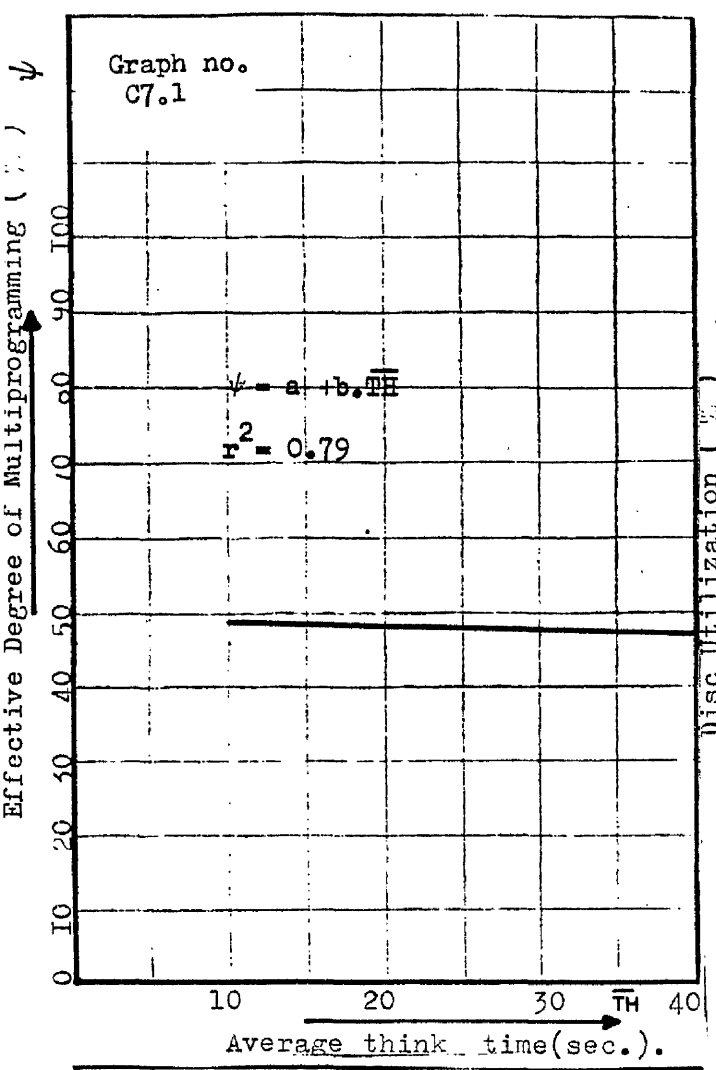
* The analyst will realize that this parameter slightly affects all the new performance indices (ψ, DU,DRU,TCT,MJP andRSR).
* The same parameter (i.e. average no. of tasks per multiaccess job) also has slight effects on the previous performance indices ($\overline{R}$, $\overline{X}$, $\overline{PBT}$). Hence, the increases of this parameter will not directly affect the computer system performance. This is quite important, since this parameter depends on the user (see graphs C2.4, C2.5 and C2.6).
* The effects of the parameter, average no. of tasks per multiaccess job on the new selected performance indices are modelled. The models can be used to predict the future changes.

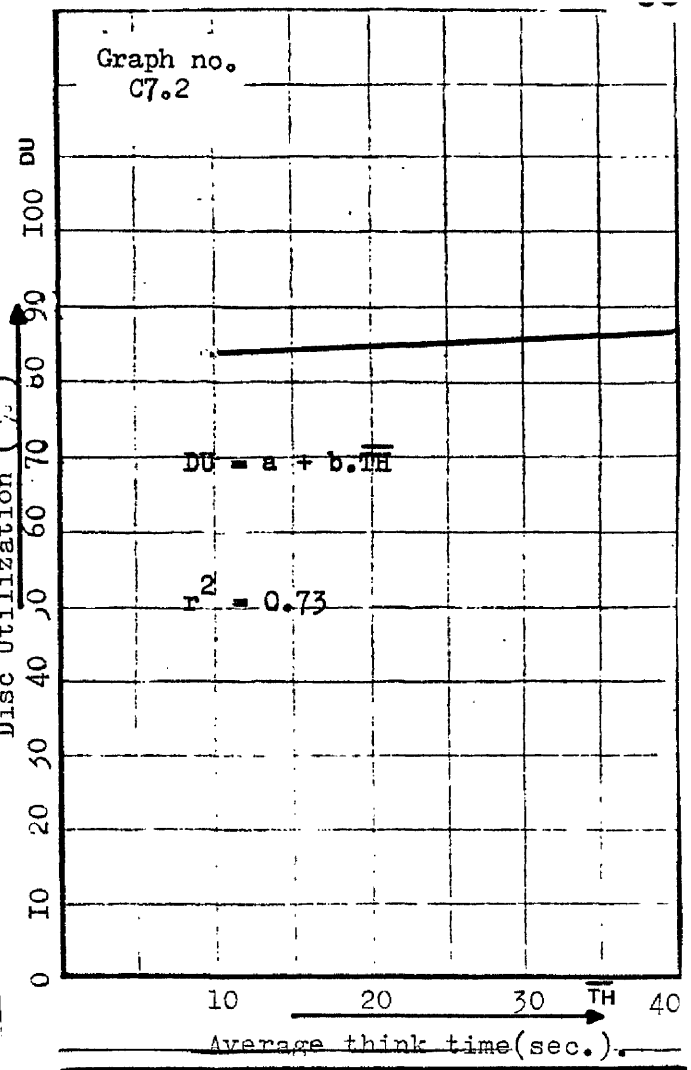
3.3.7. Case 7: Other effects of the parameter: Average think time.

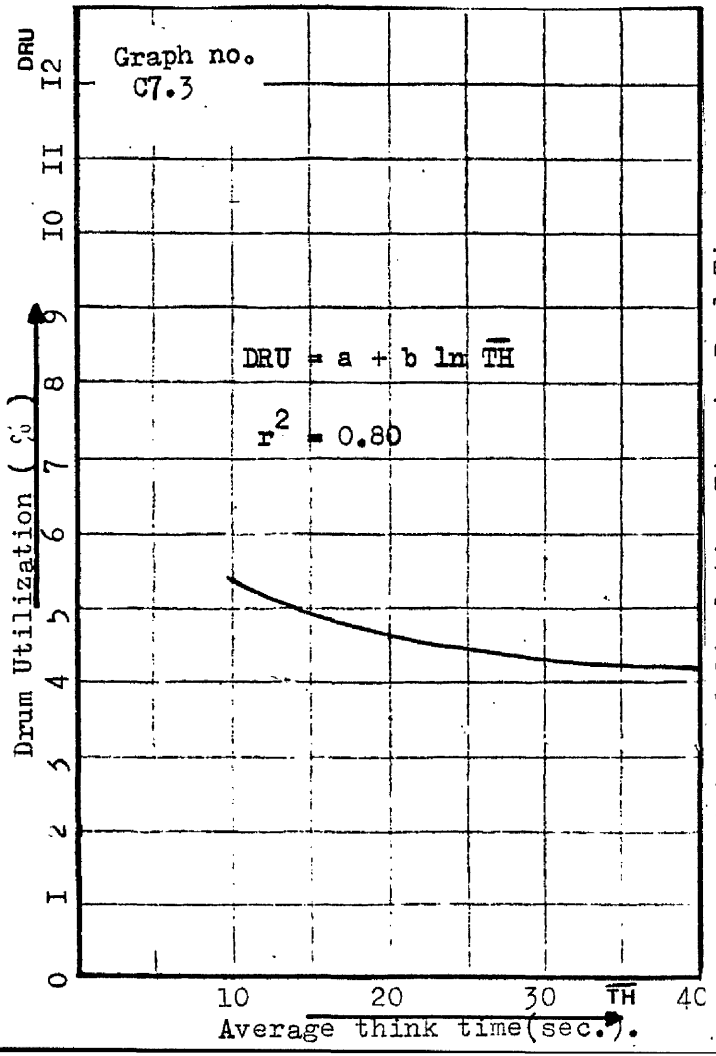
This case study addresses the effects of varying the average think time ($\overline{TH}$) on the new selected performance indices (i.e. Ψ, DU, DRU, TCT, MJP and RSR). This can be achieved by running the GST with a workload of average think time from 10 to 40 secs in step of 10. Refer to table C7.1 and graphs C7.1, C7.2, C7.3, C7.4, C7.5 and C7.6.

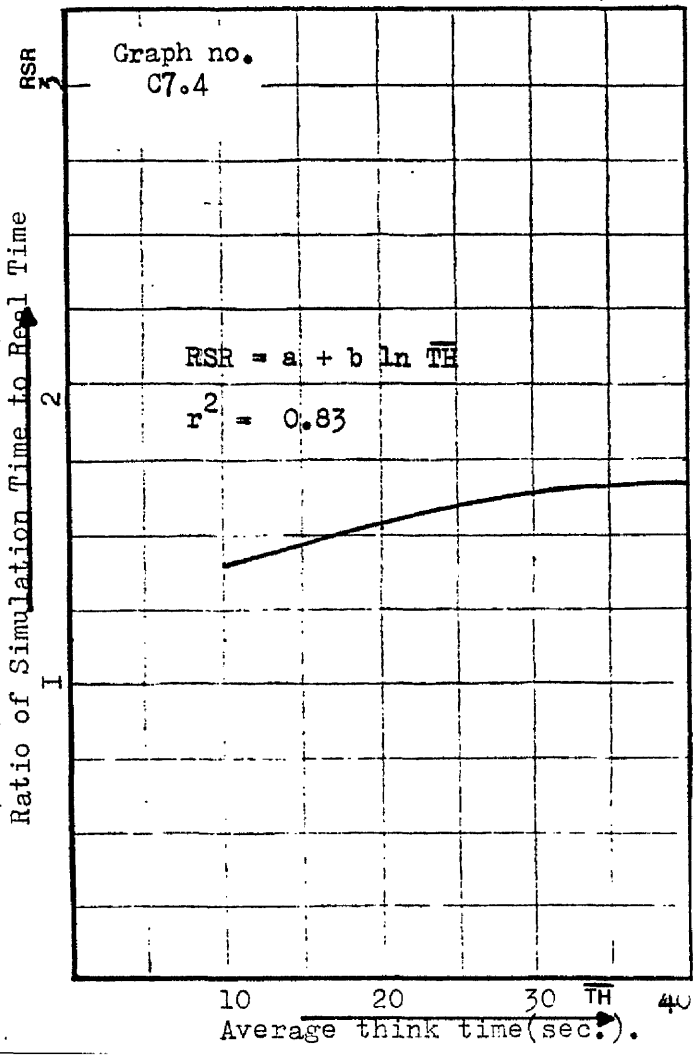
Table No. C7.1

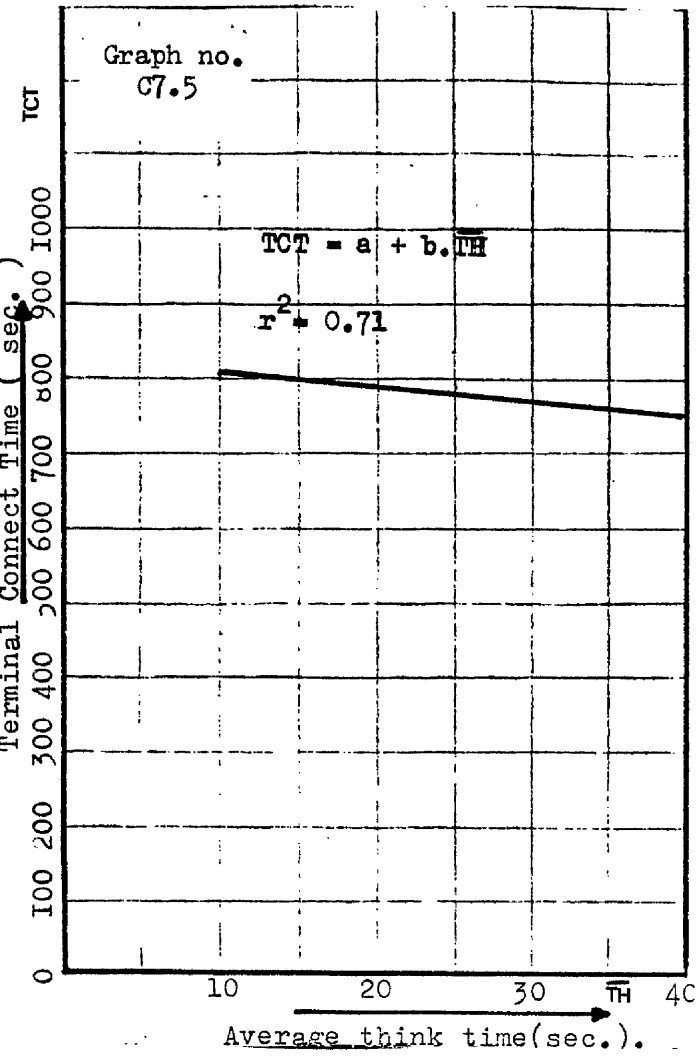
Performance Index	Simulation Time (Mins)	The Performance Parameter : Average think time ($\overline{TH}$)							
		10 sec.		20 sec.		30 sec.		40 sec.	
Degree of Multi-Programming (%)	5	22.08		21.57		21.26		21.12	
	10	38.18		37.80		36.97		36.89	
	15	43.85		43.42		43.15		43.13	
	20	46.25		46.14		45.81		45.90	
	25	47.14		47.51		47.15		47.18	
	30	47.91		48.41		48.39		48.27	
	35	49.14		48.83		48.81		48.94	
Device Utilization (%)		Disc	Drum	Disc	Drum	Disc	Drum	Disc	Drum
	5	83.65	1.78	86.30	0.99	86.27	4.45	74.53	1.12
	10	82.72	3.43	85.06	2.61	88.52	6.50	80.12	2.14
	15	83.45	4.75	84.65	3.42	89.28	7.50	82.62	2.82
	20	84.57	4.41	85.78	3.14	89.47	7.61	83.75	2.78
	25	84.98	3.99	85.38	3.06	89.72	7.04	83.42	2.51
	30	83.86	4.88	84.95	3.50	89.83	6.89	83.97	2.69
	35	82.64	5.35	84.64	3.70	89.94	6.59	84.24	2.78
Terminal Connect Time (Secs per user)	5	167		173		200		231	
	10	239		249		244		269	
	15	401		376		397		406	
	20	512		489		516		488	
	25	680		602		570		557	
	30	774		667		622		631	
	35	788		828		748		767	
Number of Multiaccess Jobs processed (jobs)	5	1		1		1		1	
	10	6		6		5		4	
	15	10		10		9		8	
	20	16		14		15		13	
	25	24		21		23		20	
	30	33		26		26		24	
	35	34		33		31		28	
Ratio of Simulation Time to Real Time	5	1.84		2.07		1.68		2.31	
	10	1.52		1.65		1.43		1.92	
	15	1.42		1.58		1.35		1.78	
	20	1.41		1.59		1.35		1.80	
	25	1.43		1.59		1.37		1.80	
	30	1.41		1.59		1.39		1.81	
	35	1.42		1.61		1.42		1.83	

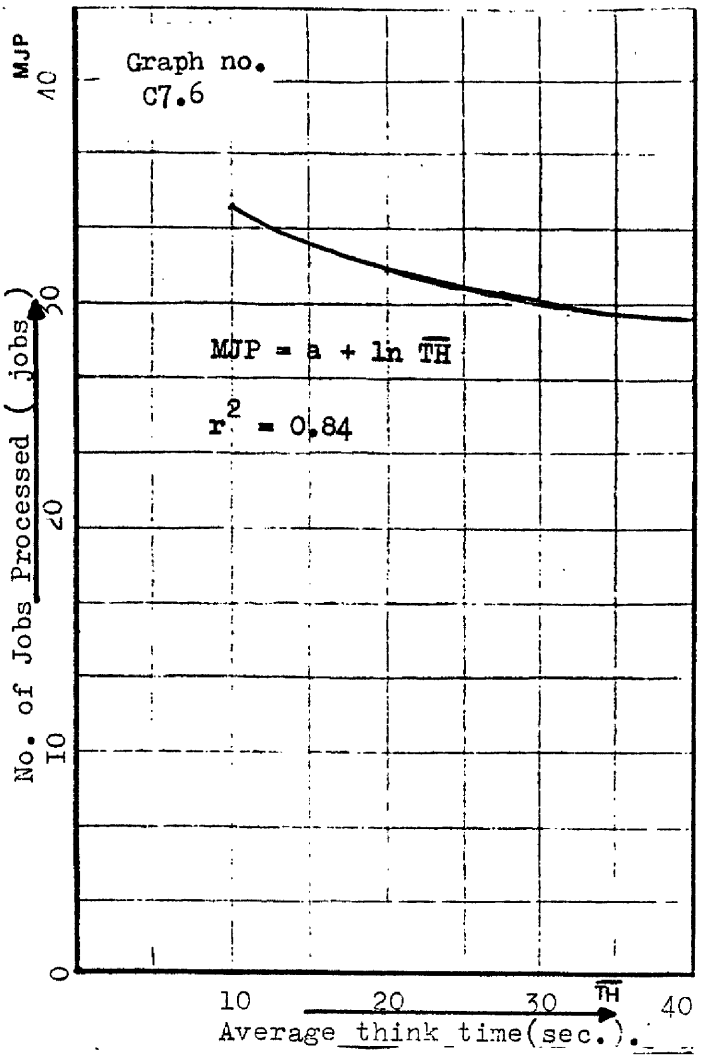

Graph no.
C7.1
Effective Degree of Multiprogramming (ψ)
ψ = a + b.TH
r² = 0.79
0 10 20 30 40 50 60 70 80 90 100
10 20 30 TH 40
Average think time(sec.).


Graph no.
C7.2
Disc Utilization (DU)
DU = a + b.TH
r² = 0.73
0 10 20 30 40 50 60 70 80 90 100
10 20 30 TH 40
Average think time(sec.).


Graph no.
C7.3
Drum Utilization (DRU)
DRU = a + b ln TH
r² = 0.80
0 1 2 3 4 5 6 7 8 9 10 11 12
10 20 30 TH 40
Average think time(sec.).


Graph no.
C7.4
Ratio of Simulation Time to Real Time
RSR = a + b ln TH
r² = 0.83
1 2 3
10 20 30 TH 40
Average think time(sec.).





The relationship between the parameter, average think time and the new performance indices can be constructed from table C7.1 using regression analysis and under heavy loaded system condition. These relationships represent simple hybrid models which can be expressed using the relational graphs (from C7.1 to C7.6) and table C7.2.

PERFORMANCE INDEX	RELATIONSHIP EQUATION	REGRESSION CONSTANT VALUES	GRAPH NO.	EQUATION NUMBER
DEGREE OF MULTIPROGRA-	$\Psi = a + b\,\overline{TH}$	a = 49.09 b = −0.01	C7.1	E3,4
DISC UTILIZATION	$DU = a + b\,\overline{TH}$	a = 82.89 b = 0.10	C7.2	E3,5
DRUM UTILIZATION	$DRU = a + b \ln \overline{TH}$	a = 7.39 b = −0.90	C7.3	E3,6
TERMINAL CONNECT TIME	$TCT = a + b\,\overline{TH}$	a = 818.50 b = −1.43	C7.4	E3,7
No. OF JOBS PROCESSED	$MJP = a + b \ln \overline{TH}$	a = 0.84 b = −402	C7.5	E3,8
RATIO OF SIMU. TIME TO REAL	$RSR = a + b \ln \overline{TH}$	a = 0.93 b = 0.21	C7.6	E3,9

TABLE No. C7.2

Conclusions:

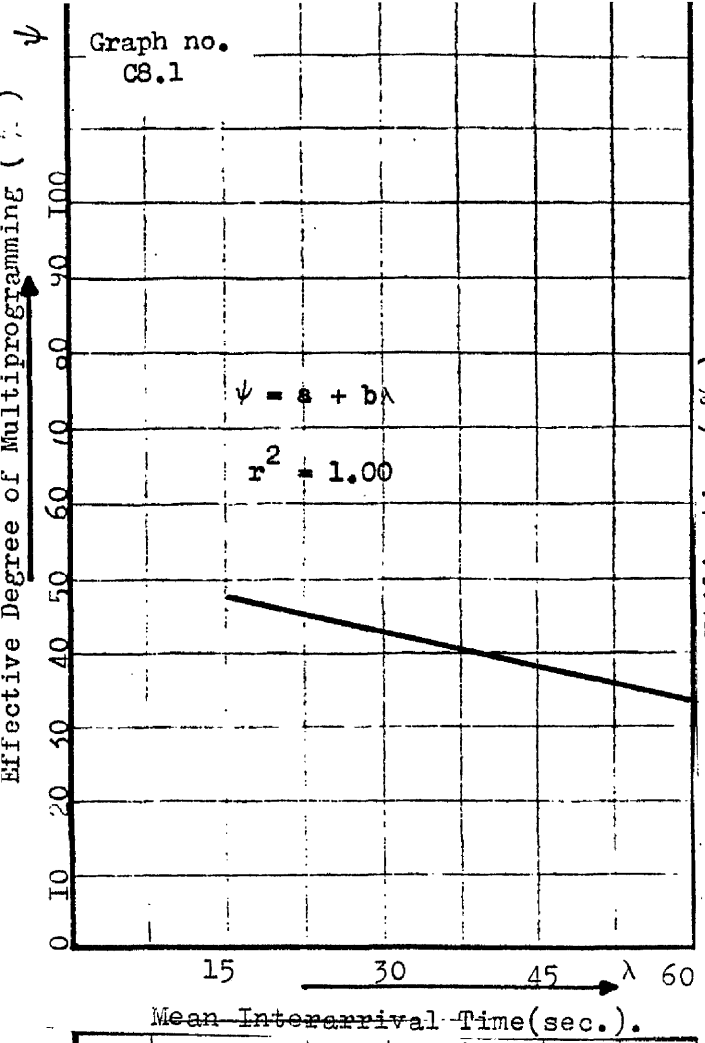
* Increasing the parameter, average think time, shows no major effects on the new performance indices, except it decreases the no. of jobs processed (see graph C7.6). But this parameter, as we have seen in case 3, causes considerable changes to the average response time and the interactive throughput (see graphs C3.4 and C3.6). Hence, we may consider that the users behaviour can affect the system performance, since the above parameter ($\overline{TH}$) is a user-oriented factor. Although, fixing the average think time in the GST to 15 secs. is a very reasonable decision for the interactive system which supports a number of terminals. The reasons for considering it so are:
  1. Customers requests require considerable use of resources and this makes response time long.
  2. Customers can stack requests. That is, while a customer is waiting for the system to respond to one request, he can make additional requests.
  3. Many computer systems support graphic terminals, in which customers interact with these terminals, mainly by means of lightpen, which is used to pick items on a menu and to pick lines on the drawings. This can be done very quickly, and so tends to keep user think time low.
* The effects of the parameter, average think time on the new selected performance indices are modelled. The models can be used to predict the future changes.

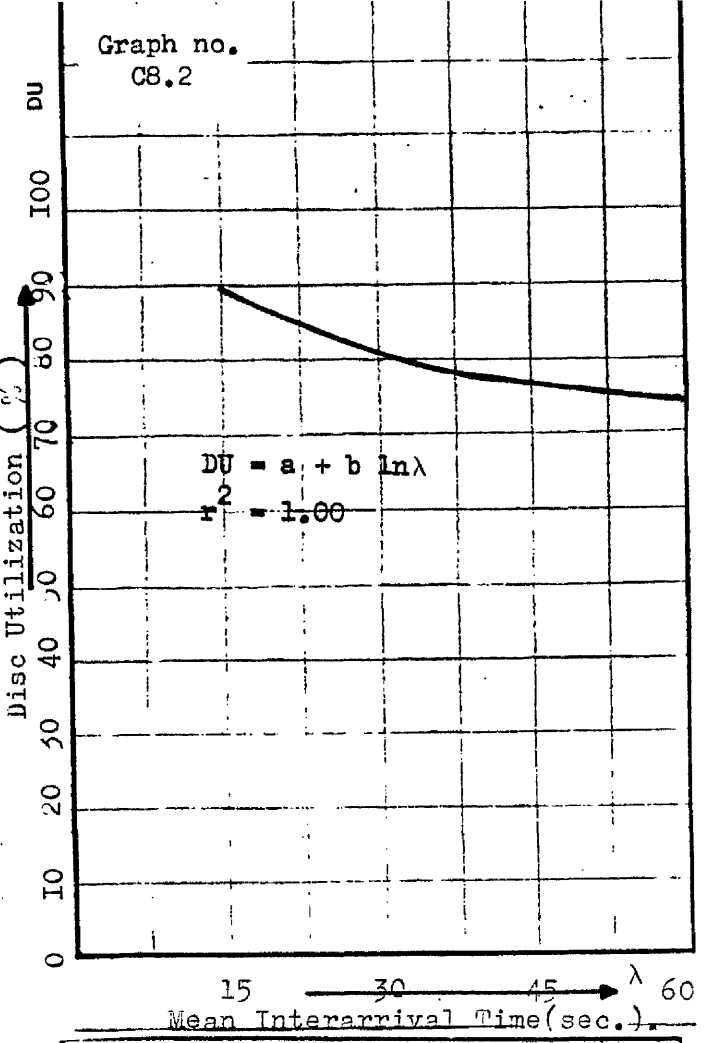
3.3.8. Case 8: Other effects of the performance parameter: mean interarrival time.

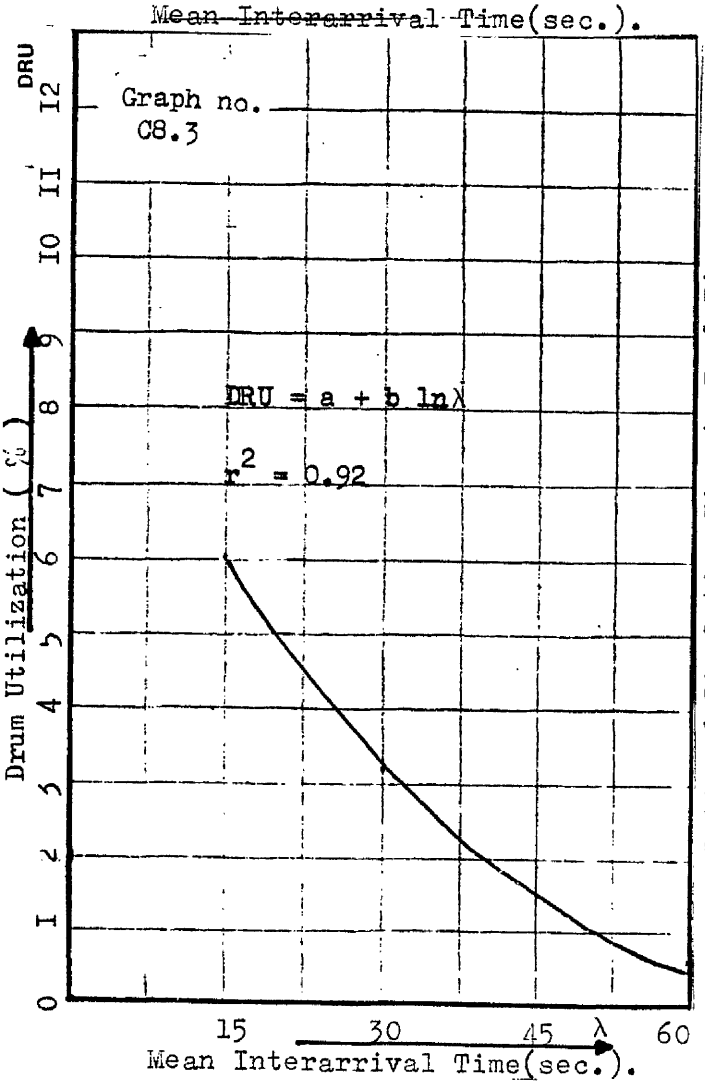
This case study addresses the effects of varying the mean interarrival time on the new selected performance indices (i.e. Ψ, DU, DRU, TCT, MJP and RSR). This can be achieved by running the GST with a workload of mean interarrival time from 15 to 60 secs. in steps of 15. Refer to table C8.1 and graphs C8.1, C8.2, C8.3, C8.4, C8.5 and C8.6.

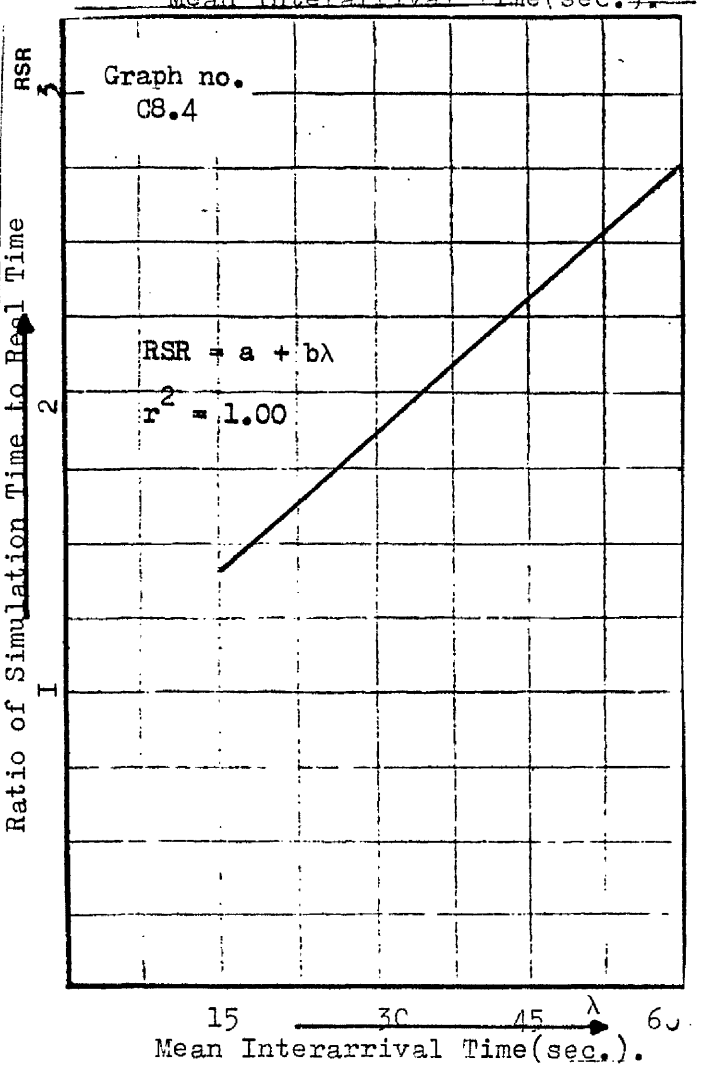
Table No. C8.1

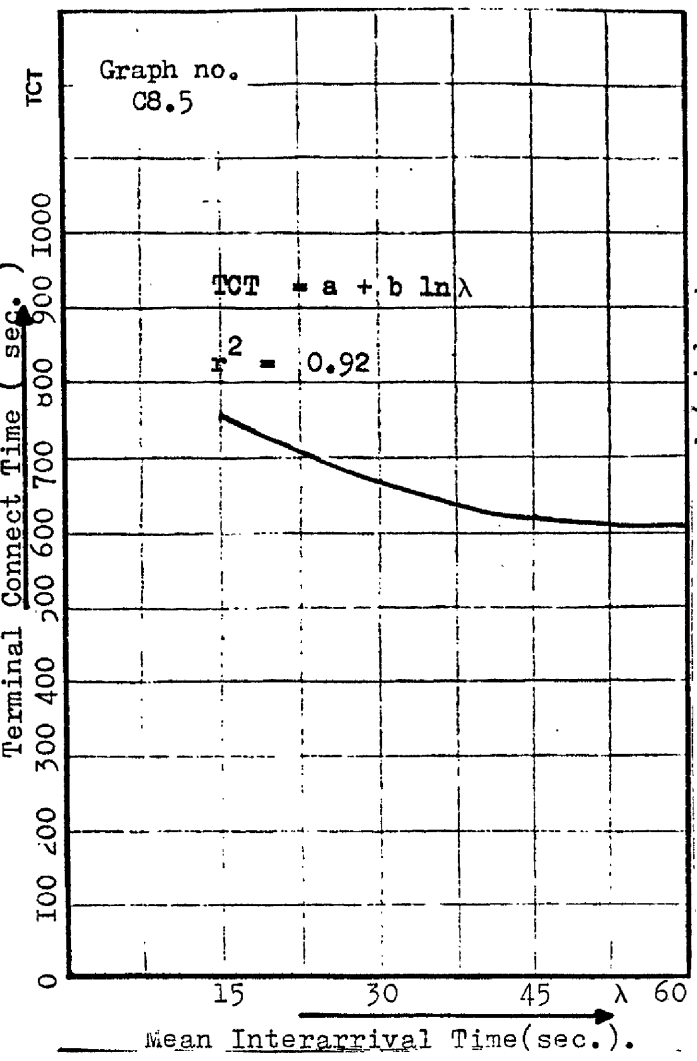
Performance Index	Simulation Time (Mins)	The Performance Parameter: Mean Interarrival Time (λ)									
		15 sec		30 sec		45 sec		60 sec.			
Degree of Multi-programming (%)	5	21.26		11.81		7.93		5.93			
	10	36.97		22.19		15.03		11.85			
	15	43.15		32.10		22.44		16.99			
	20	45.81		37.64		29.32		22.40			
	25	47.15		40.72		34.33		27.47			
	30	48.39		42.56		37.08		31.64			
	35	48.81		44.10		38.91		34.18			
Device Utilization (%)		Disc	Drum	Disc	Drum	Disc	Drum	Disc	Drum	Disc	Drum
	5	86.27	4.45	68.17	0.00	60.86	0.00	53.33	0.00		
	10	88.52	6.50	71.03	0.54	59.94	0.02	44.84	0.00		
	15	89.38	7.50	75.78	1.47	64.50	0.38	54.59	0.04		
	20	89.47	7.61	78.33	1.79	70.44	0.76	62.56	0.26		
	25	89.72	7.04	79.88	1.75	74.70	1.05	68.04	0.48		
	30	89.83	6.89	81.39	1.70	76.67	1.08	71.73	0.68		
	35	89.94	6.59	81.70	2.25	77.88	1.42	73.73	0.93		
Terminal Connect Time (Secs per user)	5	200		225		218		233			
	10	244		253		233		230			
	15	397		247		252		268			
	20	516		400		344		310			
	25	570		489		428		373			
	30	622		599		505		424			
	35	748		705		614		607			
Number of Multiaccess jobs processed (jobs)	5	1		2		1		1			
	10	5		3		2		3			
	15	9		7		5		4			
	20	15		12		10		9			
	25	23		16		14		11			
	30	26		26		21		16			
	35	31		29		28		27			
Ratio of Simulation Time to Real Time	5	1.68		2.81		3.25		3.74			
	10	1.43		2.66		3.65		4.18			
	15	1.35		2.14		2.99		3.87			
	20	1.35		2.00		2.50		3.06			
	25	1.37		1.96		2.29		2.71			
	30	1.39		1.94		2.19		2.50			
	35	1.42		1.89		2.14		2.36			

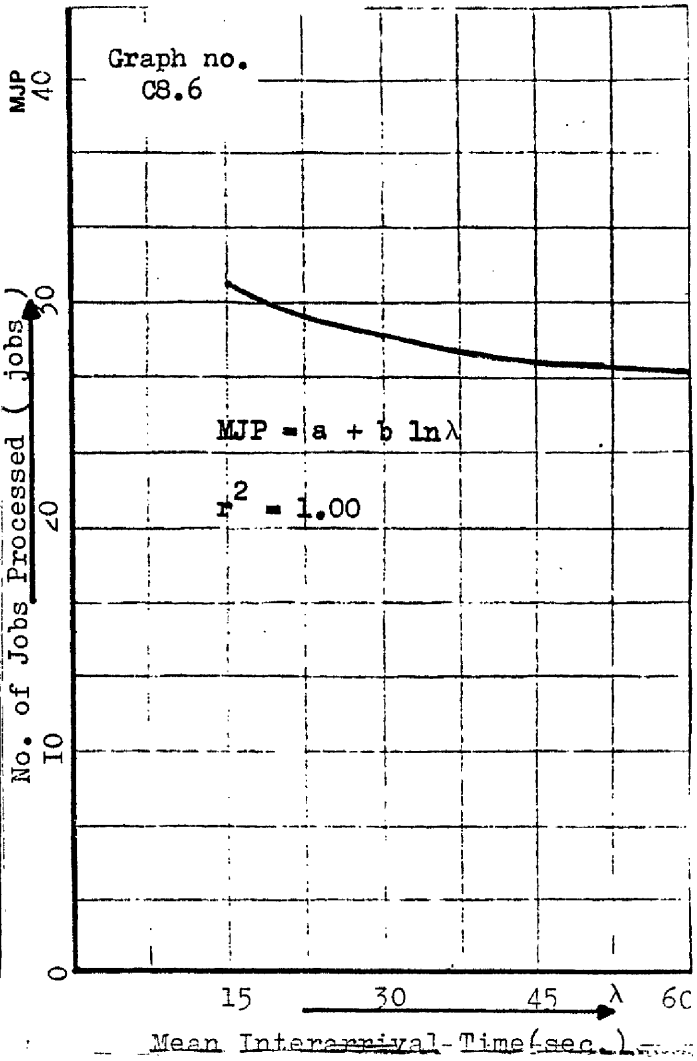

Graph no.
C8.1
Effective Degree of Multiprogramming (ψ)
ψ = a + bλ
r² = 1.00
0 10 20 30 40 50 60 70 80 90 100
15 30 45 λ 60
Mean Interarrival Time(sec.).


Graph no.
C8.2
DU
Disc Utilization (%)
DU = a + b lnλ
r² = 1.00
0 10 20 30 40 50 60 70 80 90 100
15 30 45 λ 60
Mean Interarrival Time(sec.).


Graph no.
C8.3
DRU
Drum Utilization (%)
DRU = a + b lnλ
r² = 0.92
0 1 2 3 4 5 6 7 8 9 10 11 12
15 30 45 λ 60
Mean Interarrival Time(sec.).


Graph no.
C8.4
RSR
Ratio of Simulation Time to Real Time
RSR = a + bλ
r² = 1.00
1 2 3
15 30 45 λ 60
Mean Interarrival Time(sec.).





The relationships between the parameter, mean interarrival time and the new performance indices can be constructed from table C8.1 using regression analysis and under heavy loaded system condition. These relationships represent simple hybrid models which can be expressed using the relational graphs (from C8.1 to C8.6) and table C8.2.

PERFORMANCE INDEX	RELATIONSHIP EQUATION	REGRESSION CONSTANT VALUES	GRAPH NO.	EQUATION NUMBER
DEGREE OF MULTIPROGRA-	$\Psi = a + b\lambda$	$a = 53.77$ $b = -0.33$	C8.1	E4,4
DISC UTILIZATION	$DU = a + b \ln\lambda$	$a = 120.95$ $b = -11.49$	C8.2	E4,5
DRUM UTILIZATION	$DRU = a + b \ln\lambda$	$a = 17.28$ $b = -4.13$	C8.3	E4,6
TERMINAL CONNECT TIME	$TCT = a + b \ln\lambda$	$a = 1055.62$ $b = -110.52$	C8.4	E4,7
No. OF JOBS PROCESSED	$MJP = a + b \ln\lambda$	$a = 38.69$ $b = -2.84$	C8.5	E4,8
RATIO OF SIMUL TIME TO REAL	$RSR = a + b\lambda$	$a = -0.40$ $b = 0.67$	C8.6	E4,9

TABLE No. C8-2

Conclusions:

* The mean interarrival time (for a poisson input distribution stream) is an important factor of a specified workload in the GST. The effects of increasing it give us the following situations:
    * It decreases the effective degree of multiprogramming.
    * It decreases the disc and drum utilization.
    * It reduces the ratio of simulation time to real time, and therefore the cost.
    * It reduces the multiaccess jobs in the system.
    * It reduces the terminal connect time.

  The increase of this parameter also has the following effects (see Case 4):
    * It decreases the response time.
    * It decreases the interactive throughput.

  For the purpose of our experiments, we have selected a reasonable value of the mean interarrival time equal to 30 secs.
* The effects of the parameter, mean interarrival time on the new selected performance indices are modelled. The models can be used to predict the future changes.

3.4. Aim of Experimentations:

Through the previous case studies, we have built several simulation/regression analysis models for many performance indices with different given input parameters. These models specify the behaviour of GST by changing a single-input parameter.

The prime aim of these experiments is to construct a general model to represent the whole performance behaviour of the computer system which has been simulated using the GST.

Several researchers have tried to construct general models representing the behaviour of a system, but most of them found this task very difficult. Hence, most of the models existing in the literature are of components or subsystems of a computer system. In particular, a large number of models exist for:

* Memory management ((/Denning 70/)).
* I/O ((/Koffman 69/)).
* CPU scheduling algorithms....etc. ((/Kleinrock 64/)).

Some of the existing models are not really reliable. For example, consider the 'throughput model' built by Gaver ((/Gaver 67/)), for which Fenichel and Grossman ((/Fenichel and Grossman 69/)) claim that "it is greatly to Gaver's credit that this work was published and we consider his results are strictly negative". Different researchers propose different methods in order to solve the problem of constructing a general model of a computer system. These include:

* Parameter identification method: ((/Kimbleton 72/))
  ((/Bose and Warn 75/))
  In this method only a small number of performance parameters and indices are identified. The identification process rule states "The selection of parameters depends upon which of them strictly affects the computer system behaviour".
* Hierarchical method: ((/Sekino 72/))
  In this method, the identification of parameters and indices breaks into several modules. These modules are arranged hierarchically.

The general process of our method is given by figure 3.7. This process can be performed using on the parameter identification method.

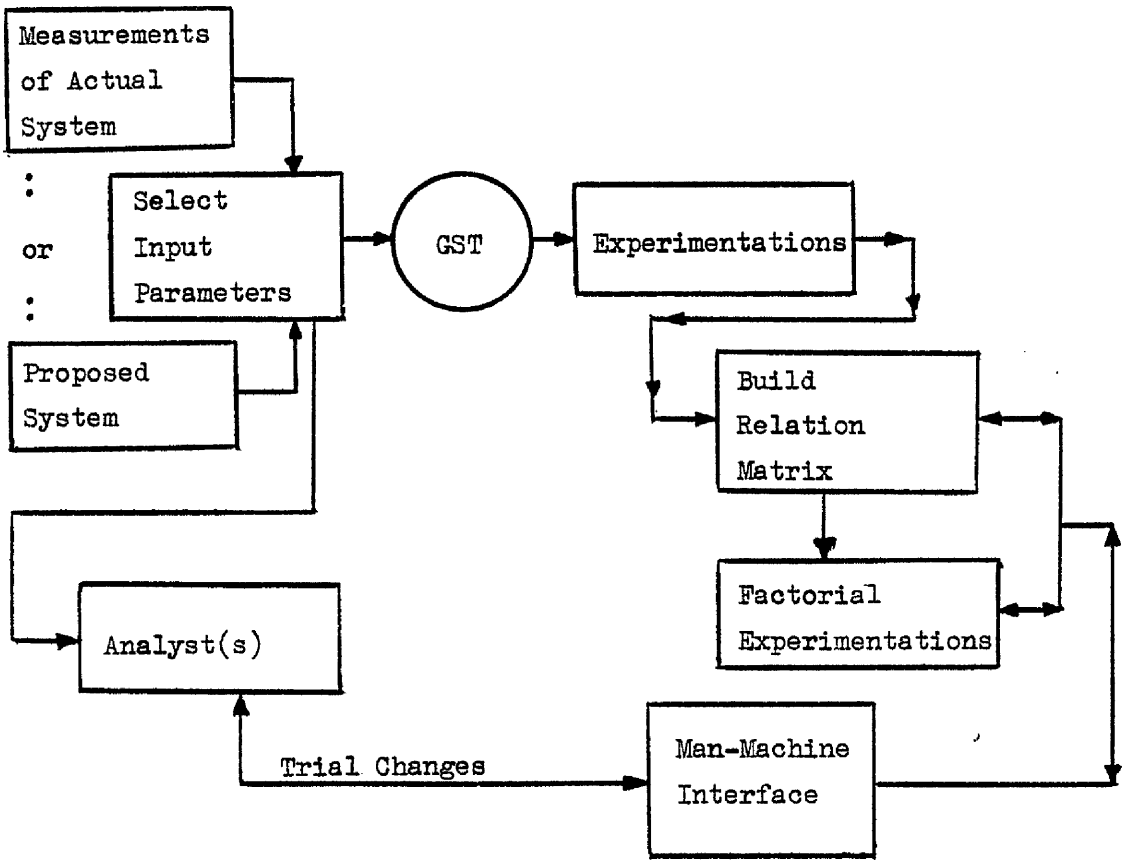


Figure 3.7 : Schematic Diagram of the Experimentations Aim.

The steps for building our general model can be summarized as follows:

1. Perform all case studies.
2. Build the Relation Matrix (i.e. model all the single parameter changes)
3. Perform factorial experiments (or case studies) on the given Matrix (i.e. Model all the multiple parameter changes).

The general model may contain a further extension (or step), in which the analyst can study the behaviour of the system in an interactive way. This extension can be done through a very simple program (i.e. Man-Machine Interface) using the relation matrix and the factorial experimentation rules. In this case, we may call the general model an interactive design tool (IDT). The first step can be done by performing the same method as the previous case studies for all selected parameters. The step of building a relation matrix follows

the experimentation step (i.e. performing all the case studies). It organizes the access to any hybrid model required. The hybrid models are a result of several case studies. The relation matrix has the following shape (see figure 3.8).

Performance Index (I) →

Parameter \ Index	$\bar{R}$	$\overline{PBT}$	$\bar{X}$	ψ	DU	DRU	TCT	MJP	RSR	PDT	WPT	WST	WTT	WDT	WDRT
M	E1,1	E1,2	E1,3	E1,4	E1,5	E1,6	E1,7	E1,8	E1,9	E1,10	E1,11	E1,12	E1,13	E1,14	E1,15
$\overline{TSM}$	E2,1	E2,2	E2,3	E2,4	E2,5	E2,6	E2,7	E2,8	E2,9	E2,10	E2,11	E2,12	E2,13	E2,14	E2,15
$\overline{TH}$	E3,1	E3,2	E3,3	E3,4	E3,5	E3,6	E3,7	E3,8	E3,9	E3,10	E3,11	E3,12	E3,13	E3,14	E3,15
λ	E4,1	E4,2	E4,3	E4,4	E4,5	E4,6	E4,7	E4,8	E4,9	E4,10	E4,11	E4,12	E4,13	E4,14	E4,15
MPT	E5,1	E5,2	E5,3	E5,4	E5,5	E5,6	E5,7	E5,8	E5,9	E5,10	E5,11	E5,12	E5,13	E5,14	E5,15
MSI	E6,1	E6,2	E6,3	E6,4	E6,5	E6,6	E6,7	E6,8	E6,9	E6,10	E6,11	E6,12	E6,13	E6,14	E6,15
MZ	E7,1	E7,2	E7,3	E7,4	E7,5	E7,6	E7,7	E7,8	E7,9	E7,10	E7,11	E7,12	E7,13	E7,14	E7,15
SPZ	E8,1	E8,2	E8,3	E8,4	E8,5	E8,6	E8,7	E8,8	E8,9	E8,10	E8,11	E8,12	E8,13	E8,14	E8,15
MNI	E9,1	E9,2	E9,3	E9,4	E9,5	E9,6	E9,7	E9,8	E9,9	E9,10	E9,11	E9,12	E9,13	E9,14	E9,15
PWS	E10,1	E10,2	E10,3	E10,4	E10,5	E10,6	E10,7	E10,8	E10,9	E10,10	E10,11	E10,12	E10,13	E10,14	E10,15
MSL	E11,1	E11,2	E11,3	E11,4	E11,5	E11,6	E11,7	E11,8	E11,9	E11,10	E11,11	E11,12	E11,13	E11,14	E11,15
FPZ	E12,1	E12,2	E12,3	E12,4	E12,5	E12,6	E12,7	E12,8	E12,9	E12,10	E12,11	E12,12	E12,13	E12,14	E12,15
ST	E13,1	E13,2	E13,3	E13,4	E13,5	E13,6	E13,7	E13,8	E13,9	E13,10	E13,11	E13,12	E13,13	E13,14	E13,15
SDF	E14,1	E14,2	E14,3	E14,4	E14,5	E14,6	E14,7	E14,8	E14,9	E14,10	E14,11	E14,12	E14,13	E14,14	E14,15
PBZ	E15,1	E15,2	E15,3	E15,4	E15,5	E15,6	E15,7	E15,8	E15,9	E15,10	E15,11	E15,12	E15,13	E15,14	E15,15
MBR	E16,1	E16,2	E16,3	E16,4	E16,5	E16,6	E16,7	E16,8	E16,9	E16,10	E16,11	E16,12	E16,13	E16,14	E16,15
MDR	E17,1	E17,2	E17,3	E17,4	E17,5	E17,6	E17,7	E17,8	E17,9	E17,10	E17,11	E17,12	E17,13	E17,14	E17,15
CST	E18,1	E18,2	E18,3	E18,4	E18,5	E18,6	E18,7	E18,8	E18,9	E18,10	E18,11	E18,12	E18,13	E18,14	E18,15
PIT	E19,1	E19,2	E19,3	E19,4	E19,5	E19,6	E19,7	E19,8	E19,9	E19,10	E19,11	E19,12	E19,13	E19,14	E19,15
PCT	E20,1	E20,2	E20,3	E20,4	E20,5	E20,6	E20,7	E20,8	E20,9	E20,10	E20,11	E20,12	E20,13	E20,14	E20,15

Performance Parameter (J) ↓

Figure 3.8*: The Relation Matrix R(I, J).

* The abbreviation used in the relation matrix can be found in abbreviation appendix A.2.

The third step will be left as future research work. Finally, the general model or the interactive design tool is a direct result of the simulation/regression tool, and therefore it has the same advantages (see section 3.3).

For more information about the GST, the reader is referred to the enclosed simulation program of the GST. The program has has been written using the programming language C ((/Kernighan and Ritchie 78/)) which may be considered as a better language to implement the GST, than the original language (a subset of PL/1), especially under our host computer system VAX 11/780. The new implementation of the GST has been achieved by Cavouras ((also see section 6.2/Cavouras 78/)). More information of how we can construct a similar simulator, the reader is referred to ((/Lindstrom and Skansholm 81/)).

CHAPTER 4

THE OPERATIONAL ANALYSIS APPROACH

4.1. Introduction:

Operational analysis represents a new approach to the problem of analysing system performance during time periods of interest ((/Buzen 77/)). This approach has been developed by Buzen ((/Buzen 76/)) and extended by Denning and Buzen ((/Denning and Buzen 77/)) to apply to queueing networks, especially in the context of the study of the performance of multiple-resource computer systems. This approach is quite attractive for the reasons mentioned in the first chapter. Although operational analysis remains a recent approach to performance evaluation ((/Sevcik and Klawe 79/)), many researchers are trying to extend this subject and develop it in order to build an ideal design and evaluation tool ((/Bouhana 78/)) ((/Roode 79/)) ((/Hofri 79/)) ((/Bryant 79/)) ((/Denning and Denning 79/)). These researchers have tried to put some missing links to the available operational analysis by comparing it to the traditional approach (i.e. Stochastic modelling). This is quite important as a first step, since the results of the operational analysis can be validated easily using the traditional approach. But, we believe that the research work in this subject should move a step further by adding new and powerful facilities to this type of analysis. The extensions or additions may consist of the following levels:

1. The representation level:

   At this level, several software components should be represented. Examples of these components will be the operating system modules, by which we can study the:

   * efficiency of memory management.
   * effects of job scheduling.
   * effects of CPU scheduling.
   * effects of resource and queue management.

2. The mathematical level:

   At this level, new mathematical structures should be added to the operational analysis. This step was started by Bouhana ((/Bouhana 78/)) in which he added the matrix algebra to the operational analysis body. The new mathematical structures will increase the efficiency of this type of analysis.

In this chapter, we will try to use operational analysis to represent the same system as the GST (see chapter 3). The research will be concentrated on interactive computer systems and closed queueing networks. For this purpose, queueing theory has been used to

represent structured operational models. A structured model is a description of the actual system components and their connections (structural models are most frequently represented by block diagrams. The level of detail in a block diagram can easily be varied, since individual blocks can, in turn, be further laid down as self-contained block diagrams) ((/Svobodova 76/)).
The operational models will be introduced as follows:

* Single-Resource Queueing System.
* Queueing-Networks.

4.2. Single-Resource Queueing System:

4.2.1. Background:

In queueing theory, the term queue is a synonym for the waiting line that forms in front of a service facility or server. The entities in a queue are generally called customers (jobs, tasks or any logical entities that can conceptually generate a request for service). A single-resource queueing system is often called an isolated queue. An isolated queue consists of the following attributes ((/Bouhana 78/)):

* Arrival process:

  The arrival process describes the protocol according to which customers arrive at a queue with their requests for service.

* Scheduling discipline:

  The scheduling discipline describes the protocol according to which customers receive service. An example, of common scheduling discipline is:

  * First-Come-First-Served (FCFS) and
  * Processor Sharing (PS)
  * Last-Come-First-Served (LCFS)
  * Round-Robin (RR).

* Service-Time Distribution:

  After a customer has progressed through a queue, the time that he is in service varies according to his need for service. The distribution of time that a server allocates to a customer in a single visit is called the service time distribution.

* Departure Process:

  The departure process is similar in concept to the arrival process, except that it cannot be arbitrarily specified.

An additional aspect of queues is the multiplicity of the server. If there is only one server present in a queue, then the queue

is said to have a simple server. If however, a queue leads into a service facility that has more than one server, then the queue is said to have multiple servers. Figure 4.1. shows a typical simple server isolated queue with its attributes. Also, we may call this type of queue a non-pre-emptive single-resource queueing system.

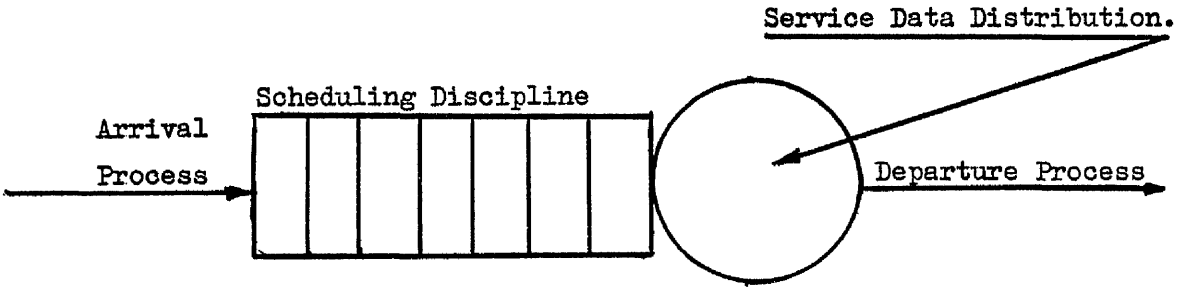


Figure 4.1: Simple Server Isolated Queue.

4.2.2. Single Resource Queue:

A single resource queueing system, with one queue and a server, is observed for an interval (0,T) . Figure 4.1. shows this type of queue. The behaviour of such queues were studied using operational analysis techniques by ((/Denning and Buzen 78/)) ((/Buzen and Denning 80/)).

To show how operational analysis can be used to construct a performance model of the'non-pre-emptive' single resource queueing system, let us define its basic operational quantities:

n = number of jobs present in the system at time t, $0 \leqslant t < T$.

T = length of the observation period.

A = number of arrivals.

B = total busy time (time during $n>0$ and $B \leqslant T$).

C = number of completions occuring during the observation period.

W = area under the graph n(t) during the observation period.

In terms of these basic quantities the following derived operational quantities are defined:

λ = A/T, the arrival rate (customers/second).

X = C/T, the output rate (customers/second).

U = B/T, the utilization(fraction of time system is busy).

R = W/C, the average of time accumulated in the queue per completed customer.

S = B/C, the mean service time per completed customer.

$\overline{n}$ = W/T, the mean queue length of the queue.

The basic quantities (A,B,C,W) are typical of "raw data"

collected during an observation, and the derived quantities (λ,X,U,R, S,$\overline{n}$) are typical of "performance measures". All these quantities are variables which may change from one observation period to another. But, to construct the relations that must hold in every observation period, regardless of the values observed, we need to derive new equations. These equations are called operational laws.
Now, the following equations represent some of the operational laws:

* Utilization law:

Since

$$X = \frac{C}{T} \quad , \quad \Longrightarrow C = XT$$

and $$S = \frac{B}{C} \quad , \quad \Longrightarrow B = SC$$

hence $$B = SXT$$

but since $$U = \frac{B}{T}$$

then $$U = \frac{SXT}{T} = SX$$

i.e. $\boxed{U = SX}$ is the utilisation law.

* Little's law:

Since

$$\overline{n} = W/T \qquad W = \overline{n}T$$

and $$X = C/T \qquad C = XT$$

but $$R = W/C = \frac{\overline{n}T}{XT} = \frac{\overline{n}}{X}$$

i.e. $\overline{n} = RX$ is Little's law.

Using the above operational laws and operational quantities, with specific testable assumptions we can construct many operational theorems. As an example, if we assume that the number of arrivals is equal to the number of completions during the observation period. This assumption is called job flow balance, that is, if we assume:

$$A = C$$

then, we can construct the Utilization Theorem as follows:

since $A = C$

, $A = \lambda T$

and $C = XT$ then $\lambda = X$

and since $U = SX$ then

$\boxed{U = S\lambda}$ is the Utilization Theorem.

4.2.2.1. Further Notes:

An isolated queue is not the only type of single-resource queueing system. Examples of single-resource queueing systems are given in figures 4.2., 4.3. and 4.4.

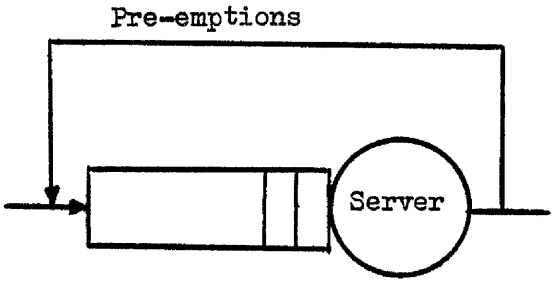


Figure 4.2: Pre-emptive Single-Resource Queueing System.

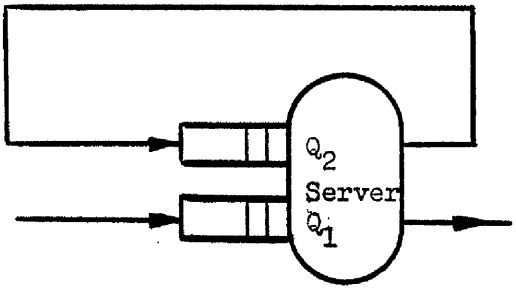


Figure 4.3: Two-level Foreground-Background Single-Resource Queueing System. (FB)

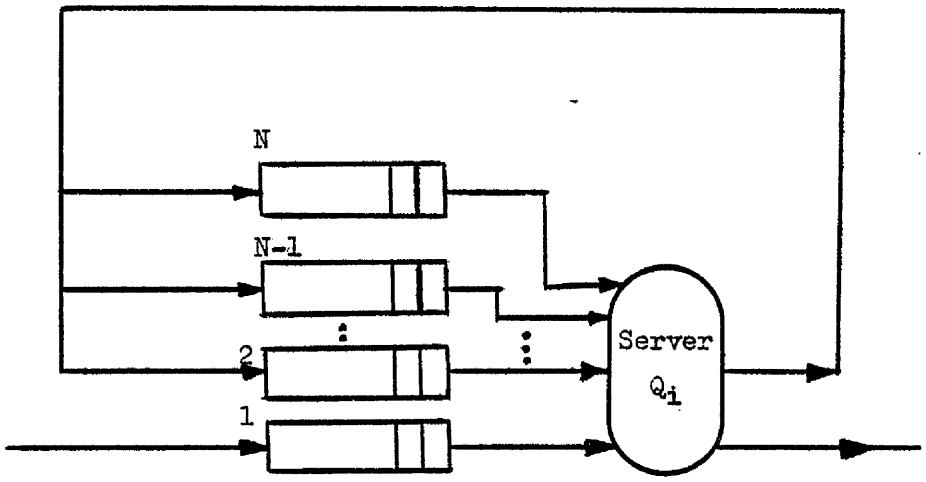


Figure 4.4: N-Level Foreground-Background Single Resource Queueing System. (FB_N)

These types of single-resource queueing systems have been intensively studied using the traditional approach (i.e. Stochastic Modelling) by many researchers ((/Takacs 63/)) ((/Estrin and Kleinrock

67/)) ((/McKinney 69/)) ((/Coffman and Denning 73/)) ((/Pujolle and Soula 79/)). But none of these researchers tried to represent them using operational analysis. The analysis of the above systems is out of the scope of this thesis.

4.3. Queueing Network System:

4.3.1. Background:

So far, we have studied a single resource queueing system which may, in fact, represent a single input/output device or central processing unit within a computer system. A model of the entire computer system can be developed by connecting single-resource queueing systems in the same way as connecting the devices of an actual computer system configuration.

Queueing networks have become a widely used analytic tool for multiple resource computer system performance studies ((/Denning and Buzen 77/)). For several years, queueing theory has been developed ((/Jackson 57, 63/)) ((/Gordon and Newell 67/)) ((/Baskett, Chandy, Muntz and Palacois 75/)) mainly depending on stochastic modelling techniques. The theoretical approach, however, has proved to be very difficult to use in practice, because many of its assumptions such as equilibrium and stationary conditions, cannot be proved to hold by observing the system in a finite time period. Hence, a new research approach which is called operational queueing network was introduced ((/Denning and Buzen 78/)). The operational approach leads to the same mathematical equations as the traditional approach (i.e. stochastic modelling). These equations can be derived in a very simple way depending on testable assumptions.

Queueing networks can be classified according to the following factors: ((/Kienzle and Sevcik 79/))

1. Model Structure:

   Describes the number of service centres and the manner in which customers flow among them. We distinguish the following:

   * Single server model (possibly with feedback loop).
   * Cyclic queueing model.
   * Central server model.
   * A centralized model.
   * General queueing network.
   * Hierarchical queueing network.

2. The Arrival Process:

   This process indicates the manner in which new customers come

into existance. A model is classified as:

* Closed (fixed number of customers in each routing chain).
* Open (arrivals and departures in all routing chains).
* Mixed (some routing chains are open and some are closed).

3. The Workload Classes:

This indicates groupings of customers that are statistically indistinguishable. Possibilities are:

* single class model.
* multiple class model.
* multiple class model with class changes.

4. The Queueing Disciplines:

We can distinguish the following:

* Station balanced disciplines (these include processor sharing (PS), pre-emptive last come first served (PLCFS) and no queueing).
* Class independent work conserving discipline (this includes FCFS).
* Strict priority disciplines (these are based on computer classes).
* General disciplines (these are mixed disciplines of the above strategies).

5. The Service Demand Description:

Can be specified as either,

* A workload vector (the mean total service required by the customer of a class at each device is stated) or,
* A routing matrix (indicates the movements of customers and the distribution of service times for each class at each device).

6. The Server Characteristics:

* Load-independent servers, and
* Load-dependent servers.

Finally, queueing networks models may have one of the following characteristics((/Chandy and Sauer 78/)):

* Tractable Solution:

  Those models which can be analyzed to give exact (as opposed to approximate) solutions in an'adequately' short time.

* Intolerably Slow Solution:

  Those models which cannot be analyzed in an 'adequately' short time to give exact solutions.

* Unsolved:

  Those models for which there is no known method of analysis guarranteed to give exact results.

The vast majority of queueing network models used for estimating and predicting computer system performance are of the tractable category. Many researchers have, however, tried to use the second and third categories with one of the following:

* Using approximate solution techniques.
* Using a simple, tractable model to obtain bounds for the performance measures of a more complex model.
* Using simulation tools specifically designed for the solution of complex queueing models.

For the purposes of this chapter, we will now try to introduce tractable operational tool which can analyse the behaviour of the closed queueing networks, in order to model the interactive computer systems.

4.3.2. The Operational Assumptions:

The following assumptions should be considered when an operational queueing network model is under construction:

1. The network should be operationally connected. This means that a customer must eventually be able to travel from any server to any other server in the network.
2. Each server has a finite mean service time.
3. A customer cannot be either enqueued or in a service at more than one server simultaneously (apart from this assumption, there is no CPU-I/O overlapping).
4. No customer waits in front of an idle server.
5. No blocking (i.e. no part of the system can block progress in another part).
6. A customer incurs no delay in travelling between servers.
7. The rules governing the routing of customers through the network do not change with the passage of time.
8. Servers do not interfere with each other in the sense that the mean service time of a server does not depend on the number of customers enqueued or in service at any other server. This is called the homogeneity assumption.

Some other assumptions will be introduced and defined in the next section. These assumptions are:

* Classes of customers.
* Job flow balance.
* State transition balance.
* One step behaviour.
* Load-independent or load-dependent assumptions.

These are the only assumptions needed to construct any operational queueing network model. These are all testable assumptions.

4.3.3. Simple Closed Queueing Network Operational Analysis:

A closed queueing network is one in which a fixed number of customers travel among the servers. One way of allowing customers to 'arrive' and 'depart' in a closed network is to designate a single existing server as the network's conceptual entry and exit portal. A loop is placed on the designated server. Whenever a customer traverses the loop, he conceptually exits the network, changes identity, and re-enters the network as a new customer. Such a scheme models the real-world situation in which there is a continual backlog of jobs waiting to enter a computer system.

A simple example of a closed queueing network is the cyclic model, shown in figure 4.5. ((/Chandy and Sauer 78/)).

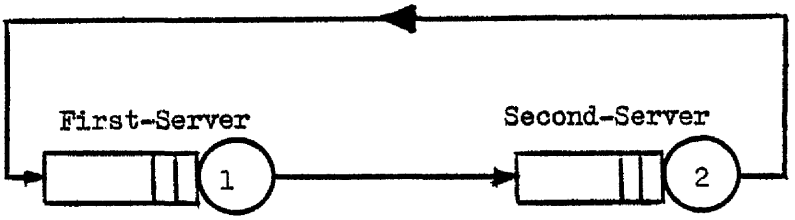


Figure 4.5: The Cyclic Queueing Network Model (CQNM).

Customers arrive at the first of two (or more) queues, and after completing service from the last queue, they may re-start in the first queue and so on. Not all queueing networks are as simple as the one above; they can become arbitrarily complicated as the number of servers and the paths increase. However, the analysis of this type of queueing network will give a good introduction to the analysis of a complicated closed queueing network which is required to model the interactive computer systems. Another example of a closed queueing network is the Central Server Model (CSM) ((/Buzen 71/)).

4.3.3.1. The Operational Aspects of the Simple Closed Queueing Network:

Suppose that a closed queueing network such as CQNM or CSM is measured during an observation period of length T seconds and that the following data are collected for each device i:

T : Observation time, where $0 \leqslant t \leqslant T$.

$n_i(t)$: The operational state of the device i at time t, where

$0 \leqslant n_i \leqslant N$ (*an operational state of a device i is the number

of customers either enqueued or in service at a device i and at time t*).

(* if we consider there are M-servers in the closed queueing network then:

$\sum_{i=1}^{M} n_i(t) = N$ represent the total number of customers in the queueing network*).

$A_i(n)$: Number of arrivals to the ith device that find $n_i(t) = n$, where $0 \leq n < N$.

$C_{ij}(n)$: Number of times a customer requests service at device j immediately after completing a service request at device i, where $n_i(t) = n$, $0 < n \leq N$.

$T_i(n)$: Total time during which $n_i(t) = n$, $0 \leq n \leq N$.

We may also treat the 'outside world' as a device "0", in which case we can define:

$A_{oj}(n)$: number of customers whose first service request is for device j when $n_i(t) = n$, $0 \leq n < N$.

$C_{io}(n)$: number of customers whose last service request is for device i.

We may assume also, that $C_{oo}(n) = 0$ and it is possible that $C_{ii}(n) \neq 0$ for any device i since a customer could request another burst of service from a device which had just completed a request from that customer.

The following grand totals are defined:

$C_i(n) = \sum_{j=1}^{m} C_{ij}(n)$, total number of request completions at device i, when $n_i(t) = n, 0 < n \leq N$.

$C_i = \sum_{n=1}^{N} C_i(n)$, total number of request completions at device i.

$A_i = \sum_{n=0}^{N-1} A(n)$, total number of arrivals at device i.

$T_i = \sum_{n=0}^{N} T_i(n)$, the busy time.

Given these basic quantities, the following derived operational quantities are defined:

$S_i(n) = \frac{T_i(n)}{C_i(n)}$, mean service time at device i when $n_i(t) = n$. (defined only if $C_i(n) > 0$).

$Y_i(n) = \frac{A_i(n)}{T_i(n)}$, arrival rate at device i when $n_i(t) = n$, (defined only if $T_i(n) > 0$).

$B_i = T_i(1) + T_i(2) + \ldots + T_i(N)$, total busy time of a device i.

$S_i = \frac{B_i}{C_i}$, mean service time at device i.

$U_i = \frac{B_i}{T_i}$, utilization of device i.

$Y_i^o = \frac{A_i}{T_i}$, overall arrival rate at device i.

$Y_i = A_i(T_i - T_i(N))$, restricted arrival rate at device i.
(defined only if $T_i(n) < T_i$).

$X_i = \frac{C_i}{T_i}$, output rate of device i.
(*i.e. $X_o = \frac{C_{io}}{T_i}$ output rate from the system *)

$W_i = \sum_{n=1}^{N^i} n\, T_i(n)$, job-seconds of accumulated waiting time at device i.

$Q_i = W_i/T_i$, mean queue length of device i.

$R_i = W_i/C_i$, mean response time per completed job.

$P_i(n) = \frac{T_i(n)}{T_i}$, device i queue total distribution where $n = 0,\ldots,N$.

$P_{A_i}(n) = \frac{A_i(n)}{A_i}$, device i queue arrival distribution where $n = 0,\ldots,N-1$.

$P_{C_i}(n) = \frac{C_{i(n+1)}}{C_i}$, device i queue completions distribution where $n = 0,\ldots,N-1$.

(* $C_i(n+1)$ is used to define P_{C_i} because $P_C(n)$, refers to the queue size just after a completion whereas $C(n)$ refers to the queue size just before a completion *)

$q_{ij} = (\frac{1}{C_i}) \sum_{n=1}^{N} C_{ij}(n)$, routing frequency (*specifies the fraction of request completions at device i which are followed immediately by requests for device j*)

Using the above operational quantities we can construct the following operational laws:

* $U_i = S_i X_i$ (* Utilization law *)

* $R_i = Q_i/X_i$ (* Little's law *)

* $R_i = \sum_{n=1}^{N} n\, S_i(n)\, P_{C_i}(n-1)$ (* response time law *)

* $X_o = \sum_{i=1}^{M} X_i\, q_{io}$ (* output flow law *)

* $S_i = \sum_{n=1}^{N} P_{C_i}(n-1)\, S_i(n)$ (* for defined $S_i(n)$ *)

* $X_i = \sum_{n=1}^{N} P_i(n) / S_i(n)$ (for defined $S_i(n)$)

* $Y_i^o = \sum_{n=0}^{N-1} P_i(n)\ Y_i(n)$ (for defined $Y_i(n)$)

* $Y_i / Y_i^o = 1 / (1 - P(N))$ (if $T_i(N) < T_i$)

* $P_{A_i}(n) = P_i(n)\ (\ Y_i(n) / Y_i^o\ )$ (if $Y_i(n)$ defined)

Now, using the above quantities and laws we can construct many operational theorems by imposing some additional simplifying assumptions on the system. These assumptions will yield a tractable operational analysis. These assumptions are:

1. Job Flow Balance:

For each device i, the overall output rate X_i is equal to the overall input rate Y_i^o to device i. When a system conforms to this assumption, the quantities X_i are called device throughputs. This is equivalent to assuming that the total number of arrivals A_i is equal to the total number of completions C_i, or that the initial state $n_i(0)$ is the same as the final state $n_i(T)$. Expressing the balance principle as an equation,

$$C_j = A_j \quad (* \text{ Job flow balance assumption } *)$$

Also, $\sum_{n=1}^{N} C_j(n) = \sum_{n=1}^{N} \sum_{i=0}^{M} C_{ij}(n)$

$$\therefore \quad C_j = \sum_{i=0}^{M} C_{ij}$$

But $q_{ij} = C_{ij} / C_i$ from this quantity we may derive,

$$C_j = \sum_{i=0}^{M} C_i\ q_{ij}$$

Now dividing by the time interval and employing the definition $X_i = C_i / T$, we obtain the job flow balance equations.

$$\boxed{X_j = \sum_{i=0}^{M} X_i\ q_{ij} \quad \text{where } j = 0,\ldots,M.}$$

The job flow balance equations have no unique solution in closed networks ((/Denning and Buzen 78/)). These equations can be used, however, to derive other important quantities and laws. For this purpose, define:

$V_i = \dfrac{C_i}{C_o}$ the mean number of completions at device i for completion from the system.

or , $V_i = \frac{A_i}{C_o}$ (* this is also called visit ratio i.e. mean number of visits per job to device i *)

Now, since $X_i = C_i / T$

we can derive the following law:

$V_i = X_i / X_o$; Forced Flow Law.

This law states that flow in any one part of the system determines the flow everywhere in the system.

On replacing each X_i with $V_i X_o$ in the job balance equations, we obtain the visit ratio equations:

$$V_o = 1 \quad \text{or}$$

$$V_j = q_{oj} + \sum_{i=1}^{M} V_i q_{ij} \quad , j = 1,\ldots,M$$

} visit ratio equations

The solution of the above equations is always possible if the assumption of connective structure (i.e. connected network) was valid ((/Denning and Buzen 78/)). Using the visit ratio, operational quantity and some other parameters, we can determine all the performance quantities. The visit ratio represents a workload parameters.

The computation of some performance quantities using the visit ratio, are given as follows:

* Response Time

Let $Q_i = X_i R_i$ (*from Little's law *)

$X_i = V_i X_o$ (*forced flow law *)

then

$$Q_i / X_o = V_i R_i$$

and $$\sum_{i=1}^{M} Q_i / X_o = \sum_{i=1}^{M} V_i R_i$$

$$\therefore \boxed{R = \sum_{i=1}^{M} V_i R_i}$$

This new law is called the General Response Time Law.

Note:

We can derive the response time (in an interactive system) formula directly from Little's law.

i.e. $R = Q / X_o$

But the response time in any interactive system is the time spent in the wait-think cycle. This means the interactive response time is R + Z (where R is the system response time and Z is the think time). Also, the parameter Q in the interactive

systems specifies the number of users observed in the wait-think cycle, say M customers. Hence, Little's formula can be re-written as follows:

$$M = (Z + R)\, X_o$$

i.e. $\boxed{R = M/X_o - Z}$ is the response time law.

* Utilization: Since $X_i = V_i X_o$

then, $$\frac{X_i}{X_j} = \frac{V_i X_o}{V_j X_o} = \frac{V_i}{V_j}$$

and since

$$U_i = X_i S_i \quad \text{then using the above ratio,}$$

$$\frac{X_i S_i}{X_j S_j} = \frac{S_i V_i}{S_j V_j}$$

i.e. $\boxed{\frac{U_i}{U_j} = \frac{S_i V_i}{S_j V_j}}$ we will assume these ratios are the same for all N.

This assumption is used to study the system bottlenecks. Device i is saturated if its utilization is approximately 100%.

If $U_i = 1$, the Utilization Law implies that

$$X_i = 1/S_i$$

Hence, for any device i, there should be $U_i \leqslant 1$ and $X_i \leqslant \frac{1}{S_i}$. Let the subscript b refer to any device capable of saturating as N becomes large. Such devices are called <u>bottlenecks</u> because they limit the system's overall performance. Since the ratios U_i/U_j are fixed, the device i with the largest value of $V_i S_i$ will be the first to achieve 100% utilization as N increases. Thus we see that whenever device b is a bottleneck,

$$V_b S_b = \max \quad V_1 S_1, \ldots, V_M S_M \quad .$$

Hence bottlenecks are determined by device and workload parameters.

2. <u>State Transition Balance:</u>

The number of entries to each state is the same as the number of exits from that state during the observation period. Using this assumption we can establish the state space balance equations.

Since

n_i = the operational state of the device i (i.e. number of customers either enqueued or in service at device i).

we can define

$\overline{n} = (n_1, n_2, \ldots, n_m)$ is the operational state of the system (or system state space).

and,

a.....$r(\bar{n},\bar{m}) = C(\bar{n},\bar{m}) / T(\bar{n})$, the transition rate from state $\bar{n}$ to $\bar{m}$ is the number of transitions per unit time while $\bar{n}$ is occupied. $C(\bar{n},\bar{m})$ denote the number of one-step state transitions observed from $\bar{n}$ to $\bar{m}$. The one-step state transition (from $\bar{n}$ to $\bar{m}$) means the system moves from state $\bar{n}$ to state $\bar{m}$ without passing through any observable intermediate state.

Now, let:

$P(\bar{n}) = P(n_1,n_2,\ldots,n_m)$ is the fraction of total observation period T, that the system is in state $(n_1,n_2,\ldots,n_m)$.

$P(\bar{n}) = \frac{T(\bar{n})}{T}$ (from the definition).

b..... i.e. $T(\bar{n}) = P(\bar{n})\, T$

With the flow balance principle we can write the conservation of transition equations:

c.....$\sum_K C(\bar{k},\bar{n}) = \sum_m C(\bar{n},\bar{m})$ for all $\bar{n}$.

From a, b, c, we obtain the state space balance equations:

$$\sum_K P(\bar{k})\, r(\bar{k},\bar{n}) = P(\bar{n}) \sum_m r(\bar{n},\bar{m})$$

for all $\bar{n}$ in which $r(\bar{n},\cdot)$ is defined.

This assumption is quite important, since the job flow is insufficient to find flows in a closed network or to compute response times accurately. These quantities depend on how customers distribute throughout the network. The state transition balance considers the problem of customer distribution and therefore, it will give more accurate results throughout the calculations of the performance quantities.

3. One Step Behaviour:

The only observable state changes result from a single customer either entering the system or moving between a pair of devices in the system, or exiting from the system. This means that $n_i(t)$ can only change in steps of $\pm$ 1. There is, at most, one arrival or one completion at any instant; no arrival coincides with a completion.

If $n_i(0) = n_i(T)$ at any device i in the system, and if $n_i(t)$ can only change in steps of $\pm$ 1 at any device i in the system, then $A_i = C_i$ and also the number of transitions from state n to state n + 1 must equal the number of transitions from state n + 1 to state n :

$$\therefore A_i(n) = C_i(n+1) \qquad n = 0,\ldots,N-1$$

combining this observation with the preceding definitions, gives

$$P_{A_i}(n) = P_{C_i}(n) \qquad n = 0,\ldots,N-1$$

Thus, the arriver's distribution and the completer's distribution are identical whenever flow balance and one-step are satisfied.

Finally, using the above assumptions we can derive the recursive laws:

$$P_{A_i}(n) = Y_i(n)\, S_i(n)\, P_{A_i}(n) \quad \ldots\text{First Recursive Law.}$$

$$P_i(n) = Y_i(n-1)\, S_i(n)\, P_i(n-1) \quad \ldots\text{Second Recursive Law.}$$

4. Homogeneity:

To apply the first and second recursion laws, it is necessary to measure or estimate the values of $Y_i(n)$ for $n = 0, 1,\ldots,N-1$ and $S_i(n)$ for $n = 1, 2,\ldots, N$. In some cases, the number of independent variables can be reduced significantly by making one or both of the following assumptions:

$$Y(0) = Y(1) = \ldots = Y(N-1) = \text{constant} \ldots\ldots a_1$$

$$S(1) = S(2) = \ldots = S(N) = \text{constant} \ldots\ldots a_2$$

Using a_1 and a_2 with the following laws:

$$Y_i/Y_i^o = 1/(1\, P_i(N)) \quad ,$$

$$Y_i^o = \sum_{n=1}^{n-1} P_i(n)\, Y_i(n)$$

we obtain:

$$\boxed{Y_i(n) = Y_i}$$, similarly using a_2 and

$$S_i = \sum_{n=1}^{n} P_{C_i}\,(n-1)\, Y_i(n)$$ we obtain

$$\boxed{S_i(n) = S_i}$$

Equation a_1 is called the assumption of Homogeneous arrivals; it asserts that the arrival rate is independent of the queue size n. Equation a_2 is called the assumption of Homogeneous Services (HST); it asserts that the mean time between completions is independent of n. These equations are examples of general operational techniques of simplifying problems by introducing homogeneity assumptions that allow a set of conditional rates to be replaced by a single, unconditional value ((/Buzen and Denning 80/)). We may also define a Routing Homogeneity as follows:

The routing frequencies for a given total load (N) are independent of the system's state and device homogeneity as the output rate of a device is determined completely by its queue length, and is otherwise independent of the system's state.

4.3.3.2. The Operational Solution of the Simple Closed Queueing Network:

A queueing network will have a tractable solution if one or more of the following conditions are met ((/Chandy and Sauer 78/)):

1. State Space Size:

   The state space balance equations can be mechanically generated and numerically solved in an adequately short amount of time.

2. State Transition Structure:

   The state transitions are such that recursive techniques may be used to obtain the fractions of time of a few states and then, the queue length distributions can be expressed in terms of these states.

3. Product Form:

   The equilibrium state fraction of time distribution consists of factors representing the states of the individual queues, i.e. $P(n_1,\ldots,n_M) = (1/G)\ P(n_1)\ldots\ldots P(n_M)$ is the fraction of time that the ith queue is in state n_i and $P(n_1,\ldots\ldots,n_M)$ is the fraction of time that the network is in state $(n_1,\ldots,n_M)$. Where G is a normalization constant chosen so that the fractions of time sum to one.

The tractable solution can be obtained through homogeneous assumption since homogeneity is often a reasonable approximation ((/Denning and Buzen 78/)). Also, Denning and Buzen prove that any closed queueing network, such as the cyclic queueing network, has a tractable solution, since the operational solution can be represented as a product form solution:

$$P(n_i,n_2,\ldots\ldots,n_M) = F_1(n_i)\ F_2(n_2)\ldots F_M(n_M)\ /G$$

where the factor for device i is:

$$F_i(n) = \begin{cases} 1 & n = 0 \\ X_i^n\ S_i(n)\ S_i(n-1)\ \ldots S_i(1) & n > 0 \end{cases}$$

and G is a normalization constant. The $S_i(n)$ are the service functions. The X_i are a solution of the job flow balance equations, for closed system $X_i = V_i$.

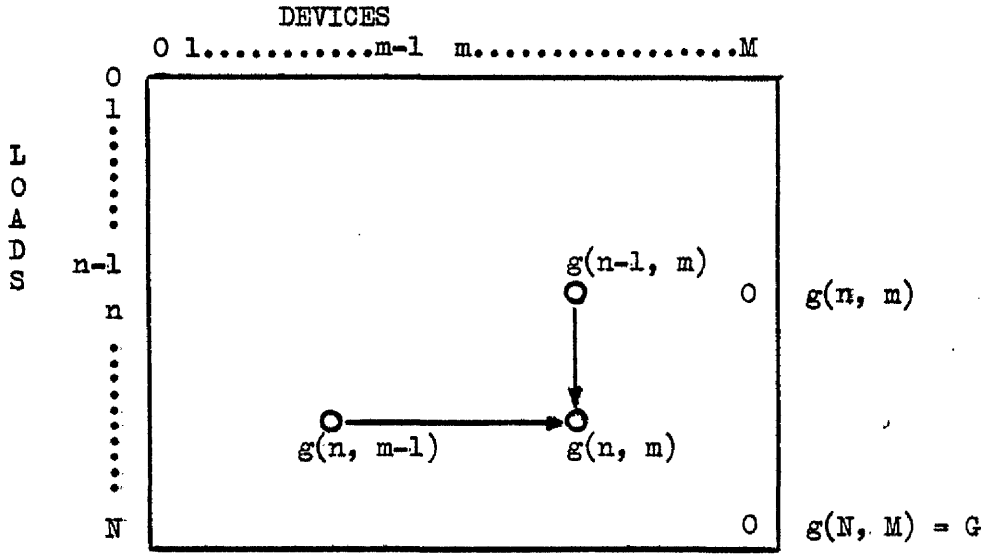
To simplify the above operational solution we may use the assumption of homogeneous service (HST) (i.e. $S_i(n) = S_i$ for all n).

The operational solution for $P(\bar{n})$ is mathematically neat but not obviously useful. We will try to introduce its uses with a closed system like the cyclic model assumming the last simplification was included.

Let N represent maximum no. of customers.

M " " " " devices.

Now, to compute the normalization factor we follow the algorithm developed by Buzen ((/Buzen 71/)) ((/Buzen 73/)); the algorithm fills in numbers in a two-dimensional matrix g. The columns of g correspond to devices, whereas its rows correspond to loads.



The Matrix g

The computation starts with 1's in the first row and 0's in the first column below the first row. A typical interior element is computed from:

$$g(n,m) = g(n,m-1) + Y_m\, g(n-1,m),$$

where $Y_m = V_m S_m$. The normalizing factor (constant) G is $g(N,M)$. It can be computed in 2MN arithmetic operations. The complete algorithm to compute G is given in Appendix B.

The results of the important system performance indices using the operational solution are:

* $Q_i(n) = Y_i^n \frac{g(N-n, M)}{g(N,M)}$ proportion of time $n_i \geq n$.

* $U_i = Q_i(1) = Y_i \frac{g(N-1, M)}{g(N,M)}$, Utilization.

* $X_o = \frac{g(N-1, M)}{g(N,M)}$, system throughput.

* $\bar{\bar{n}}_i = \sum_{n=1}^{N} Y_i^n \frac{g(N-n,M)}{g(N,M)}$, Queue length of device i.

4.3.3.3. Example:

Assume a simple closed queueing network was given as follows (see figure 4.6.):

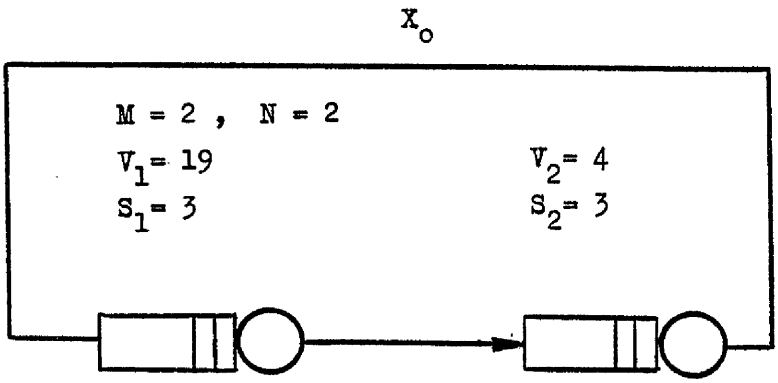


Figure 4.6

we had:

$Y_1 = V_1 S_1 = 6.33$ seconds.
$Y_2 = V_2 S_2 = 1.33$ seconds.

The table below shows the matrix g for loads N = 1,...,5 ((/Denning and Buzen 78/)):

	0	1	2
0	—	1.00	1.00
1	0	6.33	7.67
2	0	40.1	50.3
3	0	454	321
4	0	1609	2037
5	0	10190	12906

Matrix g

For example when N = 2 then,

$X_o(2) = \frac{g(1,2)}{g(2,2)} = 7.67/50.3 = .152$

The mean queue length at device i when N=2 is :

$$\bar{\bar{n}}_1 = \sum_{n=1}^{2} Y_1^n \frac{g(2-n,2)}{g(2,2)}$$

$$= \frac{(6.33)(7.67) + (6.33)^2(1.00)}{50.3}$$

$$= 1.762.$$

The utilization of device 1 when N = 2 is :

$$U_1 = Y_1 \frac{g(2-1,2)}{g(2,2)} = 0.95.$$

4.3.4. Operational Aspects of the Interactive Computer Systems:

4.3.4.1. System Outline:

A typical modern large-scale interactive computer system is depicted in figure 4.7. The entire system is composed of a processing system and a finite population of terminal users. Each of these users thinks for a while and then requests a computation (hereafter called a transaction) to be performed by the processing system, by typing a command line at his terminal. The transaction thus requested, is received and placed in the eligible queue until main memory availability permits its admission for service.

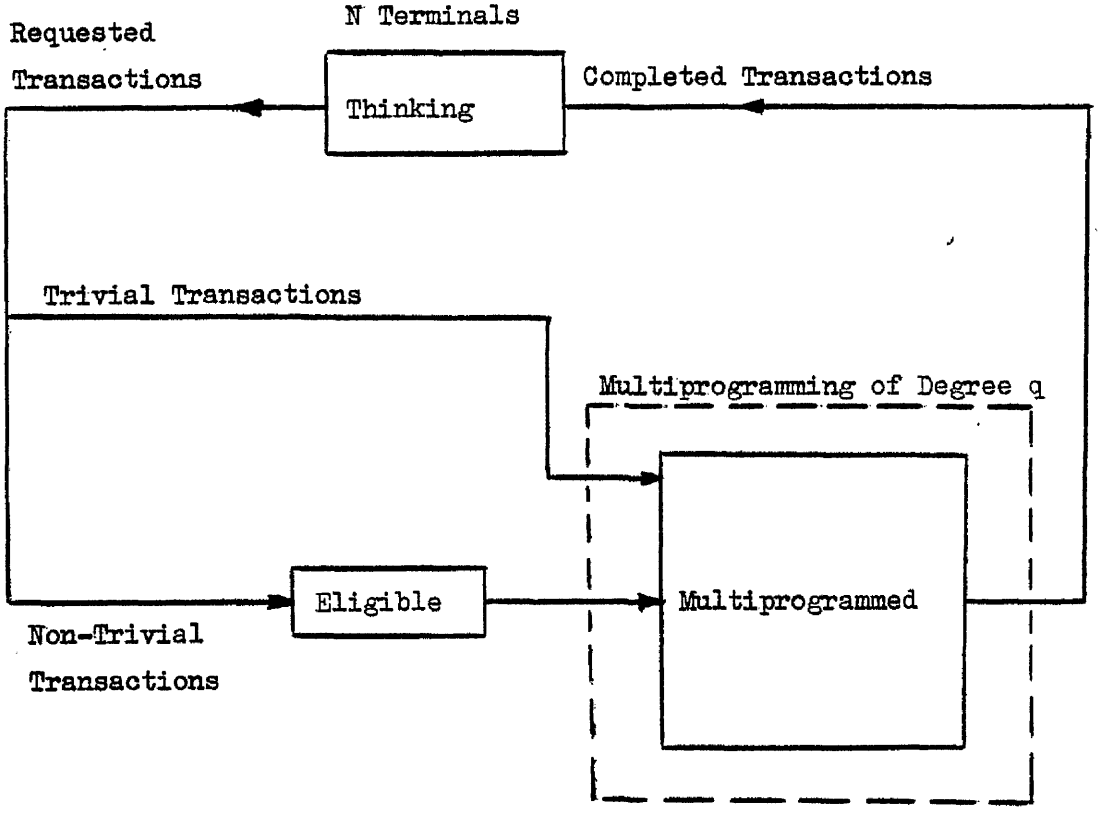


Figure 4.7: Interactive Computer System (Schematic Diagram)

The processing system consists of a central processing unit (CPU) and a two-level virtual-memory consisting of a primary memory (PM) as the first level and a large secondary memory (SM) as the second. Transactions receiving service are said to be in the multiprogramming queue, while a customer is said to be "thinking" from the time his transaction is completed until he has entered the next one.

The system may classify the customer transactions as either "trivial" or "non-trivial". Those of the first case are admitted for immediate service. In the second case, a transaction may have to wait in the eligible queue. Transactions may be further classified into different transaction classes.

The maximum number of jobs simultaneously cycling inside the multiprogrammed state is called the degree of multiprogramming and should be carefully determined by considering, at least, the transaction's demand for PM space and the total size of PM space available to customer transactions. In this chapter the computer system is assumed to use a constant degree of multiprogramming equal to q.

The operational queueing model (see figure 4.8.) used in this study characterizes requests by their alternating use of the central processing unit (CPU) and various input-output devices. At the end of each interval of CPU processing, transactions (i.e. processes) move to one of the peripheral devices.

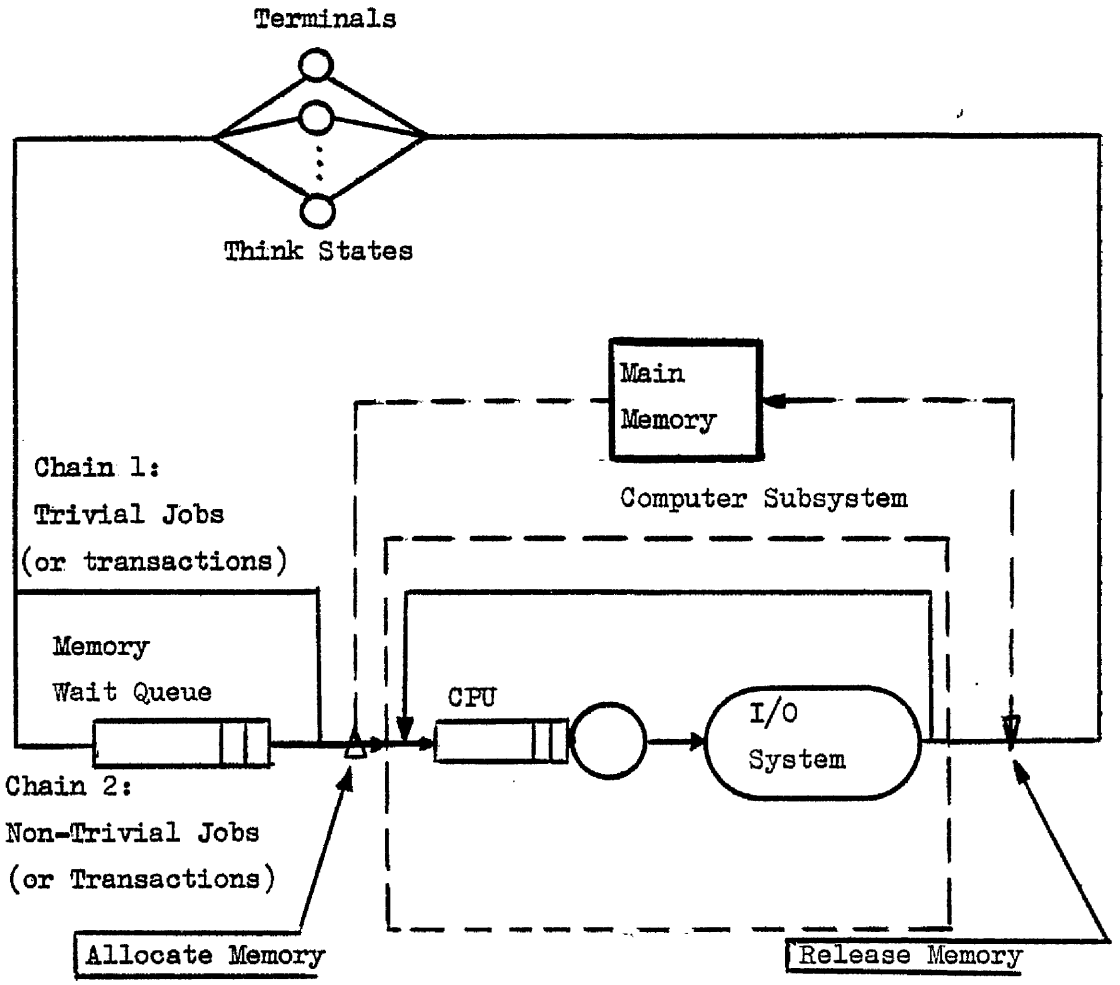


Figure 4.8: Interactive Computer System (Queueing Network Diagram).

The operational aspects of the above interactive system will be analysed in this chapter using approximation techniques. There are two major approaches to approximate solution, aggregation (decomposition) and diffusion ((/Courtois 75/)) ((/Chandy and Sauer 78/)). The reason for using approximation techniques is the difficulty of representing the effects of virtual memory in any queueing network. "The most important applications of approximation have been for virtual memory, blocking and other behaviours which cannot be represented directly in a queueing network model" ((/Denning and Buzen 78/)).

The system can be extended easily to deal with scheduling strategies. For this purpose we recommend the work of Brad ((/Brad 77/)).

To analyze the interactive computer system given in figure 4.8. we will analyze the system in two steps (see figure 4.9.):

1. The multi-class closed queueing network subsystem.
2. The overall interactive system using decomposition techniques.

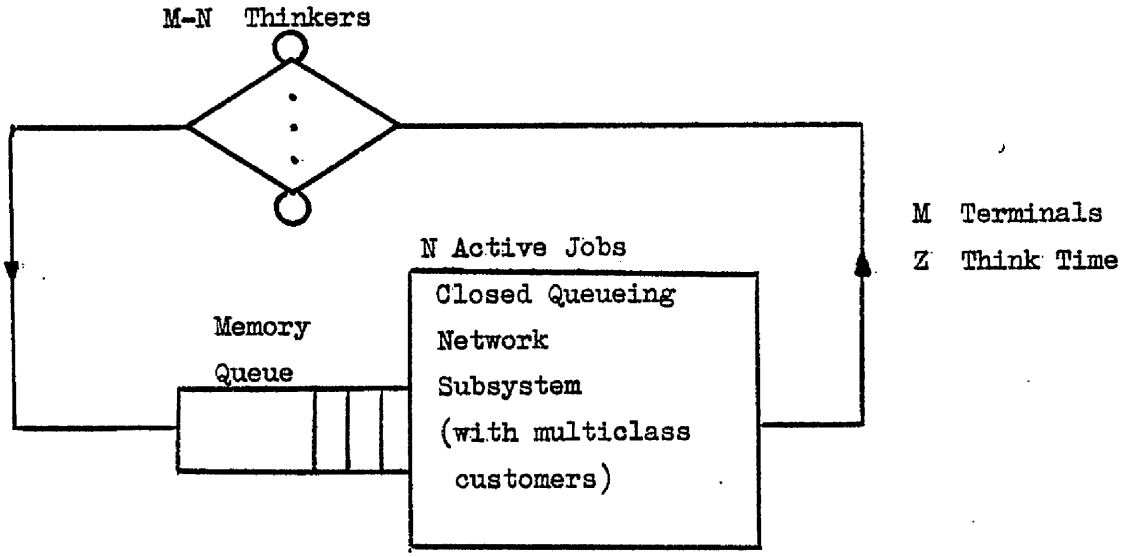


Figure 4.9: General Terminal Model With Memory.

4.3.4.2. The Operational Aspects of a Multi-Class Closed Queueing Network Subsystem:

The operational aspects of a multiclass closed queueing network was first studied by Roode ((/Roode 79/)), in which it was called the multi-class operational analysis.

The representation of multiclass subsystem is given in appendix B.2. along with the calculation procedure of the normalization factor.

Briefly, we can formulate the important performance indices. For more information the reader is referred to Roode ((/Roode 79/)).

* The Utilization:

$$U_{MR} = \frac{1}{G(N,M,R)} \sum_{m=1}^{N} m(X_{MR}\, S_{MR})^m \sum_{P=0}^{N-m} \frac{1}{P+m}\, h^{(M)}(P,R-1)\; g(N-m,P,M-1,R)$$

* The Average Queue Lengths:

$$\bar{n}_{MR} = \frac{1}{G(N,M,R)} \sum_{m=1}^{N} m(X_{MR}\, S_{MR})^m \sum_{P=0}^{N-m} h^{(M)}(P,R-1)\; g(N-m,M-1,R).$$

(continued overleaf)

* The Average Response Time at Centre M:

$$R_{MR} = \frac{\overline{n}_{MR}}{X_{MR}}$$

* The Average Request Throughput Rate at Centre M:

$$X_{MR} = U_{MR} / S_{MR}$$

4.3.4.3. The Operational Aspects of the Overall Interactive Cumputer System:

In the previous section, the closed queueing network subsystem allowed any number of customers to circulate in it. This is not realistic however, when the number of terminals is quite large. Usually a computer system will only allow a certain number of jobs in the subsystem, consisting of the CPU and I/O devices. All other requests for service which have not yet been allotted main memory (and thus are denied access to the subsystem) have to wait outside until jobs depart and main memory becomes available for them. It is assumed that their queue is served on a First Come First Served basis.

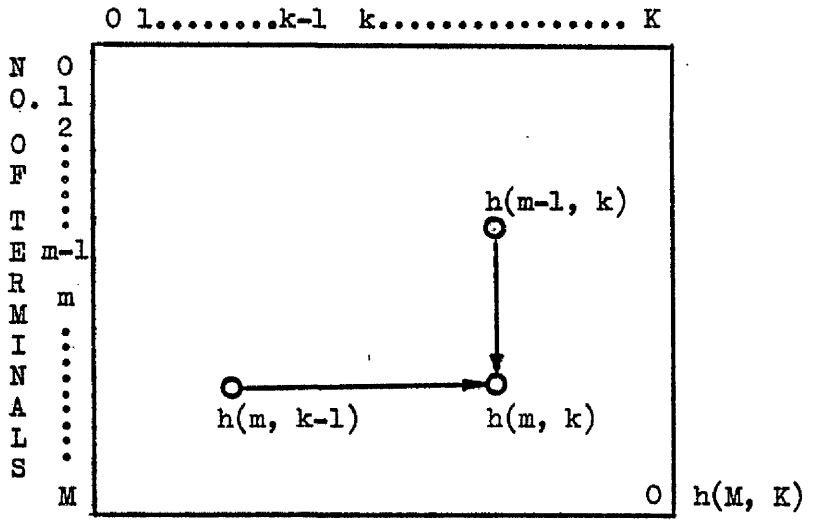
Performance measures for the interactive system (cf. figure 4.9) are best computed using the decomposition technique. The calculation can be performed depending on two factors:

1. The number of terminals.
2. The degree of multiprogramming.

We present the performance measures (indices) calculated using the first factor ((see/Denning and Buzen 78/)). For the second method the reader is referred to ((/Von Mayrhauser 79/)).

Consider, each of the M terminals has think time Z. The number of active jobs is denoted by N, and the number of thinking terminals by M - N. The closed queueing network subsystem has K devices with homogeneous service times and visit ratios independent of N.

By treating the terminals as a "device" whose service function is Z/n when there are n thinkers, we can employ efficient computational procedures to compute a normalizing constant for this system ((/Williams and Bhandiwad 76/)). The algorithm fill in a matrix h.



The Matrix h

When the computation of the normalization constant has been finished, the performance measures can be computed from the formulae below:

* Throughput:

$$X(M) = \frac{M}{Z} \frac{h(M-1,K)}{h(M,K)}$$

* Response Time:

$$R(M) = M/X(M) - Z$$

* Mean Active Load:

$$\overline{N} = M - Z\, X(M).$$

Finally, many other aspects of the interactive computer system can be represented using operational analysis technique. For this purpose the reader is referred to ((/Balbo and Denning 79/)) ((/Denning 80/)) ((/Slutz and Traiger 74/)).

CHAPTER 5

THE PERFORMANCE-ORIENTED DESIGN APPROACH

5.1. Introduction:

The approaches discussed in the previous chapters for the investigation of interactive systems have utilized either analytic or simulation techniques. But none of these methods' models, however, can answer the questions posed by the problem of prediction and dynamic control of the new systems. Some of these questions are:

* What is the optimum segment size into which programs and data are divided for multiprogramming?
* What are the input variables that define the system environment?
* What are the control variables that determine the system performance?
* What is the sensitivity of the system performance to changes in the control and/or the input variables?
* How does the computer deal with its changing environment?
  Are the control mechanisms adaptive or do they have a fixed structure?
* How does the computer learn to adapt to its environment?
* When does it pay to have an adaptive control system?
* What is the way to measure or estimate the input parameters?
* What is the most economical system configuration that will do the required job?
* What is optimum?

To the best of my knowledge, none of the analytic models discussed in the previous chapters can answer all these questions. "The estimation problem is not really an integral part of either operational analysis or stochastic modelling" ((/Buzen 79/)). With respect to simulation models, it has been pointed out by Nielsen ((/Nielsen 67/)) that the existing models are both too costly and inadequate to solve the apparently simple problem of optimum system configuration, let alone the problem of determining optimum prediction and control. In the same paper, he develops a reasonably detailed simulation model to analyse the performance of the IBM 360/67 time-sharing system for different configurations and different amounts of overhead.

The model presented in that paper and the corresponding results were considered of value to gain insight into the problems of hardware configuration and software modification for a given set of input parameters, but they cannot be used for dynamic control and prediction when the input parameters change. And this is true for

any simulation model.

Due to these shortcomings and the complexity of real systems, neither analytic nor simulation models alone can solve the problem of the optimum control of the system.

Most mathematical models which are quite close to reality are based on the implementation of the optimization models in queueing theory ((/Trivedi and Wagner 79/)) ((/Kinicki 78/)) , whereas some researchers use a different approach ((/Decegama 72/)).

Queueing models which employ the optimization technique are called prescriptive queueing models. Prescriptive queueing models are of two types ((/Gupta and Verma 80/)):

* Static (design) models.
* Dynamic (control) models.

In static (design) and dynamic (control) models, we optimize longterm average criteria such as cost or profit dependent and independent of time respectively. The latter models are sometimes dependent of the system state too. The static (design) models are often called Performance-oriented design models ((/Von Mayhauser 79/)). These models will be critically reviewed in this chapter.

The construction of Dynamic (control) models can be performed using the following fundamental approaches ((/Decegama 70/)):

* Determine in advance the optimum settings of the control variables for each different expected and possible configuration, and have the operater initiate the switching from one control mode to another at predetermined times or in emegencies.
* Let the system sense the environmental changes and switch automatically to the proper control mode.

Optimization theory plays a key role in producing optimal designs from the performance-oriented design models. For a fuller treatment of optimization theory and methods, the reader is referred to ((/Walsh 75/)) ((/Adby and Dempster 74/)) ((/Gottfried and Weisman 73/)) ((/Lenberger 73/)).

5.2. Problem Statement and Solution:

When a computer system is planned, the designer goes through three stages to find the system configuration which meets the user requirements and does not exceed a specific upper cost limit. Figure 5. 1. shows these stages ((see/Von Mayhauser 79/)).

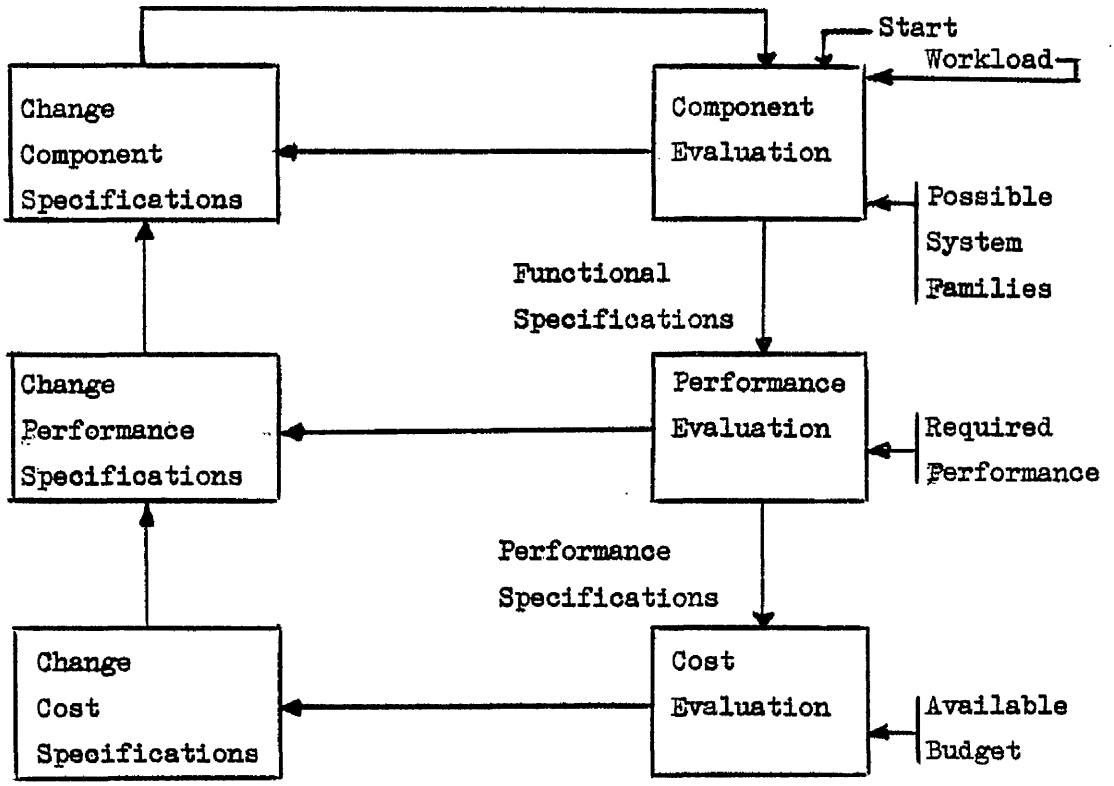


Figure 5.1. Design Stages.

The above Design stages are are a combination of hierarchical and iterative design methodology. This methodology requires the following steps:

1. Characterization of the anticipated workload. This is quite a difficult task ((/Ferrari 78/)). It can, however, be studied indirectly by providing an evaluation of the sensitivity of the optimal configuration with regard to changes to the workload parameters.
2. Derivation of functional specifications using the workload model and the specification of the general computer system family. The computer system family, for simplicity, will be of fixed type. Hence, these fixed types should be quite general.
3. Selection of the optimal configuration using performance and cost measures to assess the quality of the solution. The selection procedure is shown in Figure 5.2. and requires certain input parameters. The selection procedure determines, at best, an optimal solution for the model. The optimization is, however, as valid as its input parameters.

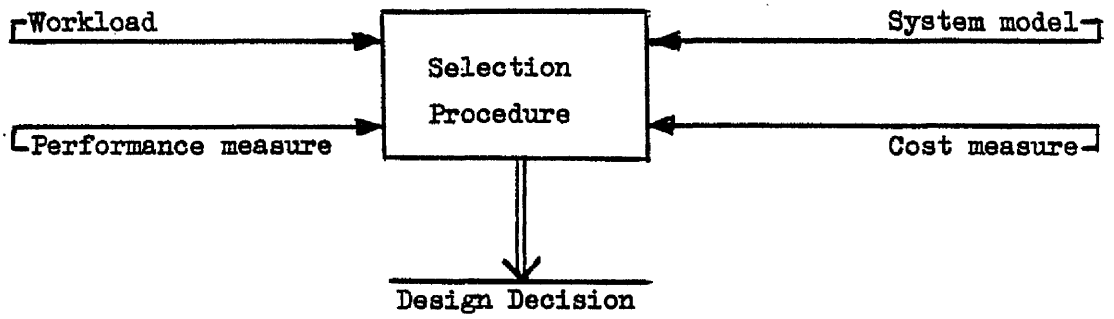


Figure 5.2. Cost/Performance Optimal Design.

The decision process will lead to a possible system design. The desirable tool that helps in making decisions (i.e. Performance-cost evaluation tool) depends upon three factors:

* Mathematical formulae for performance measures.
* Cost function.
* Workload parameters.

These factors specify the optimization problem. Such a tool is called performance-oriented design tool ((/Sigmon 79/)). Performance-oriented design produces systems whose performance objectives can be guaranteed to be met when the system is built ((/Graham 78/)). An important aspect of this approach is to be able to show that the resulting solution to the optimization problem will be a globally optimum solution.

We will try to present in this chapter several performance-oriented design models for computer systems. In all cases, the computer system is modelled via an operational queueing network ((or an exponential queueing network (Markovian queueing network))). The optimization problems which are established, seek to maximize the throughput or minimize the response time of the modelled system subject to a cost constraint. The decision variables for these design models include such items as the speeds of the devices, the capacities of the devices and a file assignment. For each problem it has been proven that any locally optimum solution is indeed a globally optimum solution thus guaranteeing the optimality of the design.

5.2.1. The Selected Model:

The selection of a general queueing network model, suitable for modelling multiprogrammed computer systems, is quite important. The selected model will be used by the performance-oriented design method to construct several design optimization problems. Such a general queueing network model was introduced by Buzen ((/Buzen 71/)), and was called the Central Server Model (CSM). Since 1971, this model has been utilized as an analytic tool to evaluate the performance of computer systems. The CSM is indeed a realistic and cost effective means for performance evaluation of computer systems ((/Hughes and Moe 73/)) ((/Rose 78/)) ((/Giammo 76/)). Figure 5.3. shows the structure of CSM:

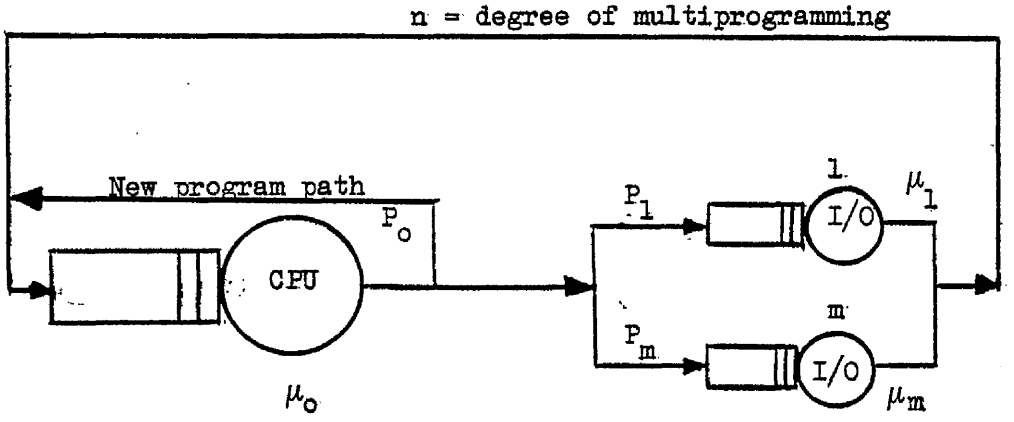


Figure 5.3. The CSM

The central server Model incorporates hardware, software and workload aspects of an actual system ((/Kinicki 78/)). These aspects have been extensively analysed by Von-Mayrhauser ((/Von-Mayrhauser 79/)). The model has been used by many researchers to construct several optimal computer system designs ((/Trivedi and Wagner 79/)) ((/Trivedi and Kinicki 78/)) ((/Kinicki 78/)) ((/Sigmon 79/)).

In this chapter we will present the optimization problems introduced by Sigmon ((/Sigmon 79/)) using the CSM. We will also introduce further extensions, especially in the area of the optimization of the design of interactive computer system.

5.3. Optimal Design of Computer Systems Without Virtual Memory:

The selected computer system model (i.e. CSM) operating in a multiprogramming environment without virtual memory, where each active program's address space resides in main memory until its completion, was studied by ((/Buzen 73/)) and developed to permit some scheduling discipline, such as PS and pre-emptive LCFS by ((/Chandy, Howard and Towsley 77/)). The problem was represented as an exponential queueing network. This representation was used by Sigmon ((/Sigmon 79/)) to construct a design optimization problem which can be summarized as follows:

* Let all service facilities operate under a FCFS scheduling and their service time distribution is negative exponential with mean service rate M_i $(i = 0,\ldots,m)$.
* Let n be a fixed degree of multiprogramming.
* P_o is probability that upon completion of service at the CPU the program terminates and a new program enters the system via a new program path.
* P_i $(i = 1,\ldots,m)$ is the probability that upon leaving the CPU a program will next require service at the ith I/O.
* System throughput = $P_o \mu_o U_o$, where U_o is the utilization of CPU.
* Let

  t_i be the average number of program visits to the ith facility.

  J_i be the total number of work units processed by a program at ith facility.

  then the

  - average number of instructions executed are $\frac{J_o}{t_o}$ by the CPU between two I/O requests.

  and

  - the number of information units are $\frac{J_i}{t_o}$ transferred between the ith facility and main memory per unit.
* The speed of ith service facility (b_i) is

$\frac{J_i}{t_i}\mu_i$ where i = 0,....,m.

* Now the throughput as a function of the speed of service facilities (b_i's) can be computed as follows:

$$T(\bar{b};n) = P_o \mu_o U_o = P_o \frac{G(F;n-1)}{G(F;n)}$$

where

$$U_o = r_o \frac{G(F;n-1)}{G(F;n)}$$

$$G(F,n) = \sum \left(\prod_{j=0}^{m} r_j^{n_j} : \sum_{j=0}^{m} n_j = n \right)$$

$$F = (r_o, r_1, \ldots, r_m)$$

$$\bar{b} = (b_o, b_1, \ldots, b_m)$$

$$r_o = \frac{y_o}{\mu_o} = \frac{1}{\mu_o} = \frac{J_o}{t_o b_o}$$

$$r_i = \frac{y_i}{\mu_i} = \frac{P_i}{\mu_i} = P_i \frac{J_i}{t_i b_i} \quad , i=1,\ldots,m.$$

$\bar{y} = (y_o, \ldots, y_m)$ is the left hand eigen-vector of the transition probability matrix of CSM. Since $\bar{y}$ is determined only to a multiplicative constant, we choose $y_o = 1$.

The above design problem of Buzen ((/Buzen 73/)) and Kleinrock ((/Kleinrock 76/)) has been expressed as an optimization problem by Trivedi and Kinicki ((/Trivedi and Kinicki 78/)) as follows:

They assumed that the cost of the computer system is approximately the sum of the individual component costs and the main memory cost. The cost of each component is expressed in terms of a continuous power function of the device speed and the cost of main memory is assumed to be a linear function of n.

The aim is to maximize the system throughput,

$T(\bar{b}\ ; n)$ or alternatively, minimize

$Z(\bar{b}\ ; n)$ where $Z(\bar{b}\ , n)$ is the reciprocal of the throughput. In order to simplify the mathematical calculations ((/Sigmon 79/)), the decision variables of Z were changed from $\bar{b}$ to $\bar{r}$.

Minimize $Z(\overline{r}, n)$

subject to

$$C_o \left(\frac{J_o}{t_o r_o}\right)^{\alpha_o} + \sum_{i=1}^{m} C_i \left(\frac{P_i J_i}{t_i r_i}\right)^{\alpha_i} + Mem(n) \leqslant BUDGET$$

where

$r_i > 0$ and $i = 0,1,\ldots,m$.

and

C_i , α_i is positive real numbers.

i.e. maximize the throughput by determining the specific device speeds depending on the given fixed system topology, fixed degree of multiprogramming and a workload description (in terms of P_i's and t_i , J_i $(i = 0,\ldots,m)$).

The solution of the above design problem was given by Trivedi and Wagner ((/Trivedi and Wagner 79/)), in which they proved that it is a convex programming problem. Hence, this problem has the useful property that any locally minimal solution is indeed a global optimum.

5.4. Optimal Design of Computer Systems With Virtual Memory:

To represent a virtual memory system using the selected model (i.e. CSM) the following are assumed ((/Sigmon 79/)):

either, another I/O device is added to the system to handle the paging traffic,

or, all paging I/O is handled by one of the existing I/O devices whose capacity has been increased.

This design problem can be represented as an optimization problem as follows:

Let

* The total I/O activities consist of two parts - paging I/O and all other I/O.
* The average CPU burst between two paging I/O requests be given by the system lifetime function e :

  $e\left(\frac{M}{n}\right)$ or e (mem) where M represents total ammount of main memory and n is the degree of multiprogramming.

  Since M is fixed, then for simplicity the lifetime function can be reduced to e(n).

* The average CPU burst between two non-paging I/O requests be $W = \frac{Jo}{\sum t_i + 1}$

* The average CPU burst is then :

$$\frac{1}{E(n)} = \frac{1}{e(n)} + \frac{1}{W}$$

or

$$E(n) = \left[\frac{1}{e(n)} + \frac{1}{W} \right]^{-1}$$

* Since J_o is the total number of instructions to be executed per program, then

$$t_o = \frac{J_o}{E(n)} ,$$

but $P_o = \frac{1}{t_o}$ therefore,

$$P_o = \frac{E(n)}{J_o}$$

Also the non-paging I/O devices branching probabilities are:

$$P_i = \frac{t_i}{t_o} = P_o t_i$$

Similarly, as in the non-virtual memory problem, we can define the optimization problem as follows:

minimize $Z(\bar{r}, n)$

subject to

$$C_o \left(\frac{J_o}{t_o r_o} \right)^{\alpha_o} + \sum_{i=1}^{m} C_i \left(\frac{P_i J_i}{t_i r_i} \right)^{\alpha_i} + Mem(n) \leqslant BUDGET,$$

where

$r_i > 0$, $i=0,1,\ldots,m$ and

C_i , α_i are positive real numbers.

The only difference between the previous two design models is in the characterization of the workload. In particular, the branching probabilities in the virtual models are functions of the degree of multiprogramming, n, instead of being fixed.

Additionally, we can calculate using the virtual memory model, the CPU overhead generated by the page fault handler as follows:

$$J_o = J_o + \left[\frac{J_o}{e(n)} \right] PHF$$

where

$\left[\frac{J_o}{e(n)}\right]$ is the number of page faults that were generated and

PHF is the number of instructions executed by the page fault handler.

The above design optimization problem proved by Sigmon ((/Sigmon 79/)) to be a convex optimization problem.

Using the above design methods, a decision as to whether to use virtual memory or not can be taken. A particular advantage of this tool is that it provides a simple and inexpensive method of gaining insight into a large number of different system configurations operating under varying workloads and constrained by different cost estimates. The following example showing the difference between the virtual memory optimization problem and the non-virtual memory optimization problem is given by Sigmon ((/Sigmon 79/)).

Example:

This example demonstrates how a decision can be taken on whether to use virtual memory or not.

The performance-oriented method will be used as a tool to aid the designer to take such a decision. The comparisons will be based on the following models:

1. a multiprogrammed computer system without virtual memory and having three I/O devices.
2. a multiprogrammed computer with virtual memory and having three I/O devices one of which handles both paging and non-paging I/O.

For more information about this example see section 2.3. of Sigmon research work ((/Sigmon 79/)).

The model parameters are given in tables 5.1. and 5.2.

Figure 5.5. shows the graphs of optimal throughput versus dollars spent on main memory for the non-virtual memory and the virtual memory with three I/O devices. The dashed lines represent the results from the non-virtual model and the solid lines those from the virtual model with three I/O devices. The results from two total system budgets and for the three values of the page fault handler overhead PFH are plotted on the same graph. Each point of the virtual model's curves was obtained by choosing the optimal point after a discrete search over n, the degree of multiprogramming, was performed. The small numbers written beside each point of the virtual model's curves are the optimal degrees of multiprogramming.

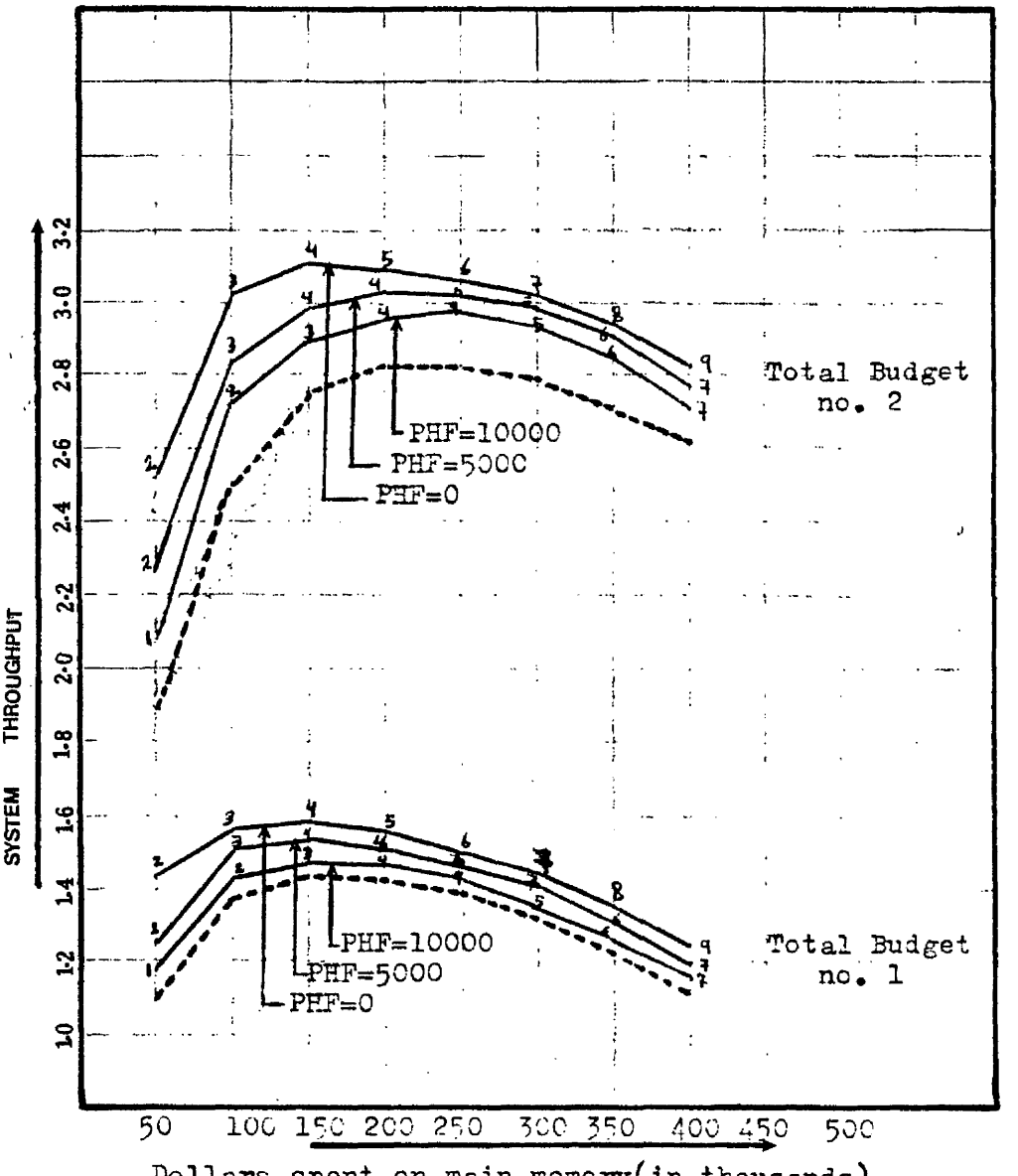


Figure 5.5 : Throughput vs. dollars spent on main memory for the non-virtual memory(dashed lines) and the virtual model.

Table 5.1.: Input Parameters for the System Without Virtual Memory and Three I/O Devices.

i	Device Name	J_i	T_i	P_i	C_i	α_i
0	CPU	400,000	20	0.05	1,147,835	0.55309
1	Drum	10,000	10	0.50	1,432,664	1.00000
2	Disc 1	6,000	6	0.30	707,648	0.67290
3	Disc 2	3,000	3	0.15	707,648	0.67290

Notes:
1. Memory Price \$1 / 32 - bit word.
2. Budget, \$ 1000,000 - \$ 2000,000, with 250,000 increment
3. Amount memory required by each program in the active set = 50,000 words.
4. n (degree of multiprogramming) = \$ MM/50000 where \$ MM is amount of money spent on main memory.

Table 5.2.: Input Parameters for System With Virtual Memory and Three I/O Devices.

i	Device Name	J_i	T_i	C_i	α_i
0	CPU	400,000	*	1,147,835.00	0.55309
1	Paging Drum	*	*	2,865,328.00	1.00000
2	Disc 1	6.000	6	707,648.00	0.67290
3	Disc 2	3.000	3	707,648.00	0.67290

Notes:
1. CPU burst between page faults is represented by $e(mem) = a\ mem^b$ where a = 4.69 and b = 2.88 (this function called life-time function).
2. Memory Price \$1 / 32 bit word.
3. Budget, \$ 1000,000 - \$2000,000 with 250,000 increment.
4. PFH = 0, 5000 and 10000.
5. n is given on the graph near each point as optimal degree of multiprogramming.
6. Budget 1 = \$1,250,000 and Budget 2 = \$1,750,000.

* these values are dependent on n and the life-time function.

Here, it seems that for all three values of PFH, the virtual curve lies completely above the non-virtual curve. Thus, for this set of values for the model parameters, we conclude that virtual memory will yield a performance increase when the paging I/O is handled by an existing I/O device.

5.5. The Selected Model Extension:

The goal of developing CSM was to make that model applicable to the optimum design of a terminal oriented computer system. The computer system family under investigation consists of a number of terminals (m in this case) connected to a central subsystem (cf. figure 5.6.). The queueing network model for this system family represents the m terminals as a multiple server node with multiplicity m connected to a CSM which models the central subsystem (cf. figure 5.7.)

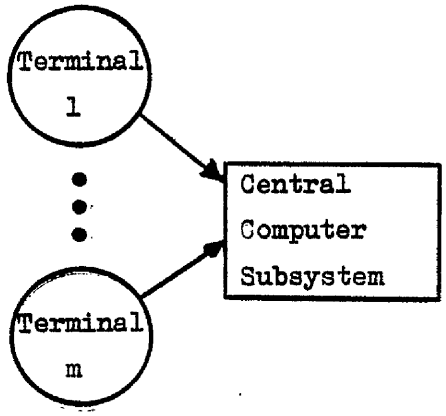


Figure 5.6. Interactive System with m Terminals.

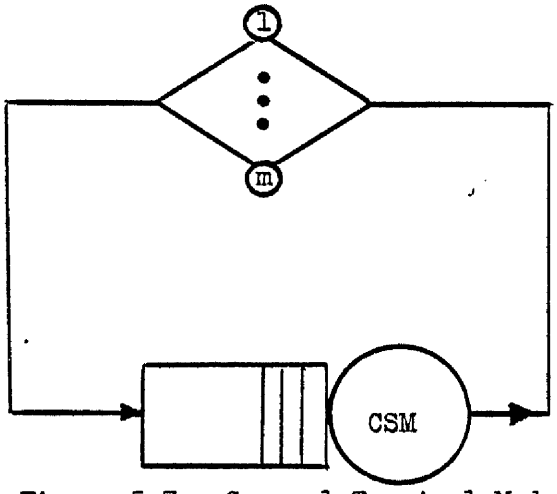


Figure 5.7. General Terminal Model.

The workload description of the terminals is condensed in the average think time Z, i.e. how much time elapses on average, between the return of a request from the CPU to the terminal until a new request is issued from the terminal to the CPU. This is the time the user spends to prepare and to input messages. Since there are m terminals, the maximum degree of multiprogramming is m. There is no queue at the terminals. It will take a certain time R, to process the user request in the submodel. Since the model of the subsystem is of the central server type, all assumptions and restrictions mentioned in section 5.3. of this chapter will apply. Figure 5.8. shows the terminal central server model (TCSM). The parameters M_i, P_i ($i = 0,1,\ldots,k$) correspond to those in section 5.3. Instead of taking the new program path which models the arrival of a new program, the processed request

now returns to the terminal and after a delay of Z seconds that terminal issues a new request. The jobs will be distributed in this model between the terminals and the CSM. When there are m-n jobs (i.e. user or customers) thinking, then n jobs are being processed in the CSM. This means that the degree of multiprogramming in the CSM varies between 0 (all terminals thinking) and m (all terminals have issued requests and are waiting for an answer).

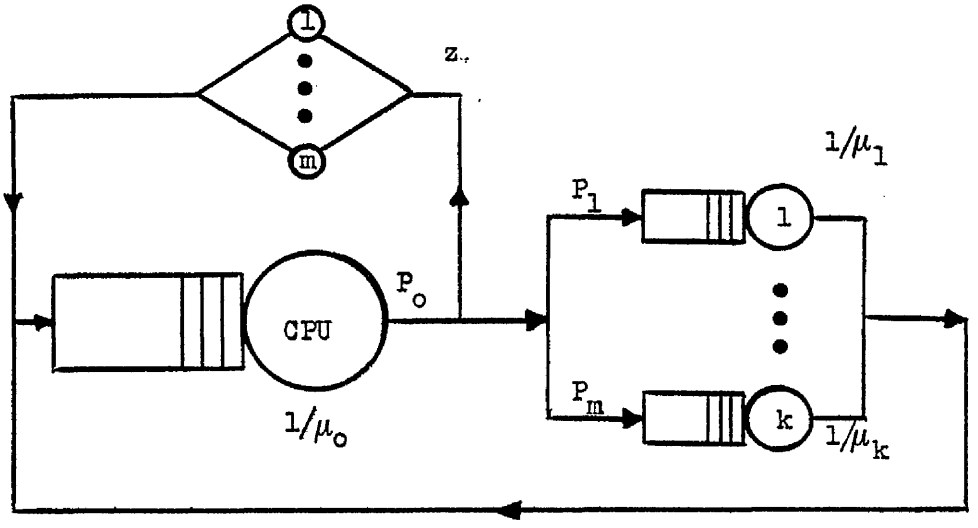


Figure 5.8. Terminal Central Server Model (TCSM).

The number of terminals, m, can be quite large. The number of active jobs in the computer depends on the amount of main memory available for these jobs. In many applications not all m programs will fit into main memory. Therefore it is more realistic to assume that the degree of multiprogramming in the subsystem is at most $n \leqslant m$. In the model this results in the formation of an additional queue Q_B in front of the submodel (figure 5.9.). The subsystem blocks jobs from entering, when the maximum degree of multiprogramming is reached. Figure 5. 10. shows the detailed terminal model with blocking and a central server subsystem (TBCSM). The dashed box marks the CSM submodel with degree of multiprogramming of at most n. Whenever a new terminal request is issued and the maximum number of jobs is in the CSM, the request queues at Q_B. The departure of a job from the CSM. Then the job is loaded into main memory. There is no swapping. Once a program has been loaded, it will stay in main memory until it terminates.

The optimal design of the TCSM and TBCSM models have been studied intensively by Von Mayrhauser ((/Von Mayrhauser 79/)) and we will, in

the next sections summarise their findings.

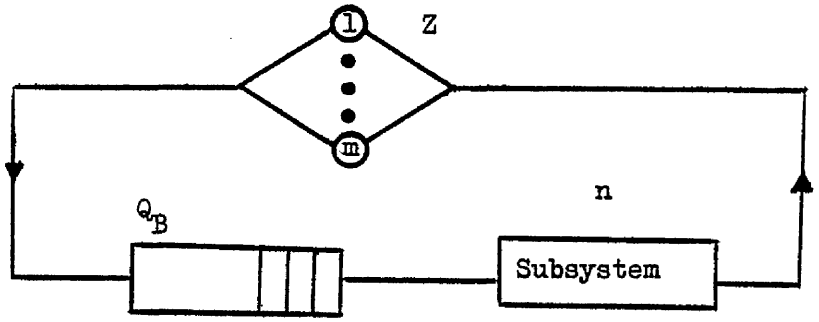


Figure 5.9. General Terminal Model With Blocking.

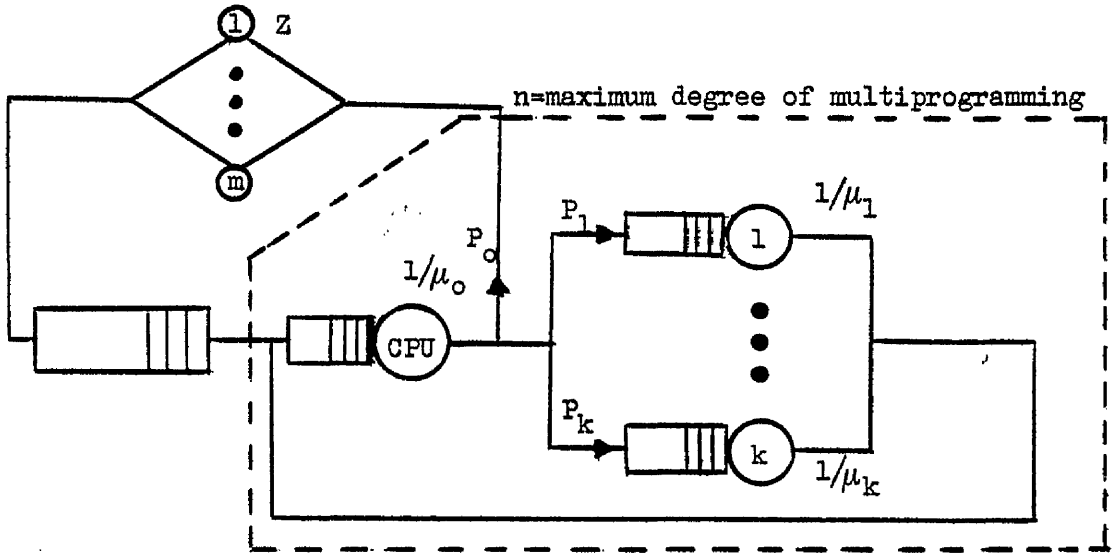


Figure 5.10. Terminal Central Server Model With Blocking (TBCSM).

5.6. Optimal Design of Terminal Computer Systems Without Blocking:

This section investigates the form and the characteristics of the response time function and its gradient for the TCSM and presents the objective function and constraints formulae for the design optimization as a non-linear minimization problem.

The TCSM falls into the category of Jackson's model and the solution technique by Kleinrock ((/Kleinrock 75/)) can be used to derive a closed form solution.

Let

* n_i = number of customers at service centre i, i = 0,..., k + 1.

$\sum_{i=0}^{k+1} n_i = m$ where m is number of terminals.

* the transition probability matrix for the TCSM is given by $P = (P_{ij})$ $(i = 0,\ldots,k+1;\ j = 0,\ldots,k+1)$
* the relative throughputs are the elements of the left eigen-vector of P, a solution of $y = yp$. The relative utilizations are given by

$$x_i = \frac{y_i}{\mu_i} \quad (i = 0,\ldots,k) \text{ and } x_{k+1} = y_{k+1} \; z \text{ respective-}$$

-ly, where y_i is the relative throughput and $\frac{1}{\mu_i}$ is the average service time for device i. z is the average think time at the terminals.
For this model $y = \left(\frac{1}{P_o}, \frac{P_1}{P_o}, \ldots \frac{Pk}{P_o}, 1 \right)$ is a left eigen-vector.
This makes $x = \left(\frac{1}{P_o\mu_o}, \frac{P_1}{P_o\mu_1}, \ldots, \frac{P_k}{P_o\mu_k}, z \right)$ the corresponding relative utilizations.
* The probability that there are n_i customers at device i $(i = 0,\ldots, k+1)$ can be expressed as :

$$P(n_o, n_1, \ldots, n_{k+1}) = \frac{1}{H_m(x)} \prod_{i=0}^{k} x_i^{n_i} \frac{x_{k+1}^{n_{k+1}}}{n_{k+1}!}$$

The last factor represents the terminal node. The sum of the n_j's $(j = 0,\ldots, k+1)$ has to be m. $H_m(x)$ is the normalization constant which ensures that all probabilities sum up to one. $H_m(x)$ is given by

$$H_m(x) = \sum_{S_m} \prod_{i=0}^{k} x_i^{n_i} \frac{x_{k+1}^{n_{k+1}}}{(n_{k+1})!}$$

where

$$S_m = \left\{ \bar{n} = (n_o, \ldots, n_{k+1}) \;\middle|\; \sum_{j=0}^{k+1} n_j = m \right\}$$

Substituting the expression which relates x_i to M_i and P_i, $H_m(x)$ becomes:

$$H_m(\mu) = \sum_{S_m} \prod_{i=0}^{k} \left(\frac{P_i}{\mu_i P_o} \right)^{n_i} \frac{z^{n_{k+1}}}{(n_{k+1})!}$$

If the relation between the mean service time $\frac{1}{\mu_i}$ and the workload parameter I_i (the average I/O service time of device i, i.e. the number of words per I/O transaction) and the device speed b_i is employed, namely

$\frac{1}{\mu_i} = \frac{I_i}{b_i}$ then the normalization constant transforms into:

$$H_m(b) = \sum_{S_m} \left(\frac{I_o}{b_o P_o}\right)^{n_o} \prod_{j=1}^{k} \left(\frac{I_j P_j}{P_o b_j}\right)^{n_j} \frac{z^{n_{k+1}}}{(n_{k+1})!}$$

* The CPU utilization (U_o) is given by:

$$U_o = x_o \frac{H_{m-1}(x)}{H_m(x)}$$

* The system throughput (i.e. the average rate of flow of programs from the CPU to the terminals) can be expressed as:

$$T(x,m) = \frac{H_{m-1}(x)}{H_m(x)}$$

* The TCSM response time is a function of throughput:

$$R(x,m) = \frac{m}{T(x,m)} - z$$

i.e.

$$R(x,m) = m \frac{H_m(x)}{H_{m-1}(x)} - z$$

i.e. R is a function of device speeds, also for any utilization vector x the corresponding device speed can be computed using the following formulae:

$$b_o = \frac{I_o}{P_o x_o} \text{ and } b_i = \frac{I_i P_i}{P_o x_i}$$

Now, the optimization problem, namely the minimization of the response time for the TCSM subject to budgetary constraints, can be specified. The decision variables are the speeds of CPU and secondary devices. The cost constraint for the TCSM is specified as follows:

$$\sum_{i=0}^{k} c_i' b_i^{\alpha_i} \leq c_{rel}$$

where

$c_{rel} = c_{tot} - c_{sys}$ (c_{tot} ≡ total budget and c_{sys} ≡ the basic system cost).

b_i = the speed of the device i.

c_i' & α_i ≡ positive constants.

(i.e. The system cost depends on component speeds).

Since device speeds and relative utilizations are related, the cost constraints transforms into:

$$\sum_{j=0}^{k} c_j \left(\frac{1}{x_j}\right)^{\alpha_j} \leq c_{rel}$$

where

$$c_o = c'_o \left(\frac{I_o}{P_o}\right)^{\alpha_o} \text{ and}$$

$$c_j = c'_j \left(\frac{I_j P_j}{P_o}\right)^{\alpha_j}, \quad j = 1,\ldots,k.$$

and is now dependent on the relative utilization vector x. The response time function can now be minimized with respect to the relative utilizations subject to the above cost constraint. m, the number of terminals, and z, the average think time at the terminals are constants. This means that any solution which minimizes the reciprocal throughput also minimizes the response time. To conclude, the TCSM optimization problem can be stated as follows:

$$\text{minimize } f(x) = \frac{H_m(x)}{H_{m-1}(x)}$$

subject to

$$c(x) = \sum_{i=0}^{k} c_i \left(\frac{1}{x_i}\right)^{\alpha_i} \leq c_{rel}$$

$$\left.\begin{array}{l} x_i > 0 \\ c_i > 0 \\ \alpha_i > 0 \end{array}\right\} \quad i = 0,1,\ldots,k.$$

This problem represents a constrained non-linear optimization problem which can be solved with any of the constrained optimization techniques available. One of them, the Lagrange multiplier method, requires the gradient of the objective function. For further information the reader is referred to ((/Von Mayrhauser 79/)).

5.7. Optimal Design of Terminal Computer Systems With Blocking:

Usually, a computer system will only allow a certain number of jobs in the subsystem consisting of the CPU and the I/O devices. All other requests for service which have not yet been allocated main memory (and thus denied access to the subsystem) have to wait outside until jobs depart and main memory becomes available for them. It is assumed that their queue is served on a 'first come first served'basis. The number of jobs in the subsystem depends on the amount of main memory available. This, in turn, is a question of budget or rather

of how much of the budget should be spent on buying main memory.

The computer system model used to investigate the blocking phenomenon was introduced in section 5.5. As for TCSM, the performance measure which is used as the objective function for the optimization problem is the system response time. The cost function of the TBCSM was augmented by a linear term which represents the cost of main memory.

Now, compared to the TCSM discussed in the previous section, the TBCSM has an additional queue between terminals and the CSM subnetwork, since admission to the CSM is restricted. No more than n jobs are allowed inside the CSM. If the terminals issue more than n requests, those which cannot enter the CSM subnetwork have to queue for admission to it.

Performance measures for this model are best computed using the technique of decomposition as in ((/Courtois 75/)). When no blocking occurs, the normalization constant is computed as:

$$H(x,m) = \sum_{i=0}^{m} \prod_{j=1}^{i} x(j) \frac{z^{m-i}}{(m-i)!}$$

where x(i) is the reciprocal CSM throughput for the CSM with degree of multiprogramming i. Moreover, x(i) is the reciprocal throughput when the terminals have issued i requests to the CSM subnetwork.

Since the CSM is able to accomodate all active requests, its reciprocal throughput is given by x(i). Now, in the terminal system with blocking, the highest degree of multiprogramming in the CSM subnetwork is n. This means that, even though more than n requests are issued, the subsystem only processes n jobs at a time and its reciprocal throughput $\hat{x}(i)$ is given by:

$$\hat{x}(i) = \begin{cases} x(i) & \text{for } i \leq n \\ x(n) & \text{for } i > n \end{cases}$$

The normalization constant for TBCSM is given by:

$$H(x,m,n) = \sum_{i=0}^{n} \prod_{j=0}^{i} x(j) \frac{z^{m-i}}{(m-i)!} + \prod_{j=1}^{n} x(j) \sum_{i=n+1}^{m} x(n)^{i-n} \frac{z^{m-i}}{(m-i)!}$$

using

$$x(j) = \frac{1}{T(j)} = \frac{G(x,j,k)}{G(x,j-1,k)} \qquad \text{and}$$

$P_i = G(x,i,k)$ the normalization constant becomes:

$$H_m(x,n) = \sum_{i=0}^{n} G(x,i,k) \frac{z^{m-i}}{(m-i)!} + G(x,n,k) \sum_{i=n+1}^{m} \frac{G(x,n,k)^{i-n}}{G(x,n-1,k)} \frac{z^{m-i}}{(m-i)!}$$

Also, applying

$U_o = x_o \frac{H_{m-1}(x)}{H_m(x)}$ (i.e. CPU utilization) and

$T(x,m) = P_o \mu_o U_o$ (i.e. system utilization)

the response time for the TBCSM is given by

$$R(x,m,n) = m \frac{H_m(x,n)}{H_{m-1}(x,n)} - z$$

Now, to study the minimization of the response time for the TBCSM subject to budgetary constraints, we need to formulate an optimization problem. The decision variables are the speeds of the CPU and the secondary devices. In addition to these, there is a new decision variable, the maximum degree of multiprogramming in the CSM subsystem.

Let c_{mem} is the cost of main memory.

If the maximum degree of multiprogramming is n, an amount of $n\, c_{mem}$ has to be spent on memory. Then, the cost constraint which will be used for TBCSM is given as follows:

$$\sum_{i=0}^{k} c_i' b_i^{\alpha_i} + c_{mem}\, n \leq c_{rel}$$

Where c_i' and α_i $(i = 0,\ldots,k)$ are the cost coefficients for the devices and b_i $(i = 0,\ldots,k)$ are the component speeds.

But since b_i and x_j are related, the cost constraint can be transformed into (where X_i means the utilization of ith device) :

$$\sum_{j=0}^{k} c_j \left(\frac{1}{x_j}\right)^{\alpha_j} + c_{mem}\, n \leq c_{rel} .$$

where

$$c_o = c_o' \left(\frac{I_o}{P_o}\right)^{\alpha_o} \text{ and}$$

$$c_j = c_j' \left(\frac{I_j P_j}{P_o}\right)^{\alpha_o}$$

Hence, the following optimization design problem can be stated:

$$\text{minimize } f_m(x,n) = \frac{H_m(x,n)}{H_{m-1}(x,n)}$$

subject to

$$c(x) + c_{mem}\, n \leqslant c_{rel}$$

with

$$c(x) = \sum_{i=0}^{k} c_i \left(\frac{1}{x_i}\right)^{\alpha_i}$$

$x_i \geqslant 0,\ c_i \geqslant 0,\ \alpha_i > 0,\ i = 0,\ldots k.$

$n > 0$ integer.

To simplify the optimization problem we may consider n as a fixed variable and the above optimization problem is reduced to:

$$\min f_m(x,n) = \frac{H_m(x,n)}{H_{m-1}(x,n)}$$

subject to

$$c(x) \leqslant c_{dev}$$

where

$$c_{dev} = c_{rel} - c_{mem}\, n$$

$$x_i > 0, \alpha_i > 0,\ c_i > 0$$

and hence it can be solved with any of the constrained optimization techniques available. Again, for more information the reader is referred to ((/Von Mayrhauser 79/)).

5.8. Further Notes:

The performance-oriented design models that have been presented in this chapter, could be developed into very useful analytic tools to aid in computer system design. This work represents a review of the research carried out by a group of researchers, mainly Trivedi, Kinicki, Von Mayrhauser, Wagner and Sigmon, at Duke University. "There are countless possibilities for extending these models to provide more realism and for developing new, more comprehensive design models"((/Sigmon 79/)). In particular, the optimal design of the storage hierarchies aid the optimal design of batch and interactive computer systems so that to maximize reliability, subject to cost and performance constraints, is possible ((/Trivedi 80/)). It is also possible to develop this method of design as an interactive design tool (i.e. to construct automated design optimizer) ((/Von Mayrhauser 79/)).

The author proposes to extend this method further in a more

simple way, using the operational analysis approach instead of the exponential queueing network (i.e. Stochastic) modelling, together with optimization theory and techniques. Such an extension will allow us to solve more complicated design problems and is suggested for future research work.

CHAPTER 6

COMPARISONS AND CONCLUSIONS

6.1. Thesis Overview.

6.2. Comparison of Methods.

6.3. Future Research Work.

6.1. Overview:

In this research work we have presented a number of ways of building system performance models. The first method was based on simulation techniques. A simulation model may model a computer system at almost any required level of detail. Many simulation models represent computer systems in considerable detail. In these cases, especially with the General Simulation Tool (GST) the greatest drawback is the relatively high cost.

A more promising alternative is to combine simulation with different modelling techniques to produce hybrid models of computer system performance. This was done by regression techniques with simulation techniques (see section 3.3.). A regression model is a fast statistical model of computer system performance which relies on workload and performance data collected from the system being evaluated ((/Grenander and Tsao 72/)). However, it has the disadvantage of not being capable of modelling logical and structural relationships in the system. Simulation does not suffer from this limitation, but a simulation model which produced results similar to a regression model would probably need to model the system in considerably more detail, and consequently be more expensive to implement. By combining simulation and regression techniques, the advantages of both may be exploited. The regression models were constructed using simple case studies. These case studies will produce a relation matrix in which all the performance parameters/ indices equations are contained. These equations are quite simple and may be used to construct an interactive design tool (IDT).

The second method of building system performance models was based on operational analysis techniques. These techniques were selected from different available approaches, such as stochastic modelling and mean-value analysis. The selection was done according to certain factors, such as:

* Understandability.
* Cost.
* Degree of resolution.
* Ease of parameter optimization or estimation.
* Breadth of applicability.
* Relevance to the actual system.

Operational analysis is based on the premise of testability. All the basic performance quantities - utilization, completion rates,

mean queue size, mean response time, load distributions - are defined as they would be in practice, from data taken over a finite period. The analyst can test whether the basic assumptions - flow balance, one-step behaviour, and homogeneity - hold in any observation period. The operational laws are identities among operational quantities. They are a consistency check - a failure to satisfy an operational law reveals an error in the data. They simplify data collection by showing alternatives for computing performance quantities ((/Denning and Buzen 78/)).

In practice, errors from these assumptions are not serious. Even when the additional assumption of homogeneous service time is used to simplify the analysis further, these models estimate utilizations, throughputs and system response times typically within 10 %, and mean queue lengths and device response times typically to within 30 % ((/Giammo 76/)).

Using the operational analysis approach we tried to represent the behaviour of a general interactive computer system. This approach can be combined with the simulation (using GST) approach to produce other hybrid simulation/operational analysis models (see the following section). Furthermore, in the operational general model of the interactive computer systems we have tried to show the effects of the changes of some system software.

Performance-oriented design was the third method of building system performance models. This method has been introduced, due to the shortcomings concerning the ability of parameter estimation or optimization. It is in the realm of inductive mathematics, whereas operational analysis is a branch of deductive mathematics ((/Denning and Buzen 78/)). With respect to simulation models, the existing models are both too costly, and inadequate to solve the apparently simple problem of optimum system configuration ((/Nielsen 67/)). The performance-oriented design method solved this problem using optimization theory and techniques. Several optimization and design problems have been introduced to minimize the response time or maximize the system throughput of the modelled system subject to a cost constraint.

Our experience with the above methods has shown that there is no single best way to design a computer system. A 'good' computer system designer must creatively deal with the problems of the intended

system users, the problems of technology from which the system will be built, and the problems of the people who will implement his design. It is an artistic blend of theory, engineering and pragmatism which will allow him to produce a system which meets the functional, performance and cost specifications from which he began.

This is not an easy task and the basic conclusion of this thesis is that all three methods presented above should be combined in such a way as to help the designer in building computer performance models. An implementation of this idea is given in the following sections.

6.2. Comparison of Methods:

The performance evaluation and design methods presented have largely been compared by introducing each of these models serparately. One important concept must, however, still be discussed. This is the validation and prediction of the models produced by the above methods. Validation refers to extensive testing of a model to determine its accuracy in calculating performance measures. Prediction refers to using the validated model to calculate performance measures for a time period (usually in the future) in which the values of parameters required by the model are uncertain.

These will be examined in the following, using the simulation/regression models as prediction models and the interactive operational analysis models and interactive performance-oriented design models as validation models. The prediction models will be compared with the validation models and the results will be plotted on a graph which will show how far the results of the validation (i.e. operational models or performance-oriented design) models differ from the very detailed results produced by the prediction (i.e. simulation/regression) models. In other words, the results of the simulation/regression models which have been argued to be realistic and correct ((/Cavouras 78/)) will be used to validate the results produced by the operational analysis models and the performance-oriented design models. Figure 6.1. illustrates the steps followed in our validation and prediction scheme.

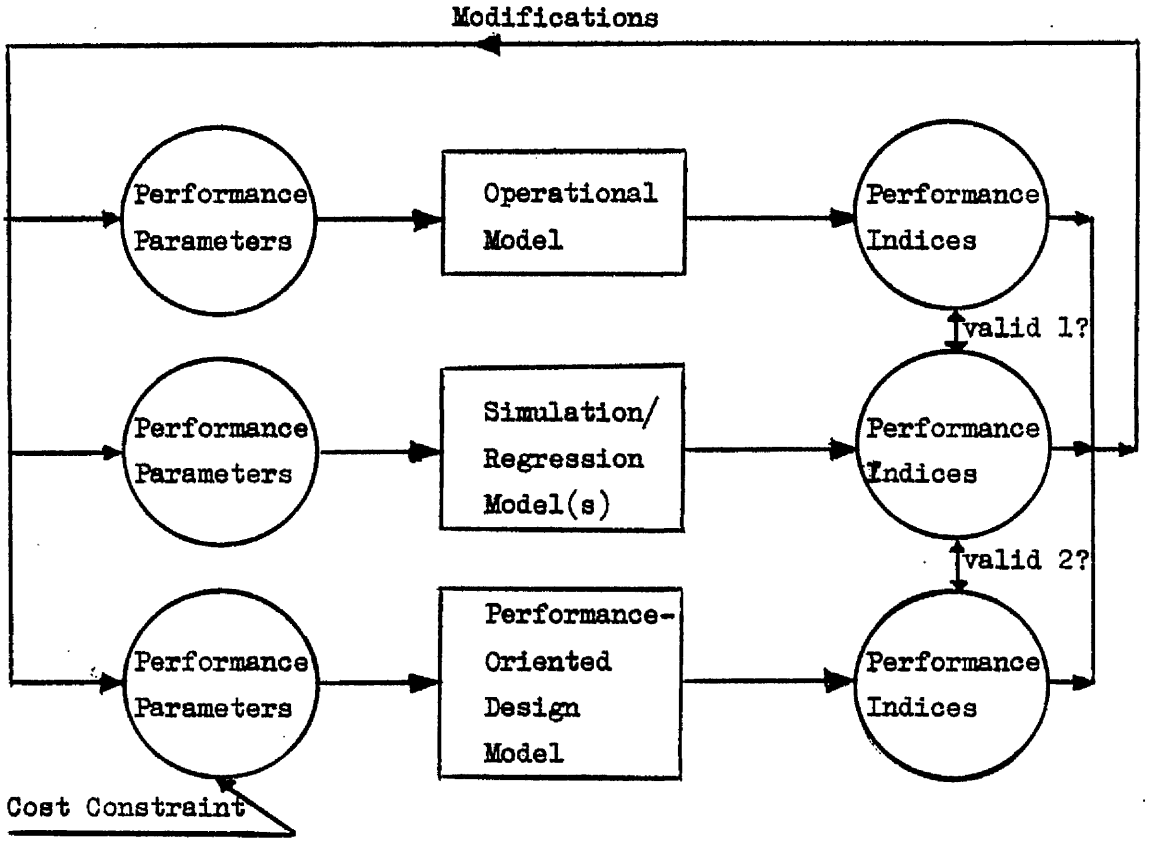


Figure 6.1: Performance validation - Prediction scheme of the presented evaluation and design methods.

First, the analyst runs the selected Performance Parameters on the simulation/regression models (the prediction models). He then collects (measures or calculates) the performance quantities such as throughput and response time and also the parameters of the devices. After that, the analyst applies the same selected performance parameters to the operational model and compares the results against the collected performance quantities. If, over many different observation periods, the computed values compare well with the collected values, the analyst will come to believe that the operational model is good. Thereafter, he will employ it confidently for predicting future behaviour and for evaluating proposed changes in the system being designed. Similarly, the collected performance quantities can be compared with the results obtained from the performance-oriented models using the same selected performance parameters and different cost constraints.

The comparison of the different models is done concurrently with their modification and analysis. In this, new assumptions can be

added to the modelled system (using operational analysis or performance-oriented design techniques). Such assumptions typically include that device and workload parameters do not change unless they are explicitly modified. Though such assumptions are usually satisfactory, they can lead to trouble if a given change has side effects, for example, increasing the number of time-sharing terminals may unexpectedly reduce the batch multiprogramming level even though the batch workload is the same.

Based on the comparisons and modifications, several different case studies can be carried out in order to analyse the three methods - simulation/regression, operational analysis and performance-oriented design - using the same supplied performance parameters and different total cost values. These case studies will be called Prediction-Validation examples and include the following:

* Response time vs. No. of terminals Prediction - Validation example.
* Response time vs. think time Prediction - Validation example.
* Response time vs. devices speeds Prediction - Validation example.
* Response time vs. degree of multiprogramming Prediction - Validation example.

In order to demonstrate the approach, one example will be studied in the following section. To analyse the other examples would require repeating the same procedure for a number of sets of given performance parameters.

Prediction-Validation Example.

In this example we will study the effects of changing the maximum no. of terminals upon the system response time, using the three methods introduced in the previous chapters. Figure 6.2. shows the graphs produced by these methods using the same selected performance parameters given by Cavouras ((/Cavouras 78/)). (see Table 6.1.)

The simulation/regression model graph has been plotted using the following equation. For further details the reader is referred to case study no. 1 chapter 3.

$$R = a \cdot e^{bM}$$

where:

R represents the average response time.

M represents the no. of terminals.

a,b are positive constants:

$$a = 5.00$$

$$b = 0.02$$

The operational model graph has been plotted using:

$$\boxed{R = \frac{M}{X - Z}}$$

where:

$$X = \frac{M}{Z} \quad \frac{h(M-1,K)}{h(M,K)}$$

M = no. of terminals.

K = no. of devices. (= 3)

Z = average thinking time. (= 30 seconds)

h(. , .) = a normalization factor calculated using the algorithm of Williams and Bhandiwad ((/Williams and Bhandiwad 76/)).

Finally, the performance-oriented design graphs have been plotted using the following optimization problem (see Von Mayhauser 79):

$$\min \quad f_m (x,n) = \frac{H_m (x,n)}{H_{m-1}(x,n)}$$

subject to

$$C(x) \leqslant C_{dev}$$

where

$$C_{dev} = C_{rel} - C_{mem} * n$$

$$X_i > 0 \quad *_i > 0 \ , \ C_i > 0.$$

The three graphs show many differences. An important one is that the values of the response time produced by the simulation/ regression model are higher than the values (at most) of both the operational model and the performance-oriented design model. The reason for this difference is due to the fact that the simulation/ regression model takes into account the overhead time spent in the system. It is also clear from the performance-oriented design model that increasing the total cost spent, will yield better response time.

Due to the mathematical structure of both the operational and performance-oriented models, hardware effects (such as the maximum number of terminals) or software effects (such as the degree of multiprogramming) can be easily computed. This is of great importance

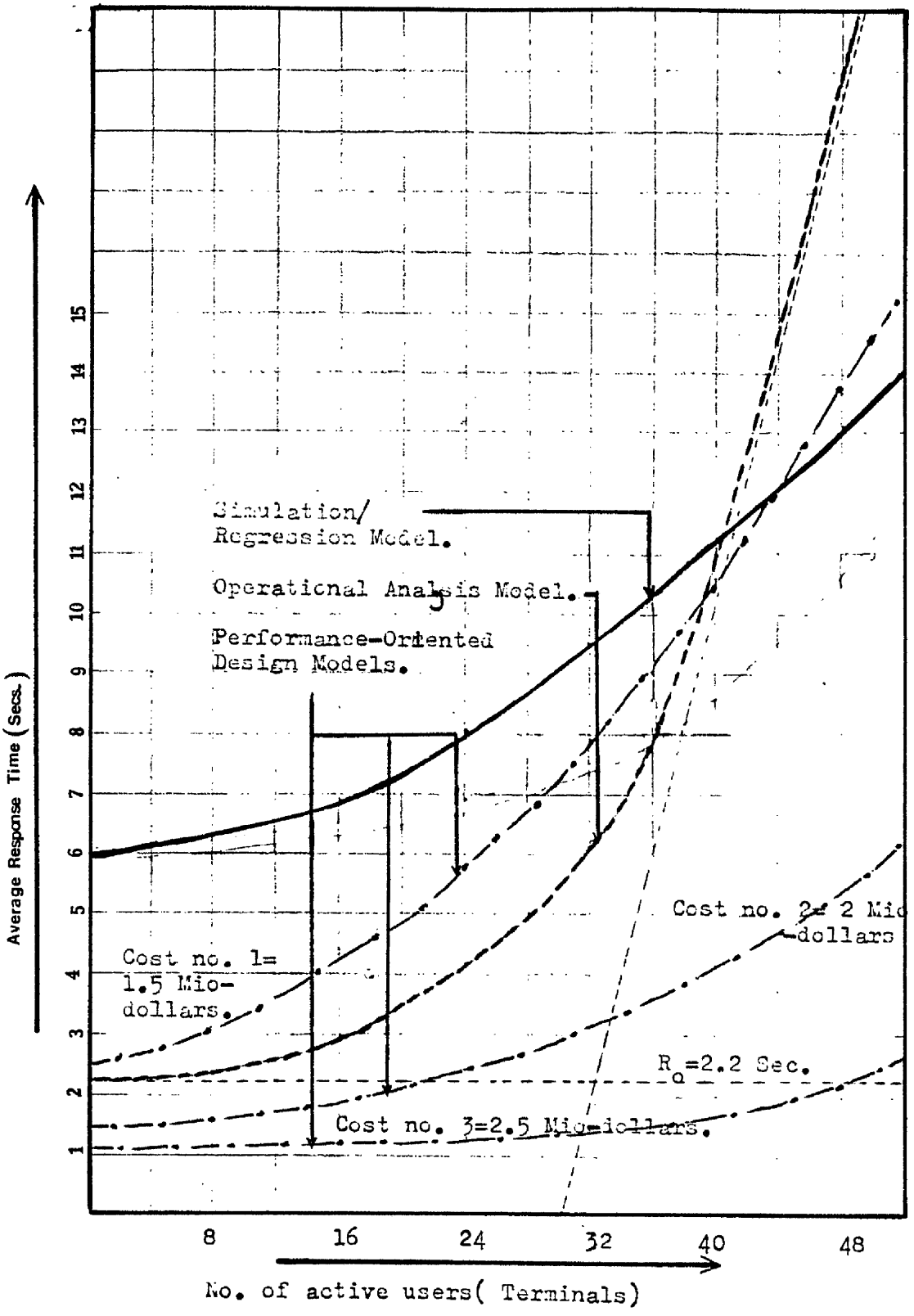


Figure 6.2. : Response time Vs. No. of active users (Terminals) prediction-validation graph .

Number of Active Users	Response Time Simulation /Regression Method	Response Time of Performance-Oriented Design Method			Response Time of Operational Analysis Method
		Cost 1	Cost 2	Cost 3	
16	6.89	4.21	1.91	1.12	2.96
24	8.08	5.81	2.23	1.21	4.13
32	9.48	7.82	3.19	1.32	6.12
40	11.13	10.47	4.12	1.62	10.98
48	13.06	13.82	5.32	2.22	18.87

Table 6.1: Response Time vs. no. of Active Users (Terminals) under the Three Performance Evaluation Methods.

for studying and analysing the behaviour of the system according to the parameters changes. It will provide the designer with more speed and more information about the best system design which can be produced within a certain cost limit.

To conclude, the idea of comparing different evaluation methods always provides better information about the computer system required to be designed.

6.3. Future Research Work:

Throughout this thesis we have suggested several areas of possible refinements and extensions to this work. The possible areas are virtually unlimited. Possible topics include the following:

* The effects of different scheduling disciplines on the important performance parameters. This problem can be studied in detail using the simulation technique. Specifically, we can use GST to study several policy functions ((see/Cavouras 78/)). This problem can also be studied using performance-oriented design techniques. For this purpose we suggest to generalize the work of Mahl ((/Mahl 70/)), Badel and Leroudier ((/Badel and Leroudier 78/)) and Gotlib and Schonbach ((/Gotlib and Schonbach 80/)). The Mahl approach depends only upon the optimization technique. He defines an economic function which can be maximized by selecting a certain set of jobs to enter the main memory (i.e. the set of active jobs) depending on a specific scheduling discipline. The approach of both Badel and Leroudier, and Gotlib and Schonbach is a simulation approach.

  Similarly, the effects of the scheduling disciplines can be studied using the operational analysis technique. For this purpose we suggest to use the idea of Brad ((/Brad 77/)) as the base for that analysis.

  Again, the results of modelling the scheduling disciplines derived from the three methods can be compared and analysed for further design and evaluation.
* Studying the effects of different designs of the storage subsystem, using the three design and evaluation methods.

For this purpose, we suggest to modify the general simulation model (GST) in order to implement different storage subsystem designs. For the performance-oriented design method we suggest to modify the research work of ((/Trivedi and Sigmon 81/)), ((/Chow 74/)), ((/Gecsei and Lukes 74/)) and ((/Ramamoorthy and Chandy 70/)). The results obtained after implementing different storage subsystem designs, using the three different methods, can be compared for more information.

* Studying the errors that are due to the approximation methods or assumptions which have been used to produce both the operational models and performance-oriented design models. This type of analysis is called sensitivity analysis ((/Buzen and Denning 80/)).
* Further studies in models' validity. For this purpose we suggest to use the measurement techniques on an actual system. This idea may involve constructing a sampling software monitor. The results of this monitor will be used to validate the models produced by the three design and evaluation methods. We may also use the research of ((/Kumar 80/)) as the base of this work.
* Constructing a general interactive design tool (GIDT) based upon the three design and evaluation methods. This tool should include graphical facilities. The abstract idea of such a tool is given in Figure 6.3. The idea involves constructing three interactive design tools and a selection procedure. The selection will be based on the advantages of each particular design tool, for a given design problem. For the purpose of constructing the GIDT, we may use the BEST1 design tool introduced by Buzen ((/Buzen, Goldberg, Langer, Lentz, Schwenk, Sheets and Shum 78/)).
* Further investigation to add new powerful mathematical structures to operational analysis. This idea was originally started by Bouhana ((/Bouhana 78/)) in which he implemented the theory of matrices within operational analysis.
* Studying the effects of program behaviour using the three design and evaluation methods. Some necessary modifications should be added to these methods. For the operational analysis part we may use the work of ((/Denning 80/)) and ((/Slutz and Traiger 74/)) as a base for these modifications.
* Further studies should be carried out to implement the user

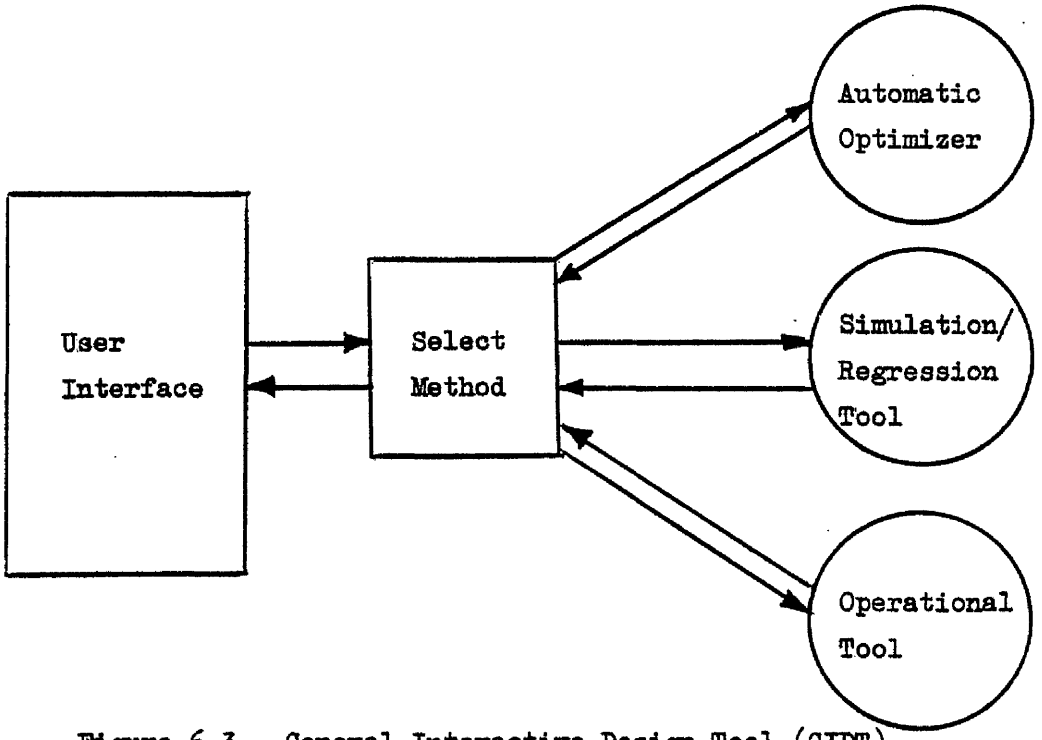


Figure 6.3. General Interactive Design Tool (GIDT).

effects, especially with the interactive computer system models. For this purpose, new performance parameters should be added. Examples of user effects include their productivity and satisfaction. For the purpose of implementation we might be able to use the work of ((/Barber 79/)).

* Using the performance-oriented design technique we aim to represent a general interactive computer system, based on the idea of multi-customer classes. We also aim to represent in such a system the cost of each component as a function of their characteristic parameters. Finally, we may increase the resolution power of the workload in that system.

It is believed that in this work the basic framework of computer design and evaluation techniques has been provided. This is but a start in a relatively new area. There are countless possibilities for extending these methods to provide more realism and for developing new, more comprehensive design models. The path to further knowledge awaits our exploration.

REFERENCES

/Adams 78/: J.C. Adams "Performance Measurements and Evaluation of Time-Shared Operating Systems" Ph.D. thesis, University of Edinburgh 1978.

/Adby and Dempster 74/: P.R. Adby and M.A.H. Dempster "Introduction to Optimization Methods" John Wiley and Sons, N.Y. 1974.

/Badel and Leroudier 78/: M. Badel and J. Leroudier "Optimal Multi-Programming: Principle and Implementation" IRIA, Laboria, Le Chesnay, France, report no. 276, 1978.

/Baird 62/: "Experimentation" Prentice-Hall 1962.

/Balbo and Denning 79/: G. Balbo and J.P. Denning "Homogenous Approximations of General Queueing Networks" 4th Int. Symposium of Modelling and Performance Evaluation of Computer Systems, Prepints, Vienna, Austria, Feb. 1979, PP.333-354.

/Barber 79/: R.E. Barber "Response Time, Operator Productivity and Job Satisfaction" Ph.D. Diss. New York University 1979.

/Baskett, Chandy, Muntz and Palacois 75/: F. Baskett, K. Chandy, R. Muntz and J. Palacois "Open, Closed and Mixed Networks with Classes of Customers" JACM, vol.22, no.2, April 1975, PP.248-260.

/Bayer, Graham and Seegmuller 78/: R. Bayer, R.M. Graham and G. Seegmuller "Operating Systems:Advance Course" Springer-Veriage N.Y., 1978.

/Benwell 75/: N Benwell (ed.) "Benchmarking" Halsted Press Wiley, N.Y. 1975.

/Bock,Yancy and Judge 73/: M.E. Bock, T.A. Yancy and G.G. Judge "The Statistical Consequence of Preliminary Test Estimators in Regression" The J. of American Statistical Assoc. No.68,March 73, PP.109-116.

/Bose and Warn 75/: J.W. Bose and D.R. Warn "A Straightforward Model for Computer Performance Prediction" ACM Computing Surveys, vol. 7, no. 2, 1975, PP.73-93.

/Bouhana 78/: J. Bouhana "Operational Aspects of Centralized Queueing Networks" Ph.D. thesis, University of Wisconsin-Madison 1978.

/Box and Jenkins 70/: G.E.P. Box and G.M. Jenkins "Time Series Analysis : Forecasting and Control" San Francisco, Holden-Day Inc. 1970.

/Brad 71/: Y. Brad "Performance Criteria and Measurements for a Time Sharing System" IBM Sys. J. 10,1971, PP.193-219.

/Brad 77/: Y. Brad "The Modelling of some Scheduling Strategies for an Interactive Computer System" Proc. of Int. Symposium on Computer Performance Modelling, Measurements and Evaluation, Yorktown, Heights, N.Y., Aug.1977, PP.113-137.

/Bryant 79/: R. Bryant "A Critique of Operational Analysis of Computer Systems" Workshop on the Theory and Application of Analytical Models to ADP Systems Prediction, University of Maryland, March 1979.

/Buzen 71/: J.P. Buzen "Queueing Network Models of Multiprogramming" Ph.D. thesis, Dev. Eng. and Applied Physics, Harvard University, May 1971.

/Buzen 73/: J.P. Buzen "Computational Algorithms for Closed Queueing Networks with Exponential Servers" CACM 16, 9, Sept. 1973 PP.527-531.

/Buzen 76/: J.P. Buzen "Fundamental Operational Laws of Computer System Performance" Acta-Informatica, 7, 1976, PP.167-182.

/Buzen 76a/: J.P. Buzen "Operational Analysis: the Key to the New Generation of Performance Prediction Tools" Proc. IEEE COMPCON, 1976, PP.166-171.

/Buzen 77/: J.P. Buzen "Principles of Computer Performance Modelling and Prediction" State of Art Report on Performance Modelling and Prediction, Infotech, Ltd. U.K. 1977, PP. 3-18.

/Buzen 78/: J.P. Buzen "Operational Analysis: an Alternative to Stochastic Modelling" Proc. Int. Conf. Performance of Computer Installations. 1978, PP.175-194.

/Buzen, Goldberg, Langer, Lentz, Schwenk, Sheets and Shum 78/: J.Buzen, R. Goldberg, A. Langer, E. Lentz, H. Schwenk, D. Sheets and A. Shum "Best 1- Design of a Tool for Computing System Capacity Planning" National Computer Conference, 1978, PP. 447-455.

/Buzen 79/: J.P. Buzen "The Predictable Problem" Computing Surveys, Vol.11, no.1, March 1979, PP. 70-72.

/Buzen and Denning 80/: J.P. Buzen and P.J. Denning "Operational Treatment of Queue Distributions and Mean-Value Analysis" Computer Performance, vol.1, no.1, PP. 6-15. IPC.

/Blytheway 80/: A.J. Blytheway "On the Proper Treatment of Measurement Data" Computer Performance, Vol. 1.,no.1., PP.28-36.

/Calingaert 67/: P. Calingaert "System Performance Evaluation: Survey and Appraisal" CACM 10, January 1967, PP.12-18.

/Cantrell and Ellison 68/: H.N. Cantrell and A.L. Ellison "Multi-Programming System Performance and Analysis" AFIPS, 1968, PP.213-221.

/Cavouras 78/: J.C. Cavouras "The Development and Application of a Method for Producing Software Tools for Computing System Design" Ph.D. thesis, Glasgow University, 1978.

/Cavouras and Davis 81/: J.C. Cavouras and R.H. Davis "Simulation Tools in Computer System Design Methodologies" BCS Computer Journal, vol.24, no.1, 1981.

/Chandy, Howard and Towsley 77/: K.M. Chandy, J.H. Howard and D.F. Towsley "Product Form and Local Balance in Queueing Networks" JACM 24, 2, April 1977, PP.256-263.

/Chandy and Sauer 78/: K.M.Chandy and C.H. Sauer "Approximate Methods for Analysing Queueing Network Models of Computing Systems" ACM Computing Surveys, vol. 10, no.3, Sept. 1978, PP.281-317.

/Chouinard 76/: P.L. Chouinard "The Statistical Estimation of Throughput and Turnaround Functions for a University Computer System" Ph.D. Diss. University of Illinois 76.

/Chow 74/: C.K. Chow "On Optimization of Storage Hierarchies" IBM Journal of Research & Development, May 1974, PP.194-203.

/Coffman and Matrani 75/: E.G. Coffman Jnr. and I. Mitrani "Selecting a Scheduling Rule that meets Aprespecified Response Time Demands" Proc. ACM SIGOPS. 1975, PP. 187-191.

/Courtois 75/: P.J. Courtois "Decomposibility, Instabilities and Saturation in Multiprogramming Systems" CACM vol.18, no.7, July 1975, PP.371-377.

/Cox 58/: D.R. Cox "Planning of Experiments" Wiley, 1958.

/Decegama 70/: A.L. Decegama "Performance Optimization of Multiprogra-mming Systems" Ph.D. thesis, Carnegie)Mellon University 1970.

/Decegama 72/: A.L. Decegama "A Methodology for Computer Model Building" AFIPS, 1972, PP.299-310.

/Denning 70/: P.J. Denning "Virtual Memory" ACM Computing Surveys, Vol.2, no.3, PP.153-189.

/Denning and Buzen 77/: P.J. Denning and J.P. Buzen "An Operational Overview of Queueing Networks" in Infotech State of Art Report on Performance Modelling and Prediction, U.K. 1977, PP.75-108.

/Denning and Buzen 78/: P.J. Denning and J.P. Buzen "The Operational Analysis of Queueing Network Models" ACM Computing Surveys, vol. 10, no.3, Sept. 1978, PP. 225-261.

/Denning and Denning 79/: P.J. Denning and D.E. Denning "Operational Analysis of Data Sampling Problem" Tech. Report no. CSD-TR-302, Purdue University, April 1979.

/Denning 80/: P.J. Denning "Working Sets Past and Present" IEEE Trans on Software Engineering, vol. SE6,no.1, Jan.1980 PP. 64-84.

/Estrin and Kleinrock 67/: G. Estrin and L. Kleinrock "Measures, Models and Measurements of Time-Shared Computer Utilities" Proc. ACM National Meeting 1967 PP.85-95.

/Fenichel and Grossman 69/: R.R. Fenichel and A.J. Grossman "An Analytic Model of Multiprogramming Model" AFIPS 1969, PP. 717-721.

/Ferrari 78/: D. Ferrari "Computer Systems Performance Evaluation" Prentice-Hall, Inc., 1978.

/Gaver 67/: D. Gaver "Probability Models for Multiprogramming Computer Systems" JACM, vol.14, no.3, July 1976, PP.423-438.

/Gecsei and Lukes 74/: J. Gecsei and A. Lukes "A Model for Evaluation of Storage Hierarchies" IBM Syst. J., no.2,1974 PP.163-178.

/Giammo 76/: T. Giammo "Validation of a Computer Performance Model of the Exponential Queueing Network Family" Acta -Informatica 7, 2,1976, PP.137-152.

/Goh 76/: T.N. Goh "Computer System Simulation" m.Sc. thesis, University of Glasgow 1976.

/Gomma 76/: H. Gomma "A Modelling Approach to the Evaluation of Computer System Performance" Ph.D. thesis, Imperial College of Science and Technology, 1976.

/Gotlieb and Schonbach 80/: C.C. Gotlieb and A. Schonbach "System-Oriented Macro-Scheduling" Technical Report no. CSRG-113, University of Toronto, May 1980.

/Gottfried and Weisman 73/: B.S. Gottfried and J. Weisman "Introduction to Optimization Theory" Prentice-Hall, Englewood Cliffs, New Jersey, 1973.

/Graham 78/: B.S. Graham "Queueing Network Models of Computer System Performance" ACM Computing Surveys 10,3, Sept. 1978, PP.219-224.

/Grenander and Tsao 72/: V. Grenander and R. Tsao "Quantative Methods for Evaluating Computer System Performance: A Review and Proposals" Statistical Computer Performance Evaluation, W. Freiberger (ed.) Academic Press 1972, PP.3-24.

/Gupta and Verma 80/: P.K. Gupta and R.K. Verma "Optimal Resource Allocation in a Complex Queueing System" Math. Operations forsh. Statist., Ser. Optimization, Vol.11, 1980, no.3,PP.507-512.

/Hansen 73/: P.B. Hansen "Operating Systems Principles" Prentice-Hall Inc., N.Y. 1973.

/Hellerman and Conroy/: H. Hellerman and T.F. Conroy "Computer System Performance" McGraw-Hill, N.Y. 1975.

/Hofri 79/: M. Hofri "On Modelling for Performance Evaluation of Computing Systems Operational Analysis" Dept. of Computing Science, Technion 1979.

/Hughes and Moe 73/: P.H. Hughes and G.A. Moe "A Structural Approach to Computer Performance Analysis" AFIPS 1973, PP.109-119.

/Ihrer 67/: F.C. Ihrer "Computer Performance Projected Through Simulation" Computer & Automation, April 1967, PP.24-28.

/Irani and Uppal 72/: K.B. Irani and I.S. Uppal "Performance Analysis and Optimization of Computer Systems Using Markovian Models" First USA-JAPAN Computer Conference 1972, PP.145-152.

/Jackson 57/: J.R. Jackson "Networks of Waiting Lines" Oper. Res. No. 5, 1957, PP.518-521.

/Jackson 63/: J.R. Jackson "Jobshop Like Queueing Systems" Manage. Sci., no.10, 1963, PP.131-142.

/Kachhal 74/: S.K. Kachhal "Configurational Optimization of Computer System" Ph.D. thesis, University of Minnesota, 1974.

/Kempethorne 52/: O. Kempthorne "The Design and Analysis Experiments" J. Wiley, 1952.

/Kernighan and Ritchie 78/: B.W. Kernighan and D.M. Ritchie "The C Programming Language" Prentice-Hall, N.J. 1978.

/Kienzle and Sevcik 79/: M.G. Kienzle and K.C. Sevcik "Survey of Analytic Queueing Network Models of Computer Systems" Conf. on Simulation, Measurement and Modelling of Comp. System aug. 1979. Boulder, Colorado, PP.113-129.

/Kinicki 78/: R.E. Kinicki "Queueing Models for Computer System Configuration Planning" Ph.D. Diss. Duke University 1978.

/Kimbleton 72/: S.R. Kimbleton "Performance Evaluation-A Structured Approach" AFIPS 1972, PP.411-416.

/Kleinrock 75/: L. Kleinrock "Queueing Systems: Theory" Vol.1, J. Wiley 1975.

/Kleinrock 76/: L. Kleinrock "Queueing Systems: Computer Applications" Vol. 2, J. Wiley 1976.

/Koffman 69/: E.G. Koffman "Analysis of a Drum I/O Queue Under Scheduled Operation in a Paged Computer System" JACM 1969, Vol.16, no.1, PP.73-90.

/Kobayashi 78/: H. Kobayashi "Modelling and Analysis: An Introduction to System Performance Evaluation Methodology" Addison-Wesley, 1978.

/Kumar and Gonsalves 79/: B. Kumar and T.A. Gonsalves "Modelling and Analysis of Distributed Software Systems" Proc. 7th Symposium on Operating Systems Principles, Dec.1979, PP.2-8.

/Kumar 80/: S.R. Kumar "An I/O Resource Allocation Methodology for Performance Evaluation and Enhancement Studies of Interactive Operating Systems" Ph.D. thesis Case Western Reserve University, 1980.

/Landy 71/: B. Landy "Development of Scheduling Strategies in the TITAN Operating System" Software Practice and Experience, Vol. 1, PP.279-295, 1975.

/Lister 75/: A.M. Lister "Fundamentals of Operating Systems" Macmillan Press ltd. 1975.

/Luenberger 73/: D.G. Luenberger "Introduction to Linear and Non-Linear Programming" Addison-Wesley, Reading, Mass. 1973.

/Mahl 70/: R. Mahl "An analytic Approach to Computer Systems Scheduling" Ph.D. thesis, University of Utah 1970.

/McKinney 69/: M. McKinney "A Survey of Analytical Time-Sharing Models" ACM Computing Surveys, vol.1, no.2, June 1969, PP.105-116.

/Mirham 72/: G.A. Mirham "Simulation: Statistical Foundations and Methodology" Academic Press, N.Y. 1972.

/Murdoch 78/: J. Murdoch "Queueing Theory: Worked Examples and Problems" Macmillan 1978.

/Muntz 75/: R.R. Muntz "Analytic Modelling of Interactive Systems" Proc. IEEE, Vol.63, No.6, June 1975, PP.946-953.

/Muntz 79/: R.R. Muntz "A Predictable Problem" ACM Computing Surveys Vol.11, no.1, March 1979, PP.70-72.

/Nielson 67/: N.R. Nielson "The Simulation of Time-Sharing Systems" CACM, July 1967.

/Noetzel 71/: A.S. Noetzel "The Design of a Meta-System" AFIPS 1971, PP.415-424.

/Pujelle and Soula 79/: G. Pujolle and C. Soula "A Study of Flows in Queueing Networks and an Approximate Method for Solution" Atrato, Butrimenko and Gelenbe (eds.) Int. Institute for Applied System Analysis, North-Holland Pub. Co., 1979, PP.375-389.

/Rakoczi 69/: L.L. Rakoczi "The Computer-Within-a Computer A Fourth Generation Concept" Computer Group News, March 1969.

/Ramamoorthy and Chandy 70/:C.U. Ramamoorthy and K.M. Chandy "Optimization of Memory Hierarchies in Multiprogramming Systems" JACM, Vol.17, no.3, July 1970, PP. 426-445.

/Riser 79/: M. Riser "Mean-Value Analysis of Queueing Networks: A New Look at Old Problem" Proc. 4th Int. Symposium on Modelling and Performance Evaluation of Computer Systems, Vienna Feb. 1979, PP. 63-77.

/Riser and Lavenberg 80/: M. Riser and S.S. Lavenberg "Mean-Value Analysis of Closed Multichain Queueing Networks" JACM, Vol. 27, no.2, April 1980, PP.313-322.

/Riser 81/: M. Riser "Mean-Value Analysis and Convolution Methods for Queueing Dependent Servers in Closed Networks" Performance Evaluation: an International Journal, Vol.1, no. 1, 1981.

/Roode 79/: J. Roode "Multiclass Operational Analysis of Queueing Networks" Proc. of 4th Int. Symposium on Computer Performance Measurement, Modelling and Evaluation, Vienna

of queueing network states at input and output instants" Research Report no.307, IRIA Recquencourt, France 1978.
(Feb. 1979), PP. 339-352.

/Sekino 72/: A. Sekino "Performance Evaluation of Multiprogrammed Time-Shared Computer Systems" Ph.D. thesis, Mass. Institute of Technology, 1972.

/Sevcik and Klawe 79/: K.C. Sevcik and M.M. Klawe "Operational Analysis Versus Stochastic Modelling of Computer Systems" 12 Annual Symposium on the Interface of Computer Science and Statistic, University of Waterloo, 1979.

/Shannon 75/: R.E. Shannon "System Simulation the Art and Science" Prentice-Hall 1975.

/Sigmon 79/: T.M. Sigmon "Performance-Oriented Design Models for Computer Systems" Ph.D. Diss. Duke University 1979.

/Sime 73/: J.G. Sime "Benchmark for the ICL 19065 Computer Systems" Glasgow University, Computing Service, June 1973.

/Slutz and Traiger 74/: D.R. Slutz and I.L. Traiger "A Note on the Calculation of Average Working Set Size" CACM Vol.17, Oct. 1974, PP.563-565.

/Sprangins 79/: J. Sprangins "Approximate Techniques for Modelling the Performance of Complex Systems" Computer Languages Vol.4, PP.91-129, 1979.

/Svobodova 79/: L. Svobodova "Computer Performance Measurements and Evaluation Methods: Analysis and Applications" American Elsevier Publ. Co., 1976.

/Takacs 63/: L. Takacs "A Single Server Queue with Feedback" Bell-System Journal 42, 1963, PP.505-519.

/Trivedi and Kinicki 78/: K.S. Trivedi and R.E. Kinicki "Mathematical Model for Computer System Configuration Planning" Performance of Computer Installations, D. Ferrari (ed.) North-Holland, Amsterdam 1978.

/Trivedi and Wagner 79/: K.S. Trivedi and R.A. Wagner "A Decision Model for Closed Queueing Networks" IEEE Transaction on Software Engineering, SE-5,4, July 1979, PP.328-332.

/Trivedi 80/: K.S. Trivedi "Designing Linear Storage Hierarchies so as to Minimize Realibility Subject to Cost and Performance Constraints" Conf. Proc. of the 7th Annual Symposium on Computer Architecture, LA BAULE, May 1980, PP.211-217.

/Trivedi and Sigmon 81/: K.S. Trivedi and T.M. Sigmon "Optimal Design of Linear Storage Hierarchies" JACM, Vol.28, no.2, April 1981, PP.270-288.

/Von Mayrhauser 79/: A.E.K. Von Mayrhauser "Performance-Oriented

Design of Interactive Computer Systems" Ph.D. Diss. Duke University, 1979

/Walsh 75/: G.R. Walsh "Methods of Optimization" J. Wiley and Sons, N.Y. 1975.

/Watson 70/: R.W. Watson "Timesharing System Design Concepts" McGraw Hill 1970.

/Wettstein and Merbeth 80/: H. Wettstein and G. Merbeth "The Concept of Asynchronization" ACM Operating System Review, Vol.14, no.4, Oct. 1980.

/Williams 72/: T. Williams "Computer Systems Measurements and Evaluation" BSC The Computer Bulletin, no.16, Feb. 1972, PP.100-104.

/Williams and Bhandiwad 76/: A.C. Williams and R.A. Bhandiwad "A Generating Function Approach to Queueing Network Analysis of Multiprogrammed Computers" Networks,6,1, 1976, PP. 1-22.

APPENDICES.

Appendix A1 : Some Helpful Statistical Methods.

Appendix A2 : Abbreviation.

Appendix B1 : The Representation of a Mutiolass Customer Closed Queueing Network.

Appendix B2 : The Algorithm of Calculating the Normalization Factor of Interactive Computer Systems Models.

Appendix A1

Some Helpful Statistical Methods.

1. Regression Methods:

The techniques of regression and correlation analysis are very useful. Regression analysis takes a set of data and fits it to an equation whose form is pre-selected by the analyst. Correlation analysis gives us some indication of how well the data points fit or cluster around the equation so derived. It is recalled that one purpose of statistics is to represent many numbers by a few numbers. One way to do this is to fit an analytic function like a polynomial to the data. Once this is done, only a few coefficients of the polynomial need to be stored to represent the data. The problem of finding equations for the approximating curves that best fit the given sets of data is called curve fitting. The analyst must select the curve to be fitted. For reference, we list several common types of approximating curves and their equations. All letters other than x and y are constants or parameters and x is the independent variable ((/Shannon 75/)).

1.	$y = a_o + a_1 x$	Straight line.
2.	$y = a_1 x + a_2 x^2$	parabola or quadratic curve.
3.	$y = a_o + a_1 x + a_2 x^2 + a_3 x^3$	cubic curve.
4.	$y = a_o + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$	quartic curve.
5.	$y = 1/(a_o + a_1 x)$ or $\frac{1}{y} = a_o + a_1 x$	hyperbola.
6.	$y = ab^x$ or $\log y = a_o + a_1 x$	exponential curve.
7.	$y = a_o + a_1 \log x$	logarithmic.
8.	$\log y = a_o + a_1 \log x$	cubic logarithm.
⋮	⋮	⋮
etc.	etc.	etc.

To decide which to use, we can examine scatter diagrams and compare results with the general shapes to the curve given by different equations; Figure A1.1 represents the best fit that can be chosen is the linear curve.

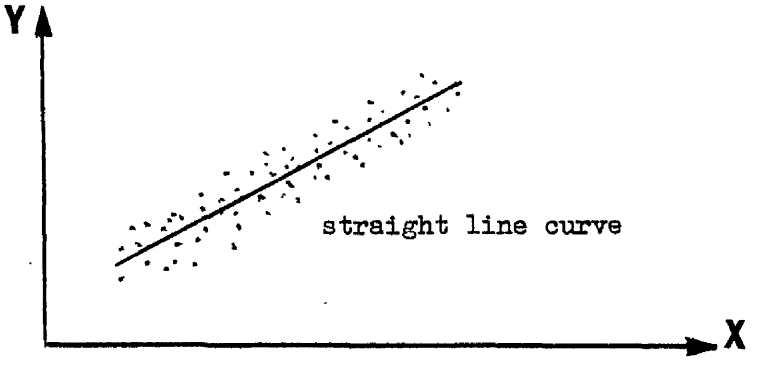


Figure A1.1 Best fit for a scatter diagram.

To evaluate whether the data are a "good fit" to our line or equation, we need the concept of correlation. Correlation tells us how close the data points cluster around the curve or line. While regression defines a proposed relationship between the variables, correlation tells us how good that relationship is. A high correlation between variables shows that they change their values in a related manner, but we must realize that this does not prove or imply a cause and effect relationship. Regression analysis assumes that there is a cause and effect relationship between the dependent and independent variables; correlation studies make no such assumption. Correlation coefficients will range from -1 to +1. A coefficient of -1 means perfect negative correlation. A coefficient of 0 means absolutely no correlation, and a coefficient of +1 means perfect positive correlation. The Square of the correlation coefficient is called coefficient of determination.

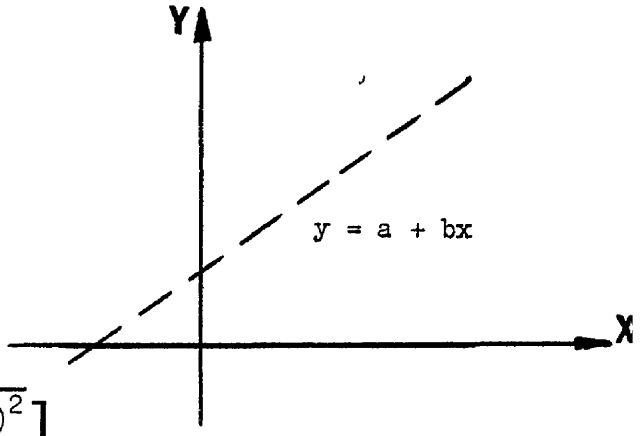
Furthermore, the equations used in curve fitting are as follows:

* Linear Regression:

$$b = \frac{\sum x_i y_i - \frac{\sum x_i \sum y_i}{n}}{\sum x_i^2 - \frac{(\sum x_i)^2}{n}}$$

$$a = \left[\frac{\sum y_i}{n} - b\frac{\sum x_i}{n}\right]$$

$$r^2 = \frac{\left[\sum x_i y_i - \frac{\sum x_i \sum y_i}{n}\right]^2}{\left[\sum x_i^2 - \frac{(\sum x_i)^2}{n}\right]\left[\sum y_i^2 - \frac{(\sum y_i)^2}{n}\right]}$$

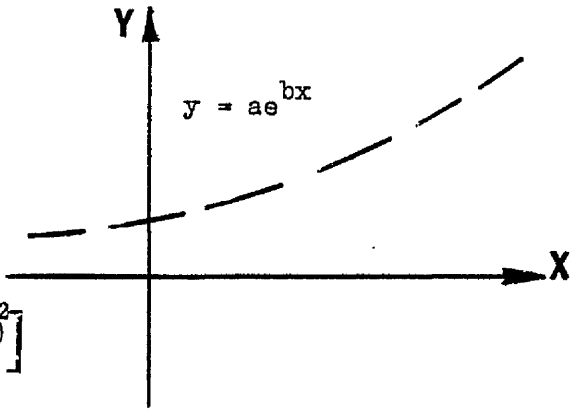


* Exponential Curve Fit:

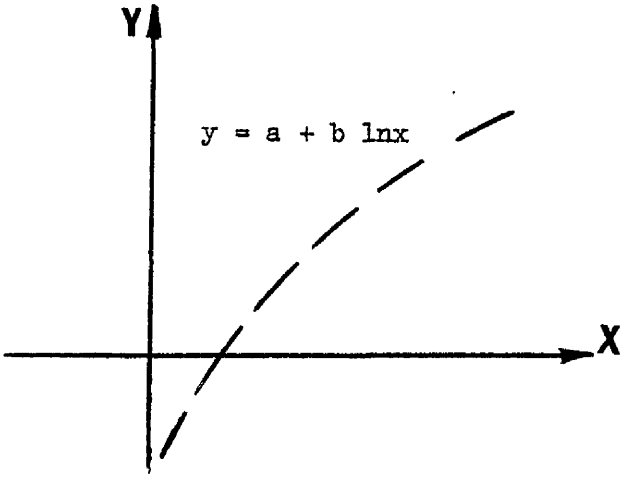
$$b = \frac{\sum x_i \ln y_i - \frac{1}{n}(\sum x_i)(\sum \ln y_i)}{\sum x_i^2 - \frac{1}{n}(\sum x_i)^2}$$

$$a = \exp\left[\frac{\sum \ln y_i}{n} - b\frac{\sum x_i}{n}\right]$$

$$r^2 = \frac{\left[\sum x_i \ln y_i - \frac{1}{n}\sum x_i \sum \ln y_i\right]^2}{\left[\sum x_i^2 - \frac{(\sum x_i)^2}{n}\right]\left[(\ln y_i)^2 - \frac{(\sum \ln y_i)^2}{n}\right]}$$



* Logarithmic Curve Fit:

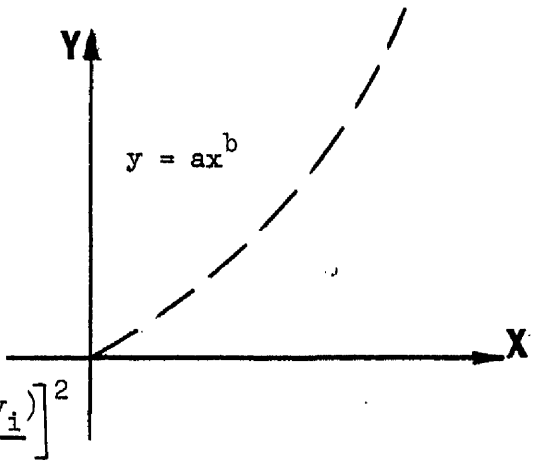


$$b = \frac{\sum y_i \ln x_i - \frac{1}{n}\sum \ln x_i \sum y_i}{\sum (\ln x_i)^2 - \frac{1}{n}\left(\sum \ln x_i\right)^2}$$

$$a = \frac{1}{n}\left(\sum y_i - b\sum \ln x_i\right)$$

$$r^2 = \frac{\left[\sum y_i \ln x_i - \frac{1}{n}\sum \ln x_i \sum y_i\right]^2}{\left[\sum (\ln x_i)^2 - \frac{1}{n}\left(\sum \ln x_i\right)^2\right]\left[\sum y_i^2 - \frac{1}{n}\left(\sum y_i\right)^2\right]}$$

* Power Curve Fit:



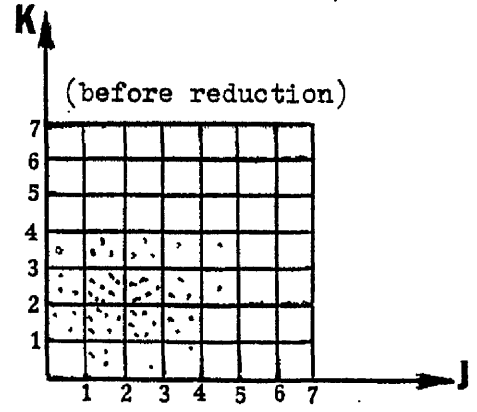
$$b = \frac{\sum (\ln x_i)(\ln y_i) - \frac{\left(\sum \ln x_i\right)\left(\sum \ln y_i\right)}{n}}{\sum (\ln x_i)^2 - \frac{\left(\sum \ln x_i\right)^2}{n}}$$

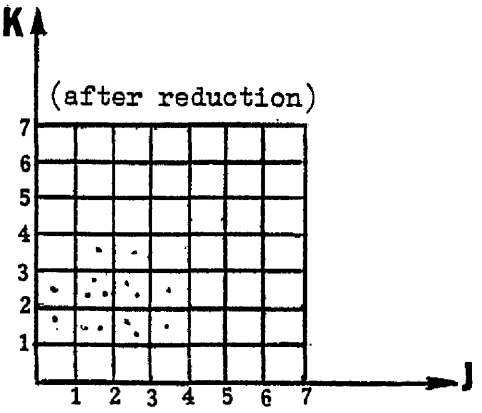
$$a = \exp\left[\frac{\sum \ln y_i}{n} - b\,\frac{\sum \ln x_i}{n}\right]$$

$$r^2 = \frac{\left[\sum (\ln x_i)(\ln y_i) - \frac{\left(\sum \ln x_i\right)\left(\sum \ln y_i\right)}{n}\right]^2}{\left[\sum (\ln x_i)^2 - \frac{\left(\sum \ln x_i\right)^2}{n}\right]\left[\sum (\ln y_i)^2 - \frac{\left(\sum \ln y_i\right)^2}{n}\right]}$$

2. Reduction Methods:

"Reduction" is a key word used by several system designers, since the significance of raw measurements will not be apparent at the first level of presentation, and their meaning must be extracted by the application of numerical methods. Reduction can be used in real time systems, since reduction means storing the measurements and reduce them later, after the end of the session.





Reduction can be performed using a reduction factor as shown in figure (a) and figure (b), which can be calculated as follows:

$$r = \frac{X}{Y} \quad (* \text{ r is a reduction factor}*)$$

with

$$X = \sum_j \sum_k x_{jk} \quad , \quad Y = \sum_j \sum_k y_{jk}$$

and

$$y_{jk} = \frac{x_{jk}}{r} \qquad \forall \text{ j and k.}$$

We are interested in the minimum value of r, which satisfies the design requirements, since this is the choice corresponding to the maximum degree of representativeness.

Often, reduction means a method of presentation also. "The form in which results are presented can greatly facilitate (or confuse) their interpretation" ((/Ferrari 78/)). One way is to use a graphical representation, such as Gantt Charts and Kiviate graphs.

Moreover, a convenient and flexible organization for the collected data may be achieved by storing them into a database and providing the users with a query language by which they can interrogate the database and interact with it.

3. Analysis of Variance Methods:

The term analysis of variance designates those data analysis methods which can be used to estimate the relative influence of different sources of variation on the values of a performance index. These methods decompose the total variation of the index into components which correspond to the sources of variation being considered. When all the factors in an experiment are quantative, regression analysis techniques can be applied, otherwise analysis-of-variance methods should be used.

Finally, there are many other helpful methods we did not discover so far, for example:

a. Reduction of dimensionality (multivariate methods).
b. Histograms and graphical presentation methods.
c. Time series analysis.
d. Monte-carlo Method.
e. Multiple-regression methods.
:
:
:etc. for further information the reader is referred to ((/Kobayashi

78/)) ((/Ferrari 78/))((/Box and Jenkins 70/))((/Bock, Yancy and Judge 73/)).

Appendix A2

Abbreviations.

* M = No, of active users (Terminals).
* $\overline{TSM}$ = Average no. of tasks per multiaccess job.
* $\overline{TH}$ = Average think time.
* λ = Mean interarrival time.
* MPT = Mean CPU time.
* MSI = Medium scheduling interval.
* MZ = Memory size.
* SPZ = Segment/Page size.
* MNI = Mean no. of interactions.
* PWS = Period of working set.
* MSL = Mean value of reference string length.
* FPZ = Fraction of process size.
* ST = Swapping time (i.e. time to move one byte into memory)
* SDF = Scheduling discipline factor.
* PBZ = Physical block size of disc file.
* MBR = Mean no. of backing store records.
* MDR = Mean no. of disc file records.
* CST = Context switching time.
* PI = Process invocation time.
* PCT = Permitive call time.
* $\overline{R}$ = Average response time.
* $\overline{PBT}$ = Average processor busy time.
* $\overline{X}$ = Average interactive system throughput.
* ψ = Effective degree of multiprogramming.
* DU = Disc utilization.
* DRU = Drum utilization.
* TCT = Terminal connect time.
* MJP = No. of multiaccess jobs processed.
* RSR = Ratio of simulation time to real time.
* POT = Processor overhead time.
* WPT = Mean waiting time in CPU queue.
* WST = Mean waiting time in storage manager queue.
* WTT = Mean waiting time in terminal manager queue.
* WDT = Mean waiting time in disc manager queue.
* WDRT = Mean waiting time in drum manager queue.
* r^2 = Coefficient of determination.

```
*  PT = Processor productive time.
*  OT = CPU overhead time.
*  RJU = Ratio of jobs processed per no. of active users.
*  SO = Simulation option.
*  NO = Number of jobs.
*  SP = Simulation period.
*  IRN = Initial random no.
*  CSP = Collecting statistic period.
*  NPB = Number of priority level of batch jobs.
*  MRT = Maximum average resonse time.
*  MPP = Maximum no. of ports of process.
*  PDZ = Process descriptor table size.
*  ASIR = Average service time of interrupt routine.
*  SDZ = Section descriptor table size.
*  ST = Simulation time.
*  PIT = Processor idle time.
*  UPD = Utilization time of peripheral devices.
*  NPC = Number of processes created.
*  NJS = Number of jobs entered to the system.
*  NPP = Number of processes processed.
*  NPPAS = Number and % of process abort by the system.
*  TSPS = The times spent in each process state.
*  KO = Kernel overhead.
*  EARUP = Estimation of the average records used by processes.
*  MNEBST = Max. no. of entries used by simulation table.
*  MNEBRT = Max. no. of entries used by real system table.
```

Appendix B1

The Representation of a Multiclass customer closed queueing network.

Consider a computer system of M devices (processors, service centres). Customers in the system may belong to any one of a finite number of classes. The collection of classes constitutes a class group (e.g. trivial and non-trivial jobs) which consists of a main customer class and a number of associated system customer classes. Let the classes be numbered 1,2,....R and let

n_{ir} be the number of customers of class r present at centre i.

$n_i = \sum_{r=1}^{R} n_{ir}$ is the total number of customers present at centre i.

$W_r = \sum_{i=1}^{M} n_{ir}$ is the total number of class r customers in the system.

$N = \sum_{i=1}^{M} n_i$ is the total number of customers in the system.

Since the subsystem model is represented by a closed queueing network, N is fixed.

Now, during an observation period $(0,T)$, the following operational quantities are collected:

$A_{ir}(n)$: number of arrivals of class r customers at centre i, when $n_{ir} = n$, $0 \leqslant n < N$.

$C_{ir}^{js}(n)$: number of times a customer of class r requests service at centre j as a class s customer immediately after completing a service request at centre i, when $n_{ir} = n$, $0 < n \leqslant N$.

$T_{ir}(n)$: total time during which $n_{ir} = n$, $0 \leqslant n \leqslant N$.

B_{ir} : total busy time of device i for class r customers.

Let, the outside world as centre 0, then

$C_{or}^{ir}(n)$: number of customers of class r whose first request is for centre i when $W_r = n$ (no class changes occur on entry to the system) $0 \leqslant n < N$.

$C_{ir}^{or}(n)$: number of class r customers whose last service request is for centre i, when $n_{ir} = n$ (no class changes occur on exit from the system), $0 < n \leqslant N$.

The number of completions of class r jobs at centre i when $n_{ir} = n$, is computed as:

$$C_{ir}(n) = \sum_{j=0}^{M} \sum_{s=1}^{R} C_{ir}^{js}(n) \quad , \; 0 < n \leqslant N.$$

The number of arrivals of class r customers at the system when $W_r = n$, $0 \quad n \quad N$, is:

$$A_{or}(n) = \sum_{i=1}^{M} C_{or}^{ir}(n).$$

Using the above quantities, the following derived operational quantities are defined:

* $X_{ir}(n) = \frac{C_{ir}(n)}{T_{ir}(n)}$, request completion rate for class r customers at centre i when $n_{ir} = n$, $0 < n \leqslant N$.
* $P_{ir}(n) = \frac{T_{ir}(n)}{T}$, proportion of time when $n_{ir} = n$, $0 \leqslant n \leqslant N$.
* $S_{ir}(n) = \frac{T_{ir}(n)}{C_{ir}(n)}$, the service function for class r customers at centre i when $n_{ir} = n$.
* $C_{ir} = \sum_{n=1}^{N} C_{ir}(n)$, the total number of completions for class r customers at centre i.
* $X_{ir} = \frac{C_{ir}}{T}$, overall request completion rate for class r customers at centre i.
* $S_{ir} = \frac{B_{ir}}{C_{ir}}$, the mean service time over all class r completions at device i.
* $U_{ir} = \frac{B_{ir}}{T}$, the utilization of centre i by class r customers.
* $U_i = \sum_{r=1}^{R} U_{ir}$, utilization of centre i.
* $J_{ir} = \sum_{n=1}^{N} n\, T_{ir}(n)$, job-seconds accumulated at centre i by class r customers.
* $\overline{n}_{ir} = \frac{J_{ir}}{T}$, the average number of customers of class r at centre i.
* $R_{ir} = \frac{J_{ir}}{C_{ir}}$, mean response time per request by class r customers at centre i.
* $q_{ir}^{js} = \frac{1}{C_{ir}} \sum_{n=1}^{N} C_{ir}^{js}(n)$, the fraction of completions of class r customers at centre i which are followed immediately by requests, as class s customers, for service at centre j.

Using the above quantities, the following operational laws have been derived:

* $X_{ir} = \sum_{n=1}^{N} P_{ir}(n) \quad X_{ir}(n)$

* $U_{ir} = X_{ir} S_{ir}$

$$* \quad \bar{n}_{ir} = \sum_{n=1}^{N} P_{ir}(n) \cdot n$$

Now, using the above quantities and laws, we can construct many operational theorems by imposing some additional simplifying assumptions upon the system. These assumptions are:

1. Job Flow Balance:

The principle of job flow balance implies the following — for each centre i, X_{ir} is the same as the total input rate of class r customers to centre i. Therefore, if job flow is balanced, we refer to X_{ir} as the centre throughputs:

$$C_{js} = A_{js} = \sum_{i=0}^{M} \sum_{r=1}^{R} C_{ir}^{js} = \sum_{i=0}^{M} \sum_{r=1}^{R} \sum_{m=1}^{N} C_{ir}^{js}(n)$$

since $q_{ir}^{js} \cdot C_{ir} = C_{ir}^{js}$

$$C_{js} = \sum_{i=0}^{M} \sum_{r=1}^{R} q_{ir}^{js} \; C_{ir} \; ,$$

hence, dividing by T:

$$\boxed{X_{js} = \sum_{i=0}^{M} \sum_{r=1}^{R} X_{ir} \, q_{ir}^{js}} \quad \text{...where } j=0,..,M. \quad s=1,..,R.$$

The above equations are called the job flow balance equations. The job flow balance equations can also have the following form:

$$\begin{aligned} \sum_{s=1}^{R} \sum_{j=1}^{M} X_{js} &= \sum_{i=0}^{M} \sum_{r=1}^{R} X_{ir} \left(\sum_{j=1}^{M} \sum_{s=1}^{R} q_{ir}^{js} \right) \\ &= \sum_{i=0}^{M} \sum_{r=1}^{R} \frac{X_{ir}}{C_{ir}} \left(\sum_{j=1}^{M} \sum_{s=1}^{R} C_{ir}^{js} \right) \\ &= \sum_{i=0}^{M} \sum_{r=1}^{R} \frac{X_{ir}}{C_{ir}} \left(C_{ir} - \sum_{s=1}^{R} C_{ir}^{os} \right) \\ &= \sum_{i=0}^{M} \sum_{r=1}^{R} X_{ir} - \sum_{i=0}^{M} \sum_{r=1}^{R} \sum_{s=1}^{R} X_{ir} \, q_{ir}^{os} \end{aligned}$$

hence

$$\sum_{s=1}^{R} X_{os} = \sum_{s=1}^{R} \sum_{i=0}^{M} \sum_{r=1}^{R} X_{ir} \, q_{ir}^{os} = \sum_{i=0}^{M} \sum_{r=1}^{R} X_{ir} \, q_{ir}^{or}$$

It is important to note that in these forms the job flow balance equations have no direct solution for the closed queueing network, since X_{os} is unknown. To solve these equations, let us define:

$$* \quad V_{ir} = \frac{X_{ir}}{X_{or}} \quad , \text{ the flow of customers in class r through centre i relative to the system throughput for class r customers.}$$

then

$V_{ir} = \frac{C_{ir}}{C_{or}}$, and V_{ir} is the mean number of completions in class r at centre i. This is also called visit ratio of class r customers at centre i.

Hence, using the above definition we can represent the job flow balance equations in a different form which can be solved:

$$\left.\begin{aligned} & V_{or} = 1 \\ & V_{js} = \sum_{r=1}^{R} q_{or}^{js} + \sum_{i=1}^{M} \sum_{r=1}^{R} V_{ir}\, q_{ir}^{js} \end{aligned}\right\} \begin{array}{l} r = 1,\ldots,R \\ j = 0,\ldots,M \\ s = 1,\ldots,R \end{array}$$

2. State Transition Balance:

The state of the system is described by a vector:

$$\underline{n} = (\underline{n}_1, \underline{n}_2, \ldots\ldots, \underline{n}_M)$$

where

$$\underline{n}_m = (n_{m1}, n_{m2}, \ldots, n_{mR}).$$

We define:

$T(\underline{n})$: the total time during which the network is in state $\underline{n}$ during the interval $((0,T))$.

$\therefore P(\underline{n}) = \frac{T(\underline{n})}{T}$, the time proportion for $\underline{n}$,

where

$$\sum_{\underline{n}} P(\underline{n}) = 1.$$

$\underline{k},\underline{n},\underline{m}$, : denote distinct system states.

$Q(\underline{n},\underline{m})$: the number of one-step transitions (i.e. without passing through any intermediate state) observed from state $\underline{n}$ to state $\underline{m}$, where

$Q(\underline{n},\underline{n}) = 0$.

Using the above definitions the job flow balance equations represent the state transition balance equations where the number of entries to every state is the same as the number of exits from that state during the observation period.

i.e. $\sum_{\underline{k}} Q(\underline{k},\underline{n}) = \sum_{\underline{m}} Q(\underline{n},\underline{m})$ for all $\underline{n}$.

Define the transition rate from $\underline{n}$ to $\underline{m}$ as follows:

$$H(\underline{n},\underline{m}) = \frac{Q(\underline{n},\underline{m})}{T(\underline{n})} \quad , (T(\underline{n}) \neq 0)$$

Then the state transition balance equations can be written as:

$$\sum_{\underline{k}} T(\underline{k})\, H(\underline{k},\underline{n}) = T(\underline{n}) \sum_{\underline{m}} H(\underline{n},\underline{m})$$

or,

$$\sum_{\underline{k}} P(\underline{k})\, H(\underline{k},\underline{n}) = P(\underline{n}) \sum_{\underline{m}} H(\underline{n},\underline{m})$$

for all $\underline{n}$ for which each $H(\underline{n}, .)$ is defined.

Now, by adding the normalizing condition $\sum_{\underline{n}} P(\underline{n}) = 1$ and noting that $P(\underline{n}) = 0$ for those $\underline{n}$ not included in the above state balance transition equations, a unique set of $P(\underline{n})$ will satisfy the equations. To show the solution, we first need to define the following:

From the one-step behaviour assumption ((The only observable state changes result from single customers either entering the system, or moving between pairs of centres in the system with accompanying class changes)), we can derive that the neighbour states of $\underline{n}$ are:

$$\underline{n}_{ir}^{js} = (n_{11},\ldots, n_{ir} + 1,\ldots,n_{js} - 1,\ldots.n_{MR})$$

$$\underline{n}_{ir}^{or} = (n_{11},\ldots, n_{ir} + 1,\ldots,n_{MR})$$

$$\underline{n}_{or}^{jr} = (n_{11},\ldots,n_{jr} - 1,\ldots,n_{MR})$$

Then for all $\underline{n}$

$$\therefore \sum_{\substack{i,j\\r,s}} P(\underline{n}_{ir}^{js})\, H(\underline{n}_{js}^{ir}, \underline{n}) + \sum_{i,r} P(\underline{n}_{ir}^{or})\, H(\underline{n}_{ir}^{or}, \underline{n}) + \sum_{j,r} P(n_{or}^{jr}) H(\underline{n}_{or}^{jr}, \underline{n})$$

$$= P(\underline{n}) \left\{ \sum_{\substack{i,j\\r,s}} H(\underline{n}, n_{js}^{ir}) + \sum_{i,r} H(\underline{n}, n_{or}^{ir}) + \sum_{j,r} H(\underline{n}, n_{jr}^{or}) \right.$$

The first on the left and on the right correspond to customers making (i,r), (j,s) transitions; The second terms on the left and the right correspond to customers exiting the system from centre i; the third terms on the left and on the right correspond to jobs entering the system at centre j. The sums over i and j extend over 1,...,M. whereas the sums over r and s extend over 1,...,R. For a closed system such as our interactive system, the second and third terms on the left and right should be dropped, and q_{ir}^{js} should be increased by $q_{ir}^{or} \cdot q_{os}^{js}$.

In order to solve the state transition balance equations we also have to express them in terms of measurable parameter. For this purpose we should use two new assumptions:

Firstly, the representation of the state transition rate is:

$$H(\underline{n}_{ir}^{js}, \underline{n}) = \frac{q_{ir}^{js}}{S_{ir}(n_{ir} + 1)} \quad \text{(and similarly for other state transitions)}$$

The two assumptions will help in simplifying the above equations. The assumptions are:

First that: (device homogenity) $\dfrac{Q(\underline{n}_{ir}^{js}, \underline{n})}{T(\underline{n}_{ir}^{js})} = \dfrac{C_{ir}^{js}(n_{ir} + 1)}{T_{ir}(n_{ir} + 1)}$

i,e. that the rate of state transitions from $\underline{n}^{js}_{ir}$ to $\underline{n}$ equals the rate at which device i throughputs of class r customers equals $n_{ir} + 1$, which immediately go to device j as class s customers irrespective of the number of class s customers already at device j.

Second that: $C^{js}_{ir}(n_{ir} + 1) = q^{js}_{ir}\, C_{ir}(n_{ir} + 1)$

(routing homogenity)

i.e. the routing frequencies are independent of the state of the system (but may depend on the load N).

Hence, we can obtain the following homogeneous rates:

$$* \quad H(\underline{n}^{js}_{ir}, \underline{n}) = q^{js}_{ir}\, I_{js}/S_{ir}(n_{ir} + 1)$$

$$* \quad H(\underline{n}, \underline{n}^{ir}_{js}) = q^{js}_{ir}\, I_{ir}/S_{ir}(n_{ir})$$

$$* \quad H(\underline{n}^{or}_{ir}, \underline{n}) = q^{or}_{ir}\, /S_{ir}(n_{ir} + 1)$$

$$* \quad H(\underline{n}, \underline{n}^{ir}_{or}) = q^{or}_{ir}\, I_{ir}/S_{ir}(n_{ir})$$

$$* \quad H(\underline{n}^{jr}_{or}, \underline{n}) = X_{or}\; q^{jr}_{or}\, I_{jr}$$

$$* \quad H(\underline{n}, \underline{n}^{or}_{jr}) = X_{or}\; q^{jr}_{or}$$

where

$$I_{ir} = \begin{cases} 1 & \text{if } n_{ir} > 0 \\ 0 & \text{if } n_{ir} = 0 \end{cases}$$

The homogenized balance equations are now:

$$\sum_{\substack{i,j \\ r,s}} P(\underline{n}^{js}_{ir}) \cdot \frac{q^{js}_{ir}\, I_{js}}{S_{ir}(n_{ir} + 1)} + \sum_{i,r} P(\underline{n}^{or}_{ir}) \cdot \frac{q^{or}_{ir}}{S_{ir}(n_{ir} + 1)} + \sum_{j,r} P(n^{jr}_{or})\, X_{or}\, q^{ir}_{or}\, I_{jr} =$$

$$P(\underline{n}) \left\{ \sum_{\substack{i,j \\ r,s}} \frac{q^{js}_{ir}\, I_{ir}}{S_{ir}(n_{ir})} + \sum_{i,r} \frac{q^{or}_{ir}\, I_{ir}}{S_{ir}(n_{ir})} + \sum_{j,r} X_{or}\, q^{jr}_{or} \right\} \quad \text{for all } \underline{n}.$$

Now, consider the right-hand side:

$$\sum_{\substack{i,j \\ r,s}} \frac{q^{js}_{ir}\, I_{ir}}{S_{ir}(n_{ir})} + \sum_{i,r} \frac{q^{or}_{ir}\, I_{ir}}{S_{ir}(n_{ir})} = \sum_{i=1}^{M} \sum_{j=0}^{M} \sum_{r=1}^{R} \sum_{s=1}^{R} \frac{q^{js}_{ir}\, I_{ir}}{S_{ir}(n_{ir})}$$

$$= \sum_{i=1}^{M} \sum_{r=1}^{R} \frac{I_{ir}}{S_{ir}(n_{ir})} \cdot \sum_{j=0}^{M} \sum_{s=1}^{R} q^{js}_{ir}$$

$$= \sum_{i=1}^{M} \sum_{r=1}^{R} \frac{I_{ir}}{S_{ir}(n_{ir})}$$

and

$$\sum_{j,r} X_{or}\, q^{jr}_{or} = \sum_{r} X_{or} \sum_{j=1}^{M} q^{jr}_{or} = \sum_{r} X_{or} = X_o$$

Hence, the state transition balance equations are reduced to:

$$\sum_{\substack{i,j\\r,s}} P(\underline{n}_{ir}^{js}) \frac{q_{ir}^{js} I_{js}}{S_{ir}(n_{ir}+1)} + \sum_{i,r} P(\underline{n}_{ir}^{or}) \frac{q_{ir}^{or}}{S_{ir}(n_{ir}+1)} + \sum_{j,r} P(\underline{n}_{or}^{jr}) X_{or} q_{or}^{jr} I_{jr}$$

$$P(\underline{n}) \sum_{i,r} \frac{I_{ir}}{S_{ir}(n_{ir})} + X_o \qquad \text{for all } \underline{n}.$$

Finally, the solution of the above equations is:

$$P(\underline{n}) = \frac{1}{G} \sum_{i=1}^{M} \sum_{r=1}^{R} F_{ir}(n_{ir})$$

where

$$F_{ir}(n_{ir}) \begin{cases} 1 & \text{if } n_{ir} = 0 \\ X_{ir} S_{ir}(n_{ir}) F_{ir}(n_{ir}-1) & \text{if } n_{ir} > 0. \end{cases}$$

$$\text{i.e. } F_{ir}(n_{ir}) = X_{ir}^{nr} \prod_{k=1}^{n_{ir}} S_{ir}(k), \quad n_{ir} > 0.$$

G is a normalizing constant given by

$$G = \sum_{\underline{n}} \prod_{i=1}^{M} \prod_{r=1}^{R} F_{ir}(n_{ir}) \text{ , where the summation extends over all possible } \underline{n}.$$

Noting that:

$$P(\underline{n}_{ir}^{js}) = \frac{X_{ir} S_{ir}(n_{ir}+1)}{X_{js} S_{js}(n_{js})} \cdot P(\underline{n})$$

$$P(\underline{n}_{ir}^{or}) = X_{ir} S_{ir}(n_{ir}+1) \cdot P(\underline{n})$$

$$P(\underline{n}_{or}^{jr}) = \frac{1}{X_{jr} S_{jr}(n_{jr})} \cdot P(\underline{n})$$

For closed queueing networks, these equations do not allow a unique set of solutions. The analyst can, however, obtain a unique set of visit ratio data and derive the X_{ir} by means of an arbitrary normalization.

The procedure of calculating the normalization factor is given in appendix B.2.

Now, we can formulate the important performance indices. For more information the reader is referred to Roode ((/Roode 79/)).

* <u>The Utilization:</u>

$$U_{MR} = \frac{1}{G(N,M,R)} \sum_{m=1}^{N} m(X_{MR} S_{MR})^m \sum_{P=0}^{N-m} \frac{1}{P+m} h^{(M)}(P,R-1)\, g(N-m,P,M-1,R).$$

* <u>The Average Queue Lengths:</u>

$$\underline{n}_{MR} = \frac{1}{G(N,M,R)} \sum_{m=1}^{N} m(X_{MR} S_{MR})^m \sum_{p=0}^{N-m} h^{(M)}(P,R-1)\, g(N-m,M-1,R).$$

Calculation of the Normalization Factor of Multiclass Customer Closed Queueing Network.

The normalization constant is defined as:

$$G(N,M,R) = \sum_{\underline{n} \in S(N,M,R)} \prod_{i=1}^{M} \prod_{j=1}^{R} (x_{ij}\, S_{ij})^{n_{ij}}$$

where

$$S(N,M,R) = \left\{ \underline{n} = (n_{11}, n_{12}, \ldots, n_{1R}, n_{21}, \ldots, n_{2R}, \ldots, n_{MR}) \;\middle|\; \sum_{i=1}^{M} \sum_{j=1}^{R} n_{ij} = N \;\&\; n_{ij} > 0 \;\forall i,j \right\}$$

Roode ((/Roode 79/)) generalizes the approach followed by Buzen ((/Buzen 73/)), he considers the following function:

$$g(n,m,r) = \sum_{\underline{n} \in s(n,m,r)} \prod_{i=1}^{m} \prod_{j=1}^{r} (x_{ij}\, s_{ij})^{n_{ij}}$$

Then, for $m > 1$ it follows that

$$g(n,m,r) = \sum_{p=0}^{n} \sum_{\underline{v} \in s(p,r)} \prod_{j=1}^{r} (x_{mj} s_{mj})^{v_j}\, g(n-p,\, m-1,\, r)$$

where

$$S(P,r) = \left\{ \underline{v} = (v_1, v_2, \ldots, v_r) \;\middle|\; \sum_{j=1}^{r} v_j = P, v_j \geq 0 \;\forall j \right\}$$

Let $h^{(m)}(P,r) = \sum_{\underline{v} \in s(P,r)} \prod_{j=1}^{r} (X_{mj} S_{mj})^{v_j}$

Then it follows that

$$h^{(m)}(P,r) = h^{(m)}(P,\, r-1) + (X_{mr} S_{mr})\, h^{(m)}(p-1,\, r)$$

with

$$h^{(m)}(P,r) = (X_{m1} S_{m1})^{P} \qquad P = 0,1,\ldots,N; \forall m.$$

and

$h^{(m)}(0,r) = 1$ for $r = 1,2,\ldots,R$; $\forall m$;
$h^{(m)}(0,0) = 1$, $h^{(m)}(P,0) = 0$, $P \geq 1$.

Thus

$$g(n,m,r) = \sum_{p=0}^{n} h^{(m)}(P,r)\, g(n-p,\, m-1,\, r)$$

and the iterative calculation of $G(N,M,R)$ is completed if we observe that:

$$g(n,1,q) = \sum_{\underline{v} \in S(n,q)} \prod_{j=1}^{q} (x_{ij} s_{ij})^{v_j} \text{ for any } q \geq 1.$$

so that

$$g(n,1,q) = g(n,1,q-1) + (X_{1q} S_{1q})\, g(n-1,1,q)$$

with

$$g(n,1,1) = (X_{11} S_{11})^n, \qquad n = 1,2,\ldots,N.$$

$$g(0,1,q) = 1 \ \forall q.$$

In fact this last calculation is unnecessary since

$$g(n,1,q) = h^{(1)}(n,q) \qquad \begin{matrix} q = 1,2,\ldots,R. \\ n = 1,2,\ldots,N. \end{matrix}$$

Note that in order to calculate the normalizing constant G(N,M,R) we need only calculate

$h^{(m)}(P,r)$, m=1,...,M; r=1,...R and P=1,....,N;

g(n,m,R) , n=1,...,N; m=2,...,M with

$g(n,1,R) = h^{(1)}(n,R)$, n=0,1,...,N.

Appendix B2

The Algorithm of Calculating the Normalization Factor of Interactive Computer System Models.

The algorithm fills in numbers in a two-dimensional matrix g. The columns of g correspond to devices, rows to loads. The computation starts with 1s in the first row and Os in the first column below the first row. A typical interior element is computed from

$g(n,k) = g(n,k-1) + Y_k\, g(n-1,k)$,

where $Y_k = V_k S_k$. The normalizing constant G is $g(N,K)$. It can be computed in 2KN arithmetic operations.

Let $G[0...N]$, initially 0, denote a vector array representing a current column of g, and let $Y[1....K]$ denote another vector containing $V_1S_1,...,V_kS_k$. Then the algorithm is

```
{initialize}: G[0] := 1;
FOR k:=1 TO K DO {Compute kth column}
   FOR n:=1 TO N DO
     {[G][n-1] contains g(n-1,k);
      G [n]    contains g(n,k-1)}
      G [n] :=G [n] + Y [k]  G [n-1] ;
```

When this procedure terminates, G [N] contains the normalizing constant.