

809

JAWAHAR

A CRITICAL STUDY OF COMPUTER SYSTEM PERFORMANCE EVALUATION

In Partial Fulfilment of the Requirements
for the degree of
MASTER OF TECHNOLOGY in ELECTRICAL ENGINEERING

by
KESAV VITHAL NORA

Department of Electrical Engineering
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

to the

August 1970

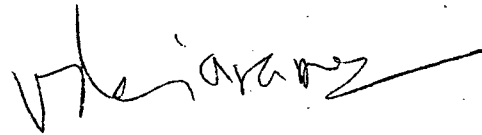
1006

004.24 TH
N77 Cr
TH3152

- to MY PARENTS

CERTIFICATE

This is to certify that the thesis entitled
"A CRITICAL STUDY of COMPUTER SYSTEM PERFORMANCE EVALUATION"
is a record of the work carried out under my supervision and
that it has not been submitted elsewhere for a degree.



V. Rajaraman
Head, Computer Centre
and
Professor of Electrical Engineering
Indian Institute of Technology
Kanpur

August 1970

ACKNOWLEDGEMENTS

I would like to express my deep appreciation and gratitude to my thesis supervisor, Professor V. Rajaraman, for initiating me to the work presented in this report. He has found time in his busy schedule for many interesting and fruitful discussions. But for his sustained interest, guidance and kind help in innumerable ways, the work presented here would not have been possible. I cannot thank Dr C.R. Muthukrishnan enough for his academic counsel, constructive criticism and friendship.

To the viva voce committee, Professors H.N. Mahabala and M.P. Kapoor, I am indebted for their criticism and helpful suggestions.

Acknowledgements are due to Mr T. Radhakrishnan for useful discussions on analytical and simulation models for computer systems; Mr R.N. Basu for clarifying many details of operation of large third generation computer systems.

My acknowledgements are also due to : Mr H.K. Nathani for his excellent and efficient typing of the report; Mrs R. Oberoi for helping with the proofs and preparing an errata. I am grateful to the staff of the Computer Centre and friends for the very pleasant time I had during my stay here.

- Kesav Vithal Noni

August 1970

ABSTRACT

Performance Evaluation is the central problem in the study of the economics of computer systems. The problem of performance evaluation, difficulties in assessing performance, aims of performance evaluation and techniques developed to date are critically discussed. Computer systems like the IBM 7044-1401 are examined in the light of performance evaluation. Various operation schedules for the IBM 7044-1401 are proposed and compared. A software monitor to obtain the user job profile is outlined and a model to compare configurations of this system is proposed.

CONTENTS

		CERTIFICATE	
		ABSTRACT	
		ACKNOWLEDGEMENTS	
<u>CHAPTER</u>	I	<u>INTRODUCTION</u>	1
		1.1 Motivation for assessing performance of computer systems	1
		1.2 The problem of performance evaluation	11
		1.3 Aims of performance evaluation investigations	16
		1.4 Obstacles to performance evaluation	19
		1.5 Some measures of performance	21
		1.6 Scope of work reported	24
<u>CHAPTER</u>	II	<u>ANALYSIS OF COMPUTER SYSTEMS FOR EVALUATIVE STUDIES</u>	26
		2.1 Analysis of digital computer computer hardware	27
		2.2 Analysis of the operating systems	59
		2.3 Analysis of workload	82
		2.4 Conclusions	95
<u>CHAPTER</u>	III	<u>SIMULATION TECHNIQUES FOR COMPUTER SYSTEM EVALUATION</u>	98
		3.1 Introduction	98
		3.2 Introduction to simulation languages	100
		3.3 Motivation for simulation of computer systems	105
		3.4 Classification scheme for various simulation models	109

		3.5 Difficulties in simulating multi-processor computer systems	117
		3.6 Acquisition of data for assessing system performance	124
		3.7 Conclusions	127
<u>CHAPTER</u>	<u>IV</u>	<u>STUDY OF EVALUATIVE PROBLEMS FOR THE IBM 7044-1401 COMPUTER SYSTEM AT IIT/K</u>	129
		4.1 Computational resources at IIT/K	130
		4.2 Operation schedules for IBM 7044-1401 at IIT/K	133
		4.3 Models for IBM 7044 computer system	142
		4.4 Specification of a software monitor and a technique for implementation	148
		4.5 Conclusions	154
<u>CHAPTER</u>	<u>V</u>	<u>CONCLUSIONS AND SUGGESTIONS FOR FURTHER WORK</u>	156
		5.1 Summary and conclusions	156
		5.2 Suggestions for further work	158
		REFERENCES	159

CHAPTER I
INTRODUCTION

Electronic digital computers have been in existence for just over two decades. From early beginnings, like the ENIAC in 1946, successive developments in speed, size, versatility and usage of computer systems have been rapid, especially in the last six years or so. Today, in the industrially advanced countries of the West in particular, computers have pervaded many spheres of human activity and endeavour. Surprisingly enough, the design, usage and the economics of computers is yet far from an exact science; the art of computer programming is one such glaring example. The study of one of these aspects, the economics of computer systems, forms the main theme of work reported here in this thesis.

Section 1.1: Motivation for assessing performance of computer systems

The study of the economics of computer systems necessitates a detailed study of factors that affect their manufacture, operation, maintenance, application and utilization. Hence motivation for improvements and innovations in computer system architecture, system design and software support can be tracked to the economic

benefits or trade offs obtained. A brief review of major advances in computer systems is therefore in order.

The evolution of computer systems is a well reported topic in literature. To quote a few, Richards [1], Hassitt [2] and Rao [3] have given excellent accounts of it in their works. Therefore, no attempt is made here to elaborate on this subject or trace its history. However, certain developments relevant to the subject of study are outlined here; their understanding constitutes some of the requisites for the study of the economics of computer systems. It may be noted in passing that terms like evaluation, performance, workload have to be defined in the context of computer systems and will be elaborated upon in the next section.

Advances in computer hardware can be categorised as shown below:

- (i) Technological advances in terms of device speeds, reliability etc.
- (ii) Advances in logical organisation. Specifically, the development of interrupt facility, memory protect feature and over-lapped input/output operations by use of data channels which together provide for non local concurrency in the operation of the computer system. The introduction of instruction look ahead and memory hierarchy in primary storage provide local concurrency in operation.

Advances in computer system software fall into two classes:

- (i) the development of programming aids through the formulation of user oriented, procedure and problem oriented, programming languages;
- (ii) and the development of operating systems [4,5,6].

The latter is of interest; it provides a logical environment in which the user may conduct his work and it allocates him the necessary resources to accomplish the work. Important features of an operating system are:

- (i) file management [7] and input/output control;
- (ii) segmentation [8] and memory management;
- (iii) multiprogramming, time sharing and processor allocation [9, 10, 11, 12, 13, 14].

As is evident from these developments, a present generation computer system is an integrated complex of hardware and software resources which afford the user necessary facilities for problem solving. An analysis of such a system, in the light of various economic factors imbibed in their design, should provide the necessary motivation for the ensuing study of this subject. The outlook that computer systems are a collection of hardware and software resources has its advantages. The motivation of the manufacturer to produce such systems and the study of resources to be allocated to the user to suffice his needs are both

presented in a uniform manner.

Hardware resources of the computer system are an aggregate of the following:

- (i) computational resource, provide by the comprehensive and powerful instruction repertoire of the central processing unit;
- (ii) memory resources for program and data storage provided by the main memory;
- (iii) a hierarchy of auxiliary storage, for storing large volumes of information, provided by peripheral storage devices, operating over a large range of information transfer rates; in general their storage capacity is inversely related to their information transfer rate;
- (iv) facilities for communication between the main memory and the peripheral auxiliary storage devices, provided by the data channels and device controllers or interfaces;
- (v) and facilities for communication between the user and the computer system, provided by the host of ever increasing input/output devices.

The underlying philosophy in the design of hardware is to achieve modularity, compatibility, adaptability and reliability in computer systems. To a certain extent, these factors have been motivated by economic considerations.

Modularity in system design allows interchangeability of system components or subsystems and flexibility in their interconnection. This means that virtually any desired system can be assembled. Compatibility has given scope to the development of a whole line or family of computer systems with very nearly the same machine language. Such computer systems are upward and downward compatible. This means that the same machine language program can be run or executed on any computer system in the family. It is therefore an easy matter to upgrade or downgrade the system depending on the changes perceived in the workload. Consequently, effort in recoding of already available or developed software is kept to a minimum, if not absent altogether. Therefore, modularity and compatibility make it possible to most economically configure any computer system of desired size and speed or change the system to suit changes in demand with least expenses incurred.

The usage of computers for a wide range of applications is enhanced by those features in system design which permit ease in adapting the system resources to the needs of the application on hand. A good example of adaptability in system design is the basic unit of storage in the IBM system/360: the eight bit 'universal' byte. This eight bit byte stores a character, two decimal digits and of course eight binary bits of information. This facilitates character handling,

decimal arithmetic and binary arithmetic and high utilization of main memory. Therefore, this 'universal byte' can be easily adapted to any type of application. Such features result in good dividends as the manufacturer does not have to produce several different systems to cover the range of applications. Reliability in computer systems is an outcome of technological progress and redundancy in system design. This feature has made possible, among other things, the space exploration mission and is absolutely necessary for the development of a 'computer utility'. Configurations of the IBM system/360 Model 67 [15] and the MULTICS System [11] embody all these principles and illustrate the extreme complexity of hardware that is now possible with present day technology.

Stand alone hardware resources cannot provide the necessary problem solving environment to the user. In fact, the user works only on a virtual machine presented him by the operating system. Saltzer [16] gives an interesting illustration of the virtual machine seen by the user of the MULTICS System. The operating system shields the user from the details of hardware and its limitations and offers the user a logical environment for programming his problems. Hence, **software** resources available to the user are analysed in the following paragraphs. These resources are:

- (i) a number of language processors which simplify his programming burden by giving him the power to express procedures for solution of his problems in a manner akin to his way of thinking;
- (ii) a large program library providing the user with 'ready made' debugged program packages which perform commonly needed functions;
- (iii) a filing system which facilitates the creation, editing and retrieval of large files of information;
- (iv) and a set of procedures which allocate to the user any of the system resources as he needs them.

The main features in the design of the afore-named software resources are:

- (i). features to attract and cater to the needs of a large user population and creating a machine or hardware independent environment to the user;
- (ii) and incorporation of proper resource allocation algorithms which are to ensure an efficient utilization of the overall system and provide the best grade of service.

It is seen that economic benefits that arise from the various considerations in hardware design are very easy to point out. They are very tangible in the sense that

quantitative assessment of trade offs obtained is possible. This is not the case with the benefits gained by use of software. Questions can be raised about the increase in efficiency of programming by use of higher level languages. Studies of the behaviour of programmers, in terms of time taken to develop programs for given problems, when programming in different environments and having different type and amount of interaction with the computer system, have been conducted and these results summarised [17]. Answers to such questions and results of such studies are difficult to evaluate as they are subjective in nature. It is apparent that there are many such problems in assessing the utility of software. Nevertheless, the benefits accrued from software are obvious even though not quantifiable.

Usage of computers is costly. Resource allocation by the operating system is therefore an important aspect of the tasks to be performed by it. Performance assessment of computers and hence the evaluation of various techniques of resource allocation indeed invite attention.

The design of computer systems described in the preceding pages is always open ended so as to leave room for future developments. It should be clear by now that the extent of analysis to be carried out before the design of computer system hardware and software is quite

tremendous and these analyses contribute greatly to the economics of manufacture of computer systems and their usage. Experience about the performance of such systems prove invaluable for the design of future systems.

Amongst the many other problems in the area, the selection of computers and the pricing of computer services have evinced interest and are worth mention. Joslin [18] and Sharpe [19] have discussed problems associated with computer selection. The main problem in the selection of computer system is the evaluation of tenders and contracts offered by the manufacturer or the vendor. Trade offs between rental costs and purchase costs have to be estimated. The main components of rental costs are the Basic Monthly Rental charges, the Extra Usage charges and the Maintenance charges. On the other hand the break up of costs in the purchase of computer systems is the Purchase charges, and the Maintenance charges. There are several hitches in the comparison of competitive efforts. The first difficulty is in estimating accurately the expected or projected workload, which the computer system in question is expected to service. Secondly, to find procedures to estimate quantitatively, criteria for performance of the computer system in the light of the workload. On overcoming these problems, the comparison of various rental contracts

and purchase terms can be obtained. The types of contracts offered have been discussed by Sharpe and he proceeds to conduct an initial analysis of the relationships between the various costs quoted in them. Other issues in computer selection have been reviewed by Sharpe and Joslin. Certain periodicals like Datamation and Computers and Automation bring out articles frequently on related topics. As they are numerous and the material presented in them is not the central theme here, reference to such works are omitted.

Two problems are prominent when pricing computer services. The first is to fix the price for usage of the computational facilities and type of service given. These issues clearly necessitate the evaluation of performance of the computer system. The second problem is one of the devising proper accounting schemes for logging the actual usage of system resources by the user [20]. The log, besides being useful in accurately billing the user, provides a rich source of information about user characteristics.

Problems of maintenance, of operation, of management and personnel for manning computer installations also come under the investigations in the economics of computer systems. As is evident from the nature of all the issues raised so far, a number of subjective criteria will play an important role in the final solution of any of these issues.

At this stage, it is quite apparent that the solutions to the afore-mentioned problems depend on the evaluation of performance of computer systems. Due to the central character of performance evaluation in the study of the economics of computer systems, this is the only problem considered at length in this thesis. The problem of imposing economic constraints on the performance of computer systems in order to solve other problems mentioned is considered secondary here.

Section 1.2: The problem of performance evaluation

The word 'performance' means the manner or success of working or the execution of a task. The engineering sense of the word is related to efficiency. And by efficiency is meant the power to produce the intended result; for most purposes, it is the ratio of machine's output of energy to its input. Unfortunately, the usefulness of computer is in no way related to the energy they consume to produce the intended result. They deal with a volatile and intangible commodity that information is. Therefore, the normal usage of words like efficiency and performance do not apply to computer systems.

It is well known that computers can deal with only those problems in which quantification is possible, even through mere coding. The study of computation as an

abstract entity is a hot topic of current research. And this effort is unfortunately not addressed to the problem of quantification of computation as in the context of a general purpose computer. A machine independent measure for computation is therefore not a reality as of now. Hence, it is noted regrettably, 'performance' of computer systems has not yet been defined in a machine independent manner.

Be that as it may, in the absence of rigour concerning the fundamentals of this area of computer science, and for the lack of a better word, performance of computer systems will be conveniently assumed to be their 'behaviour' in the execution of 'workload' presented them by the users. Several criteria for 'behaviour' of the computer systems have been proposed; examples are the utilization of system components, service rendered to users, time taken for given tasks or just a trace of the operation of the system. The 'workload' of a computer system is much more difficult to define. For the purposes of this thesis, the workload of a computer system is defined in terms of statistical measures of factors that are deemed to affect system performances. Performance evaluation of computer systems is thus a set of techniques for relating user requirements (the workload) to the capabilities of the computer system in the context of such criteria as explained above.

Two types of investigations are undertaken in performance assessment or performance evaluation of computer systems. The first concerns the study of the sensitivity of the behaviour of the computer system with respect to certain system parameters. Examples of system parameters could be the configuration of the computer system or the various resource allocation algorithms in the operating system. Such studies are directed towards improvement of the behaviour of the system for a given workload. Hence the effectiveness of certain features of system design are examined. In so far as the relationships between various aspects of the computer system and its operation are not well understood, this type of investigation may be carried out merely to gain insight into these matters. Most of the effort in performance evaluation has been of this type.

The second type of performance evaluation study is one concerning comparison of different computer systems for a given workload. Due to the fact that the workload of the computer system too has not been defined in a machine independent manner, this problem is much more difficult to tackle than a problem of the first type. However, this is an important problem to solve in order to compare various approaches to system design on a common footing.

With this background, it is appropriate at this stage to ponder about desirable characteristics of performance evaluation techniques. In essence, performance evaluation entails the following:

- (i) characterisation of user workload, the jargon is user job profile, in a machine independent manner;
- (ii) definition of suitable criteria for the behaviour of the computer system;
- (iii) imposition of software constraints on user job profile to obtain the actual work to be done by the hardware system;
- (iv) determining the behaviour of the system and hence quantitatively finding the values of various criteria set forth earlier by using the above.

No performance evaluation technique really attempts a clear and detailed solution of the problem as outlined above. The difficulties faced in evaluating performance are many, their discussion is held over to a later section.

Techniques of performance evaluation have had early beginnings. In the late 1950's, such techniques were developed to aid in the system design of Project STRETCH. But it is only in the last few years that such efforts have gathered force. The interest they have aroused is evident from survey papers on performance evaluation by Calingaert [21], Meredith Smith [22], Wickens [23] and

Drummond [24]. As the problem covers many facets and activities of computer systems, these surveys have not done full justice in bringing out the nature of the problem in its entirety. However, they have succeeded in pointing out various difficulties faced when seeking solution to the problem and have given a broad classification of efforts in this direction to date.

Performance evaluation techniques are concerned with the modelling of computer systems and users. Rather than modelling the computer system or the user in complete detail, most of these techniques only attempt to study one aspect of the user demand and system behaviour at a time. Modelling has been done either by means of analysis, analytical or otherwise, of computer systems and users or by use digital computer discrete system simulation techniques. These efforts are discussed in the succeeding chapters.

Other characteristic features of techniques of assessing performance of computer systems are:

- (i) the techniques give an absolute or relative measure of performance,
- (ii) the methods give a measure of potential or actual performance.

Section 1.3: Aims of performance evaluation investigations

The preceding sections present an intuition for performance evaluation of computer systems. It is worthwhile to consider for the sake of completeness, what the aims in assessing performance could be. As pointed out earlier, a number of subjective values creep into the solution of issues raised earlier: it follows naturally that the aims of an evaluative study are subjective. It is not possible to present these issues in any measure of completeness due to lack of experience and as such information is not readily available. However, to lend some clarity in specifying the objectives of assessing performance of computer systems, a perfunctory classification and analysis is attempted.

Obviously, the aims of assessing performance of computer systems depend on the group of people conducting the evaluative study. These groups can be broadly classified into the following categories:

- (i) the 'user group' which comprises people interested in the selection of suitable computers;
- (ii) the 'systems group', comprising system analysts and system programmers;
- (iii) and the 'manufacturers group', competitors in the computer industry who want to stay in business!

What follows is an attempt to analyse the interests of each group and hence arrive at some of their objectives in performance evaluation and the type of techniques they need.

The user group is interested in acquiring a computing facility which will suffice their needs. Unfortunately, it is not easy, even for the experienced user to define or specify the expected workload. This point will be amplified in a later section of this chapter. Assuming that the workload has been estimated, the strategy for selection would depend on the cost per unit work, the overall cost and the grade of service offered to the various priority groups in the user community. To this end the user group would have need for the following techniques.

- (i) Procedures to estimate workload that the computer system is expected to service and projection of load growth in years to come.
- (ii) Procedures to estimate trade offs between hardware and software available in the light of the defined workload.
- (iii) Procedures to determine suitable mode of operation.
- (iv) Procedures to estimate the grade of service to be expected.
- (v) Procedures for comparing performance of various configurations of a computer system for the defined workload and comparing various computer systems to the same end.

If the above considerations are quantitatively worked out for eligible systems then the final choice made is a compromise between economic constraints and trade offs available with extra investment.

The systems group is directly concerned with the management of the computer system. Their interests are primarily in affording a powerful computer service to the user community and ensuring the maximum utilization of the system resources. Naturally, details of the computer system resources indicated in the preceding section are of immediate interest to them. Their purpose in performance evaluation would be:

- (i) to compare various resource allocation algorithms,
- (ii) to compare software packages
- (iii) and to develop new techniques for the above in order to provide better service or improve system utilization.

This group will find it essential to monitor the operation of the system so as to obtain information to do the above tasks. Trade offs between system utilization and service rendered to the user community are chief factors of interest in their performance evaluation studies.

The interests of the manufacturers group are only too well known. Their aim is to maximize the returns on their investments. Besides, they have to survive in a

competitive market. This group might have for its objectives in performance evaluation the following:

- (i) measurement of improvement in performance due to the introduction of new hardware features;
- (ii) comparison of their computer system with those manufactured by competitive firms;
- (iii) development of software features in order to attract users.

In order to accomplish the above, a survey of current and projected applications of computer systems and their relative importance in the market is necessary. Techniques need be developed to achieve the same.

Section 1.4: Obstacles to performance evaluation

In the preceding sections, the problem of performance evaluation and techniques necessary to accomplish it were sketched in simple terms. Questions raised in this connection are either fundamental in character or examine details of design features and operation of computers. Illustrations of questions of a fundamental nature are: for the purposes of system performance, what does a job or task or application mean? and how does it load the system resources? Examples of questions regarding details of design or details of operation for evaluative studies are: what improvement in performance can be expected by the introduction of a scratch pad memory or an instruction look ahead feature? Or, what

scheduling algorithm should be used for processor allocation in a time shared, multi programmed environment? Questions similar to the latter two are quite objective and much fruitful analysis has been done in solving such problems. The chief difficulties in performance evaluation are in obtaining answers to questions of the first type and these are considered in the following paragraphs.

An automaton is said to be universal in computation if it can perform the mapping specified by any 'computable' function. Digital computers are engineering approximations to automata exhibiting this property of universality in computation. In the same vein, computer programs are indeed engineering approximations to the specification of the computable functions and they normally have limitations as to the range of data they can handle for any given problem. Therefore, it is seen that results from the Theories of Automata, Formal Languages and Computation are not useful in describing programs for the pragmatic purpose of defining the workload of computer systems. This situation behoves the development of a different attitude towards programs and computers. A more detailed look at what might this attitude be is necessary and is presented in a later chapter. It suffices to stress here that as of now there exists no precise formulation or quantification of user requirements.

A consequence of this state of affairs is that the user - machine interface is ill defined, be it a language or a device. This is evident from the ever increasing number of user oriented languages and 'tailormade' input/output devices being designed. Also, there exists a lack of accepted standards and hence communication between various levels of people interacting with the computer system is greatly hampered. All this stems out of the fact that the study of computers is more of an art than a science at present.

TH-3152

Besides these difficulties, there are some technological difficulties in conducting a performance evaluation study. For example, the dynamic measurement of data of all types within a computer system so as to determine its behaviour or characterise the user job profile is not always possible with available tools like hardware or software monitors. However, these problems will be hopefully overcome in the near future. At the University of California, Los Angeles, it is reported that the operation of several computers, including the IBM System/360 Model 67, is being monitored by a SDS Sigma 7 computer! Information obtained from such experiments should prove interesting indeed.

Section 1.5: Some measures of performance

In this section, the concept of 'behaviour' of the computer system is elaborated upon by defining some measures or criteria for assessing computers system Thesis 1

681.3.06

N774

a.



performance. At the outset it must be made clear that the actual computation of values for some of these criteria may not yet be possible due to difficulties outlined earlier. Again, for similar reasons, it is easier to evaluate these criteria for a computer system with change in some system parameters than for different computer systems. A few of these measures are abstracted here from the multitude devisable as the goals of evaluation of computer systems are subjective. Some of these measures are more suited to a batch processing environment and others to a time shared mode of computer operation. They are the following:

- (i) Throughput. This is a measure of the steady state state capacity of the system. In other words, it refers to the volume of work done in an accepted unit of time. Constrained by cost, this is the most objective performance criterion to those managing the system.
- (ii) Turnaround or average response time. This represents the average the average time between the receipt of a specific service request of a user by the sytem and the satisfaction of that request at the terminal or to the user. Again, constrained by cost, this is the single most useful measure of performance to a user.

- (iii) Average waiting time. This measure is an indication the load in each priority group of the user population and time required to service the group.
- (iv) Average queue length. This measure is linked with the average waiting time and indicates the load for each priority group. The relationship between allocation of priorities to user groups and their effect on these measures merits attention.
- (v) Availability. This criterion concerns the number of hours, excluding maintenance and breakdown, the computer system is available for use. It is a measure of the percentage of total time the system is in service for user programs.
- (vi) Reliability. This measure indicates the frequency of hardware and software failures and is measured as the mean time between hardware failures or mean error free time.

The mentioned measures of performance are the quantitative end results of an evaluative effort. The goals of performance evaluation themselves can be realised in terms of the criteria mentioned above. These are not all the useful measures; more will be introduced at appropriate stages in succeeding chapters.

Section 1.6 Scope of work reported

Literature on computer system performance evaluation is scattered over a wide spectrum of technical journals. An exhaustive survey of all literature has not been possible. Instead, a cogent and uniform treatment of problems of performance evaluation is presented here. Motivation for particular approaches to the solution of these problems and classification schemes for techniques developed or studies conducted is also given in this report.

In Chapter II are discussed various analyses of computer systems for the purpose of performance evaluation. They are, the analysis of computer hardware, the analysis of operating systems and the analysis of workload. The usefulness and limitations of certain techniques for comparison of computer hardware are brought out. As an illustration, sample calculations are made for comparison of central processing units by the instruction mix technique. In the discussion on operating systems, attention is restricted to the various resource allocation algorithms. In view of the preceding analyses, a quantification of workload is attempted.

Chapter III embodies an introduction and review of simulation techniques for the study of computer systems. In order to compare different techniques of discrete system simulation, the GASP II A simulation package was debugged

and implemented in the IBSYS operating system of IBM 7044. (Earlier, GPSS III was the only available programming system for discrete system simulation.) Motivation for use of simulation techniques, classification of simulation experiments and a comprehensive look at difficulties faced when using this technique are the main features of this chapter. Chapters II and III form a critical state-of-the-art appraisal of computer system performance evaluation.

Batch processing computer systems with off line satellite systems to manage slow peripheral work - in particular the IBM 7044-1401 computer system at Indian Institute of Technology, Kanpur - are examined from the point of view of performance evaluation in Chapter IV. Operation schedules for such systems are discussed in relation to throughput and turnaround. These operation schedules were implemented for the IBM 7044-1401 systems. Next, models are proposed for this system. Of particular interest is a model which takes care of parallel input-output operations. This model should prove useful in the evaluation of several configurations of a system for a given workload. No pragmatic evaluative work is possible till statistics to characterise user workload are gathered and the system studied in relation to this workload. Therefore, the requirements of a software monitor to achieve this and techniques to implement the same are outlined.

The concluding chapter, Chapter V, contains suggestions for further work and mention of problems not discussed here.

CHAPTER II

ANALYSIS OF COMPUTERS SYSTEMS FOR EVALUATIVE STUDIES

The analysis of computer systems from the standpoint of performance evaluation can be categorised in the following manner:

- (i) Analysis of computer hardware facilities and logical organisation. The purpose of this analysis is to examine various hardware features and their contribution to system performance in the light of the workload.
- (ii) Analysis of the operating system. Two aspects of the operating system are of interest. First, the features of the virtual machine offered to the user. And second, the effectiveness of various resource allocation algorithms in terms of utilization of system resources in presenting the user this virtual machine. Of special interest is the 'overhead' incurred in providing the virtual machine and in allocating the system resources. Here, 'overheads' are understood to mean the utilization of system resources by the system itself in order to provide this virtual

machine to the user.

- (iii) Analysis of workload. The workload of a computer system is a composite of user demands of system resources and the overheads incurred in offering these resources to the user. An attempt is made to clearly delineate these two components of the workload and characterise the user workload in terms of its demand on the system resources.

In this chapter, various attempts at such analyses will be reviewed. Problems yet to be tackled are pointed out and wherever possible a brief qualitative analysis will be included.

Section 2.1: Analysis of digital computer hardware

Computer system hardware is a collection of resources like the Central Processing Unit (CPU), main memory space, input/output channels, auxiliary memory, communication links and input/output devices. The analyses of these resources will be qualitative as information about details of their design is not forthcoming. The study of their interconnection is even more intriguing. The latter indeed poses a very challenging problem in performance evaluation and is the central issue in the study of computer structures. It is hoped that the analysis of computer hardware will aid in the characterisation of workload attempted in a later section.

The CPU is the most complex subsystem in computer hardware. In the following discussion, the CPU is considered only as a computational resource. Hence, for the purposes of performance evaluation, the main features of the CPU are the instruction repertoire, local and non local concurrency and the data path between the CPU circuitry and the main memory. Main memory space is a system resource by itself and the study of user demands on it will be considered separately.

The instruction repertoire also termed 'instruction set', offers a computational facility to the user. The execution of these instructions requires a wide variety of operations ranging from simple CPU register manipulations to complicated arithmetic operations. In general there are two phases in the execution of an instruction: the instruction decoding phase and the operation phase. Time taken to decode an instruction is a function of the length of the instruction and the generation of the effective address of the operand. And the time for the operation phase is related to the number of operations to be performed. In some present day computers these operations are realised as sequences of micro-operations. Through proper design of instruction codes and micro-operations, the user is given the flexibility to set up non-standard instructions to perform often needed functions. Such a feature is called 'micro program control' and it uses a 'wired in logic' or 'read only memory'. Systematic design of instruction sets is possible

due to such features in system design.

Instruction sets of computers are modular. The manufacturer offers the machine with a 'basic instruction set' or an 'extended instruction set'. The extended instruction set offers useful instructions for particular type of applications. For example, instructions for variable length operations are needed in commercial data processing operations and floating point, double precision arithmetic instructions for use in scientific computations. The need for powerful instructions is therefore seen to arise from the frequency of occurrence of certain types of operations in typical applications.

A measure of effectiveness of the computational resource for a particular type of application is the average instruction time. An arithmetic mean of all instruction times does not suffice for the simple reason that the nature of the application or workload is in no way reflected in the average obtained. Weighted instruction times are more suitable for obtaining the same. The weights assigned to each instruction in the instruction set could for example be simply computed from the frequency of occurrence of that instruction in the workload. As the instruction set of computer system can be quite extensive, instructions are grouped into classes by the operations they perform and the time taken to execute them. Weights are assigned to each of these classes by frequency of occurrence of instructions in the class. Weighted groups as described

above are termed as 'instruction mixes'. Put another way, if there are n groups of instructions with weights W_1, W_2, \dots, W_n and with execution times T_1, T_2, \dots, T_n , then the average instruction time is:

$$\sum_{i=1}^{i=n} W_i \cdot T_i$$

where

$$\sum_{i=1}^{i=n} W_i = 1 \quad \text{and} \quad W_i \geq 0 \quad \text{for all } i \quad [21,22].$$

Instruction mixes are very simple to deal with but very difficult to define. That is, the grouping of instructions into classes and assigning weights to them is not at all an easy thing to do. Firstly, a dynamic instruction-trace of the workload must be available. The instruction-trace of a program can be anything between 50 to 100 times slower than the actual execution of the program. And hence a few minutes of 'typical' computation is actually traced in practice and an instruction mix is defined using this data. Secondly, there is no standardization of the instruction set yet. In fact, each machine is empowered with some special instructions for simplifying computation of a particular type of application and this feature cannot be reflected in an instruction mix.

The implication of the above is that it is difficult to compare CPUs of two different computer systems using just an instruction mix. It is seen that the speed of execution of instructions is a function of several factors mentioned

earlier, notwithstanding the memory speed. Hence comparison of CPUs which differ in any of these factors by using instruction mixes can give misleading results. However, comparison of CPUs with similar features by this technique can be quite fruitful.

Some of the well known instruction mixes for scientific computation are the Arbuckle Mix [25] and the Gibson III Mix [22]. As an illustration, the Arbuckle Mix is given in Table 2.1 and a Commercial Mix [22] is given in Table 2.2. Also shown in tables is the computation of average instruction time obtained by use of these mixes, for IBM 7044 system. It should be noted that the group of instructions in the 'miscellaneous' category of the Arbuckle mix is really an instruction mix in itself. For the sake of simplicity, the fixed point add instruction time is used as the computation time for this category. The reason for this is that the fixed point add instruction requires both the instruction decode phase and the operate phase which uses the sophisticated circuitry of the CPU. Hence, main memory speeds and CPU arithmetic speeds are both reflected in the fixed point add time. In the computations for evaluating the commercial mix, it should be noted that operations rather than instructions have been specified. Sequences of instructions to realise these operations have therefore been used.

INSTRUCTION	FREQUENCY OF OCCURRENCE	IBM 7044 INSTRUCTION μ SECONDS	TIME ON IBM 7044 μ SECONDS
Floating point add	9.5	11.0	104.5
Floating point multiply	5.6	20.0	112.0
Floating point divide	2.0	36.0	72.0
Load/Store	28.5	4.0	114.0
Indexing	22.5	4.0	90.0
Conditional branch	13.2	2.0	36.4
Miscellaneous	18.7	4.0	74.8
			<u>593.7</u>
Average instruction time for IBM 7044 : 5.937 μ seconds			

Table 2.1: The Arbuckle Mix for scientific computation and sample calculation of average instruction time for IBM 7044.

OPERATION	EXTENT	WEIGHT	IBM 7044 TIME FOR 1 OPERA- TION μ SEC	TOTAL TIME ON -IBM 7044 μ SEC
Compare Character	1	9	10	90
	2	5	16	80
	3	7	18	136
	6	1	10	10
	10	1	26	26
	12	3	20	60
Move Character	1	1	10	10
	10	1	36	36
	60	2	44	88
Branch	Taken	15	2	30
	Not Taken	13	2	26
Add 3 Characters		2	4	8
Indexing		40	4	160
Total			100	756

Average instruction time for IBM 7044 = 7.56 μ seconds
 Table 2.2: A commercial mix and sample calculations for IBM 7044

Gross comparison of computational power of CPUs belonging to a family of computer systems like the IBM System/360 series or the ICL 1900 series is very easy, by using this technique. The average instruction time, for scientific and commercial workloads, for some word oriented computer systems is tabulated in Table 2.3. A word of caution is necessary when considering figures for average instruction times given in the table. Due to lack of familiarity with assembly language programming in PLAN (for the ICL 1900 series) and unavailability of correct information regarding usage of special features provided, the average instruction times for commercial computations could be conservative. Computation of this time for some bigger members of this family was not possible due to incomplete information given in the ICL 1900 series introductory text. No computation of average instruction times could be under taken for models of the IBM System/360 series, other than Model 44, due to unavailability of their functional characteristics. As all the computer systems compared in this table are word oriented and have some features in common, the average instruction times give a fair idea of their relative computational power.

COMPUTER SYSTEM	MEMORY SPEED μ SEC	FLOATING POINT OPERATIONS	STORAGE TO STORAGE OPERATIONS	AVERAGE INSTRUCTION TIME μ SEC		Remarks
				Commercial Mix	Arbuckle Mix	
IBM 7040	8.0	H	H	22.48	15.41	Codes used=
IBM 7044	2.0	H	H	7.56	5.93	H-Hardware
IBM System 360 Basic Model 44	1.0	H	S	7.63	5.1	S-Software
IBM System 360 Model 44 with high speed General Registers	1.0	H	S	5.02	3.39	Speed increase by constant factor of 1.6
ICL 1902	6.0	S	S	52.97	599.56	
ICL 1903	2.0	S	S	17.96	253.17	1903-1904 H/S trade off available with MOVE instruction
ICL 1904	2.0	S	H	14.50	37.79	1904-1905 H/S trade off with floating pt arithmetic
ICL 1905	2.0	H	H	14.50	8.97	

ICL 1906	1.1 - 2.25	H	H	9.09	1906-1907 CPUs are identical except for float- ing point arithmetic
ICL 1907	1.1 - 2.25	H	H	3.18	
ICL 1909	6.0	H	?	15.26	

Table 2.3: Comparison of some word-oriented computer systems for commercial and scientific applications

From the results in the table, the following observations can be made: Comparison of IBM 7040 and IBM 7044 is given below:

- (i) The IBM 7044 is about 2.6 times faster for scientific computations and about 2.9 times faster for commercial workload. The main contribution to this increase in speed comes from the higher main memory speed of the IBM 7044 (it is 4 times faster). These machines are more suited for scientific computations. The main drawback of their instruction sets is the paucity of character handling instructions.
- (ii) The IBM System/360 Model 44 is also a machine suited for scientific computation, as advertised. This system is provided with fast hardware for floating

point arithmetic. However, the advertisement that this system is 'optimised for scientific computation' also refers to the fact that several of the excellent character handling instructions available with other models have been excluded in the instruction set of Model 44. Trade offs in speed available with faster registers are easily seen. As is evident, the speed increase by a factor of 1.6 in both scientific and commercial computation, is solely due to use of high speed General Registers.

- (iii) ICL 1902, 1903 and 1904 computer systems are clearly meant for commercial workloads. Floating point arithmetic is done by software in these models and hence the anomaly in speeds for the two types of workloads. ICL 1903 does not have storage-to-storage instructions. In all other aspects (except floating point operations), it is equivalent to ICL 1904 and 1905. Hence the difference between commercial computation speeds of these models and the other two. It is seen that ICL 1904 and ICL 1905 are equally powerful for commercial workload but ICL 1905 has hardware for floating point arithmetic. This is a good example of hardware-software trade offs available with the floating point feature. The same

could be expected of the next pair in the family, the ICL 1906 and 1907 systems. Another surprising result is, that even though ICL 1906 has faster core memory speed than ICL 1905, it is slower in scientific computations. The total percentage of floating point operations in the Arbuckle Mix is only 17.1. Even so, better floating point hardware of ICL 1905 offsets the higher core memory speed of ICL 1906 for this type of computation.

In all the above calculations, various instruction times, given in the introductory texts and programming manuals supplied by the manufacturer, have been used. Normally, these figures quoted by the manufacturer are optimistic instruction times. This is done in order to account for any local concurrency in the action of the CPU. In practice the effectiveness of these features depends on the nature of the user workload. This will be elaborated in the following paragraphs. A closer look at some of the components of a CPU, which provide the basis for local concurrency in its operation, is necessary.

Local concurrency is overlap in the execution of neighbouring instructions in an instruction stream. That is, one or more instructions are in the decode phase while one or more preceding instructions are in the operation phase concurrently.

To make this possible, the CPU is designed as a set of functional hardware units as in the CDC 6600 computer system. These units each perform a micro-operation in the sequence of operations required to execute an instruction. Conventionally, in order to simplify the logical design of the CPU, these units are not available for processing succeeding instructions throughout the execution time of the current instruction. For better utilization of such hardware, the current practice is to release the units on completion of their operation; the released unit is reset or initialised immediately so as to make it available for the operation required by the next instruction, regardless of the completion of execution of the current instruction. Examples of functional hardware units are the fixed point arithmetic unit, the floating point arithmetic unit, the logical unit, the operation-code decoding unit and the effective address generation unit.

Parallel independent operation of these functional hardware units is possible only if enough information (instructions and operands) are available to them. There exists a need for buffering this information in a fast accessible store so as to bridge the CPU-speed main-memory-speed mismatch. Various techniques have been implemented; their design and performance have been studied by simulation techniques and are discussed in the next chapter. However, so long as the speed

mismatch exists, the number of registers in the CPU and extent of the data path between the main memory and the CPU determine the extent of local concurrency possible. The CDC 6600 CPU can process upto three instructions concurrently; these instructions are necessarily not interdependent, i.e. the outcome of any of these instructions does not depend on the outcomes of the preceding instructions. An even more powerful example of local concurrency in the CPU is Project Stretch, IBM 7030; in this system, upto eleven instructions can be in various stages of processing and any interdependence within these instructions is automatically taken care of by the instruction look ahead unit of the CPU.

Another point to be noted is the density of information retrieved per memory reference. With regard to instructions, this could mean the number of instructions packed into the width of the data path between the CPU and the main memory. Maximum density is achieved by provision of instructions of variable length; normally, instructions need either no memory reference or specification of one to two addresses for their execution. For operands, the density of information retrieved depends on the flexibility of the addressing scheme provided in the instruction format. The maximum density is related to the smallest addressable unit of main memory.

The reason for this elaborate digression on local concurrency features of the CPU is to bring out the inadequacy

of the simple instruction mix (representing any given application) in evaluating the speed of computation of the CPU. It is clear that a lot of information about the workload - other than the occurrence of types of instructions - is necessary to evaluate the performance of the CPU.

One way of dealing with differences in central processors and coping with difficulties like local concurrency is to compare the times required to perform specified tasks called 'kernels'. These tasks are coded in the assembly language of the computer system and hence can utilize all the features of the computational facility offered. There is of course no standardisation either of the tasks to be performed or their magnitude. According to Calingaert [21] a kernel is "the central processor coding required to execute a task of the order of magnitude of calculating a social security tax, or inverting a matrix, or evaluating a polynomial". An attempt is made to have the problem coded with equal levels of sophistication by experienced programmers in assembly language.

Again, to represent actual working conditions, a fairly comprehensive set of kernels must be defined. The problem of assigning proper weights to these kernels in synthesising the workload is crucial in evaluating the computational

speed of the central processor by this method. An idea of the types of kernels defined and used so far has been given by Sharpe [19]. Examples are matrix multiplication, square root approximation of floating point numbers, field manipulation (control card scans, source statement scans), editing, field comparison, BCD arithmetic, character manipulation, polynomial evaluation etc. Elaborations on these kernels is given by Sharpe [19].

Kernels are much more general than instruction mixes as they make it possible to compare CPUs with entirely different characteristics. Even in the comparison of CPUs of a family of computers with the same instruction set, kernels are more powerful than instruction mixes as they give the actual distribution of types of instructions. Hence, it is possible to compute CPU times in the light of the actual local concurrency possible. Put in another way, the kernel approach in comparing CPUs with identical instruction sets does not degenerate to instruction mix comparisons due to reasons outlined above (Sharpe has opined that the kernel approach degenerates to the instruction mix approach for the above case, an observation that is objected to here).

This approach has been very successfully used in the selection of computers for postal services in Britain [26].

Features like the 'stack' and associated operations can be included in evaluation of computation speed by this approach whereas this is not at all possible with the instruction mix technique. Stacks prove to be very useful in certain applications. Stack oriented programming technique is explained by Dijkstra [27] and an implementation of such a feature in KDF9 is explained by Hassit [2].

Non local concurrency within a CPU is 'simultaneous' processing of instruction streams which are not necessarily interdependent. This is made possible by the interrupt feature of the CPU. In view of performance, the effort required to switch from one instruction stream to another and levels of priority in the interrupt feature, are the main interests.

As is well known, the CPU operates in two modes or is in one of two states. These are:

- (i) the privileged mode or supervisor state and
- (ii) the normal mode or the problem state.

In a general purpose computer system, the occurrence of an interrupt switches the CPU to the privileged mode. All computations needed to service an interrupt, constitute a part of the overhead incurred.

Interrupts occur due to all sorts of causes. 'External' interrupts can occur due to end of operation of a device on a channel (channel interrupt) or due to on-line user initiated attention requests and the like. 'Internal' interrupts are caused by abnormal conditions within the CPU's operation; examples are, execution of privileged instructions by the user in problem program state or interval timer overflow etc etc.

Asynchronous parallel operation of various hardware subsystems of a computer system and pseudo parallelism in the execution of user problem programs are made possible by interrupt features. In other words, the interrupt feature acts as a vehicle of communication and a means of synchronisation between almost all of the parallel processes within a computer system.

In a large, third generation computer system, the complexity of operations being performed is tremendous. Dijkstra [28] has pointed out that re-creating the sequence of interrupts in such a system is impossible. Not only does this make the development and debugging of operating systems difficult, but it is the root cause why incorporation of this feature in the modelling of computer systems has hardly been attempted yet. As a result, input-output operations have almost been totally excluded in such efforts so far.

In general purpose computer systems, as opposed to real-time computer systems, there is no stringent time bound

on system response. Therefore, interrupts that occur during the execution of an instruction are serviced only on completion of this operation. Some real-time systems allow interrupts to be serviced at the end of the micro-operation being performed for the instruction currently under execution. The servicing of an interrupt might necessitate the saving of CPU status at that instant. System design of the CPU is such that its status can be saved by a single instruction specifically provided for this purpose, or at the most by few normal instructions. As the cost of hardware is coming down, some systems duplicate all the CPU registers so as to switch right away from normal mode to privileged mode without saving the CPU's status. However, at least current 'Program status word' (PSW - as in IBM System/360 or SDS 9400) is saved and loading of the new PSW is effected by CPU hardware.

The time for servicing an interrupt depends on the type of interrupt and will be discussed in section 2.2. Priorities in interrupts cause queuing of interrupt services; the stage of service of each level is indicated by their respective PSWs. Facilities to inhibit certain interrupts while servicing one etc., are provided to ensure proper logic in the execution of programs.

Main memory space is a set of locations whose addressing is linear. Associated with it is an addressing scheme which maps the effective addresses (generated by instructions in a

program) to locations in it. This mapping can either be as simple as a one-to-one mapping as in simple batch processing systems or be quite complicated (its complexity depends on the dynamic storage allocation feature of the computer system). Obviously, the retrieval time of instructions and operands depends on the time taken to map the effective addresses generated, to memory locations. Hardware to perform these operations has been discussed by Wilkes [29] at length.

Attributes of the main memory which affect the performance of the system are:

- (i) the size of the main memory
- (ii) unit of information accessible and
- (iii) the organisation of this store - the effective cycle time is dependent on this.

Other features of main memory - like the type and flexibility of storage protection provided - are important, but not in the context of performance evaluation. Attributes of user programs which affect the utilization of the main memory are:

- (i) their size, i.e. the number of core loads etc.
- (ii) data structures manipulated by them and
- (iii) the pattern of addresses they generate.

Modules of core memory are available with almost all computer systems. However, for every system, there is an upper bound on the size of the main memory installed. In

models for computer systems, this will naturally be reflected as a constrained resource available. Next, the packing of information into the main memory depends on features of adaptability in system design (Section 1.1). This is an indication of the ease of mapping user-data-structures on to the linear structure of the main memory itself. The organisation of memory in independent blocks and interleaving of addresses within these blocks so as to reduce the effective memory cycle time, have been studied by simulation techniques and are discussed in Section 3.4. Lastly, effects of the pattern of addresses generated - on performance of computer systems - can be discussed only in the context of features of dynamic storage allocation and organisation of memory hierarchies. Hence this aspect will be brought up at an appropriate place in the next section.

The CPU and main memory are important resources from the users viewpoint. Other resources like auxiliary storage, input-output devices, data channels, communication links etc., contribute jointly to system performance as a part of the system configuration. The problem here is to find the effectiveness of asynchronous parallelism in input-output operations, with reference to the workload. There have been a few attempts to produce a simple empirical formulae which yield a single figure of merit for the system as a whole [30, 31]. For example, Knight's formula for 'Computing Power' [31]

is

Computing power = Memory Factor X Operations per second
 where operations per second is $10^{12}/[t_c + t_{I/O}]$ in which t_c is time (in micro seconds) required to execute one million operations (instructions) and $t_{I/O}$ is the non overlapped input-output time (in micro seconds) necessary to perform one million operations. Knight measures two kinds of computing power - commercial and scientific. Therefore, the commercial or scientific instruction mix, which he too has defined, is used to compute t_c . The definition of $t_{I/O}$ is somewhat vague; its computation is based on channel width, transfer rate and start, stop, rewind times for primary and secondary input-output units, plus estimates of possible overlap in the input-output system and extent of utilization of primary and secondary units. Several coefficients required in this complex computation of $t_{I/O}$ are provided by Knight, often with one value for commercial workload and another for scientific workload.

- The second component of computing power is defined as follows:

$$\text{Memory factor} = \frac{[(L \cdot 7) N (WF)]^P}{K}$$

where

K is a constant

L word length in bits

N total number of words in memory

WF is 1 for fixed word length systems
 is 2 for variable word length systems

P is 0.5 for scientific computations
 is 0.33 for commercial computations

This formula is primarily based on opinions: "A total of 43 engineers, programmers and other knowledgeable people were contacted and asked to evaluate the influence of computing memory on performance". [31].

Such an approach is open to criticism as it is based on the opinions of various people, however qualified. A more comprehensive, logical procedure for evaluating configurations was developed by Auerbach Info, Inc., a firm of computer consultants. Their approach is known as the 'benchmark problem' technique [32, 33]. The general attempt here is to compare 'standard' configurations (as defined by Auerbach Info. Inc) of different computer systems by computing total program execution times for certain 'typical' application programs called 'benchmark problems'. Examples of benchmark problems defined by Auerbach Info Inc., are: the sequential file update problem, updating files on random access storage, sorting, matrix inversion, evaluation of complex equations and statistical computations.

In order to ensure uniform comparison of computer systems in question, the benchmark problems are rigidly

specified in terms of certain parameters like available input data, computations to be performed and expected output of results. So as to allow maximum utilization of the various distinctive capabilities of each computer system, the method for coding the problem and design of files is left flexible. As mentioned, comparisons are made for standard equipment configurations. The execution time of each benchmark problem for each standard configuration is calculated by computing all input-output times and central processor times and combining these two with due regard for the computer systems capabilities for simultaneous operations. To do this the problems are coded in assembly language and a detailed estimate of these times is computed with the help of this code. To better appreciate the magnitude of work involved in comparing systems using this technique, the sequential file update benchmark problem is taken up in detail in the following paragraphs.

In the sequential file update problem, a master file is read and updated to reflect transaction data contained in a detail file. A record for each transaction is written in a report file. Thus, there are two input files (old master file and detail file) and two output files (updated master file and report file). Such a problem situation is

typical in applications like payroll, inventory, billing etc.

It is important that the problem be parameterised in such a way as to set up the actual operating conditions. Parameters of interest of the sequential file update problem could be:

- (i) activity factor; i.e. ratio of items in detail file to items in master file.
- (ii) size of records in master, detail and report files (these affect I/O times, and packing, and editing operations) and
- (iii) amount of computation per activity.

When computing the execution time of the program, the following need be considered carefully:

- (i) effective CPU and I/O speeds,
- (ii) I/O and CPU overlap,
- (iii) available core storage for program, data and service routines,
- (iv) desirability to use off-line card-to-tape and tape-to-printer operations etc. and
- (v) special features of the computer system under consideration.

An example of a standard configuration specified by Auerbach Info, Inc., is configuration III shown below:

Configuration III

6 Tape Business system

Internal storage : 2000 one address instructions
8000 characters of data

Magnetic Tapes : 6 units @ 30,000 ch/sec.

Card Reader : 500 cards/min.

Line Printer : 500 lines/min

Card Punch : 100 cards/min

Indexing : Yes

Overlapped I/O : Yes

When considering a specific computer system, equipment closest to the specified configuration is chosen. An assembly language coding of the problem is obtained. Using this program code, the following basic timings are computed:

- (i) I/O times to read/write a physical record of the master, detail and report files.
 - (ii) Processor delays during I/O, taking into consideration the CPU-I/O overlap available.
 - (iii) Computations involved in searching the master file records for a match with the detail transaction and computations necessary to update the appropriate master file record.
- Detailed block diagrams, flowcharts are available with Auerback Info, Inc., which specify precisely the computations to be

performed. The analyst who codes the problem can therefore take full advantage of individual features of each computer system.

On completion of all these calculations, graphs of program execution time (say for processing 10,000 records of master file) vs. the activity factor are plotted for each configuration. The transformation of program execution times on standard configurations to the same on suitable configurations quoted in the manufacturers tenders is achieved with the help of the 'User's Guide' of Auerbach Standard EDP reports. Unfortunately, this information is not published in those technical articles which promote this technique.

Some of the drawbacks of the 'benchmark problem' technique are:

- (i) It is not possible to examine the effects of features like multiprogramming etc. that are possible with the given hardware configuration.
- (ii) Sharpe [19] has reported an attempt to estimate benchmark problem times by actually running the program on competitive standard configurations. There were numerous difficulties in doing so. "One program could not be compiled on one of the systems.

Execution times for another proved incomparable because the execution path was dependent on the sequence of pseudo random numbers generated and each system generated a different sequence. Execution time for yet another could not be compared because one manufacturer ran the problem in a multiprogrammed mode and obtained an elapsed processor time of nearly zero. Another simulated tapes on a magnetic drum". It is easy to observe that actual configurations and operating conditions are not very close to the standards specified by Auerbach Info, Inc. Execution time of programs obtained by directly running the program on the computer system are at best grossly approximate. Limitation on measuring time is either due to the resolution of the interval timer or due to the resolution of the meters provided on various components of the system. A more accurate estimate of time may be quite necessary in some circumstances, especially if certain programs are executed frequently in production runs.

(iii) The set of benchmark problems is too small to cover all the application areas or all types of workload. Hence benchmark problem times do not directly yield a solution to the problem of comparing configurations. These results must therefore be viewed with proper caution and in perspective with the rest of the workload and other operating conditions prevalent.

Irrational as it might seem, there have been several attempts to compare the instruction mix or the kernel technique with the benchmark problem approach [21,22]. Clearly, the former provide only a measure of effectiveness for the CPU whereas the latter is a technique for comparing configurations of a system.

The general problem of comparison of configurations is still an open problem and not much work has yet been done towards its solution. This problem is rendered more difficult by the complexity of operation of present day general purpose computer systems. For example, how does one compare the effectiveness of the configuration of CDC 6600 system (shown in Figure 2.1) to that of the GE 645 MULTICS system (shown in Figure 2.2)? There is a marked accent on parallelism in the design of the CDC 6600 configuration. There are 10 identical peripheral processing units (PPUs) connected to 12 data channels through a 10 x 12 switch. Programming

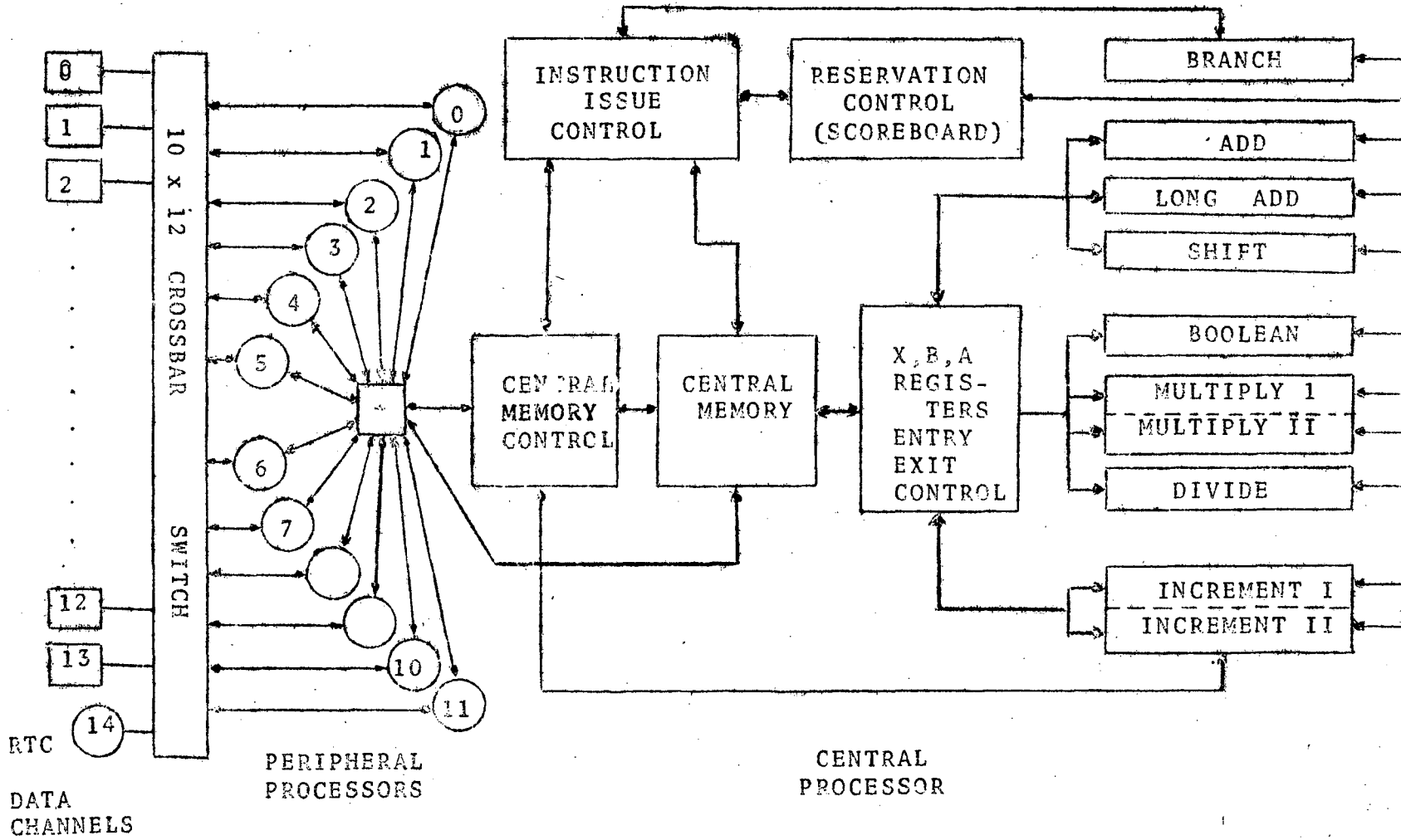


Figure 2.1: The CDC 6600 configuration

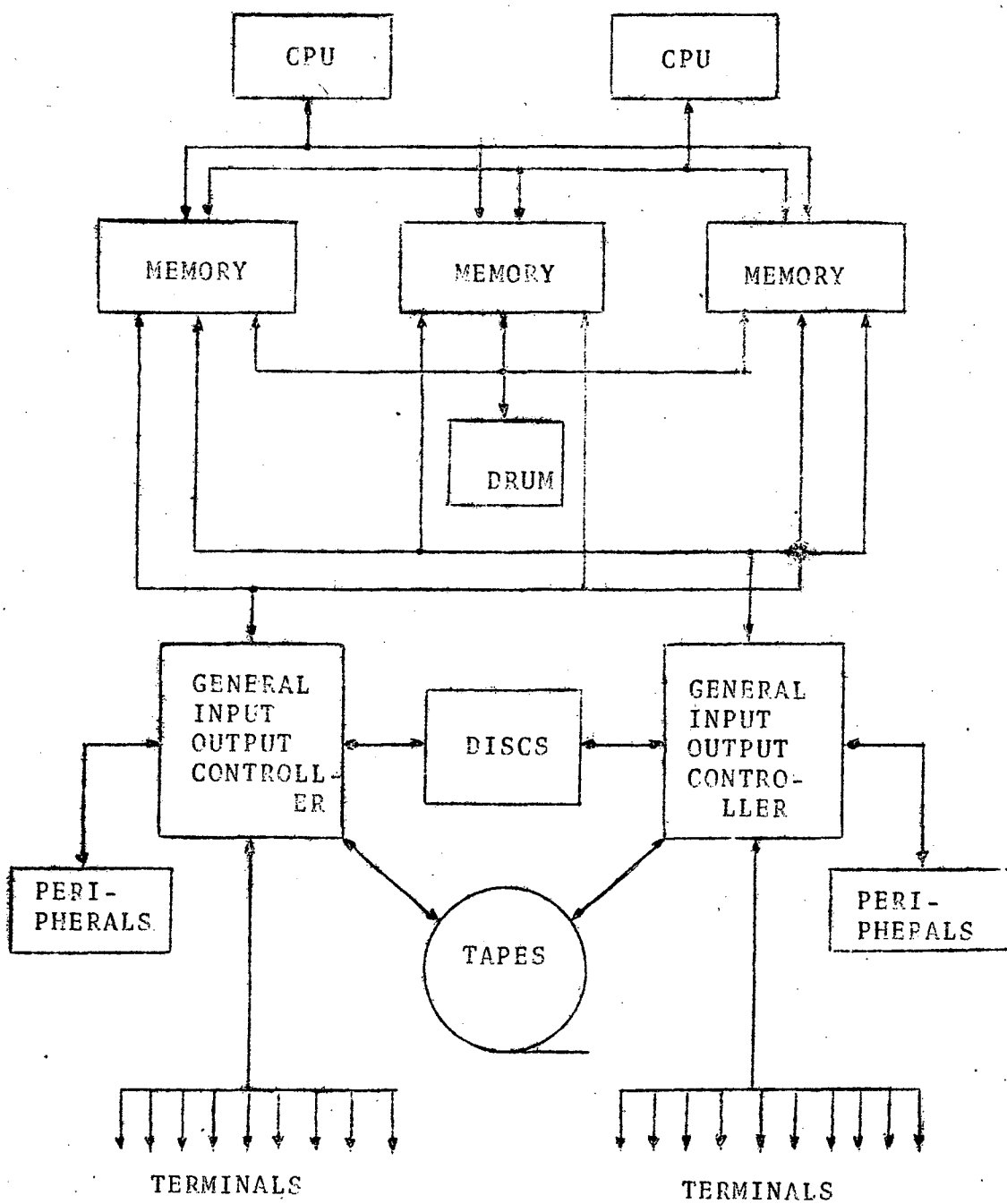


Figure 2.2: The GE 645 MULTICS configuration

the system so as to make maximum utilization of the PPU's and parallel I/O paths provided is indeed a tall order. It has not been pointed out anywhere in literature, how best to use the resources of such a system. When comparing these two computer systems (GE 645 and CDC 6600) one thing stands out. The GE 645 system architecture is such that it exhibits a 'fail soft' characteristic in operation. By this is meant that system performance is gracefully degraded on the failure of any system component (the system is not rendered inoperative when a component fails). This is not the case for the CDC 6600 system as it has only one CPU. However, unless a good user and system model (which include I/O activity) is developed, no comparison of performance of these two configurations can be made.

Interestingly enough, CDC has been the only manufacturer to produce a configuration like their 6600 system. All big systems produced by IBM, GE, RCA, UNIVAC and SDS follow the MULTICS architecture evolved by Project MAC.

In retrospect in this section the chief problems in evaluating performance of the CPU, main memory and the system configuration have been reviewed. Of the various techniques mentioned, the instruction mix, kernel and benchmark problem approaches have been elaborated on.

Section 2.2 Analysis of the operating system

The operating system is a growing collection of programs which are organised in what is called a system library. Some of these programs are executed by the system in the privileged mode whereas the rest are executed in the normal mode. The main interest here is centred around those programs ^{which} control the overall operation of the system and hence to a great extent determine its behaviour. Examples of these programs are the interrupt processing routines, the input-output control routines, the group of programs known by the general name of supervisor etc. etc. Some of these routines, the interrupt processing routines for example, have to reside in the main memory at all times. Such routines are commonly referred to as 'wired in' routines.

One of the functions of the operating system is to present a virtual machine to the user. For example, the user-machine interface is a set of higher level programming languages. In order to give such a facility to the user, the operating system has in it translator programs which convert user programs to a form executable by the hardware system. This necessary function of translation forms a part of the workload of the computer system. However, the CPU time required for this task is not accounted under 'overheads'. The importance

this function is accorded in the operation of the system depends on the nature of the user population. For instance, in a University computing service, translation of user programs could form a major part of the workload. So, it is an important aspect from the point of view of performance evaluation. It must be noted that these translator programs (appropriately termed as 'language processors') are executed in the problem mode.

The virtual machine performs some housekeeping and 'utility' functions for the user. For example, the system resources are made available to the user by a 'command language'. Programs to interpret statements in the command language and perform corresponding actions are a part of the operating system's supervisory program and are executed either in the privileged mode or normal mode depending on the function being performed. Also, the user is given a powerful means of filing information and editing it. Creation and maintenance of file directories, searches in the directory to locate an element, allocation of auxiliary memory space etc., are some of the house keeping functions to be performed by the system. In addition the system performs other housekeeping functions like maintaining a log of user usage of system resources. Computations performed for such tasks are a part of the 'overhead' incurred in providing these facilities. Therefore, an analyst interested in evaluating a computer system must

look into these features of the operating system and the capability of hardware in this direction. Once again the comment, that the nature of user population determines the importance of these facilities in the overall operation of the system, is applicable.

From the point of view of performance, the main features of the virtual machine are:

- (i) the strictly sequential execution of programs (despite the fact that there is parallelism in the operation of the system); and
- (ii) the provision of a virtual, large, addressable name space.

The first feature is made possible by the interrupt handling routines and input-output managerial routines. The second feature is a result of implementing segmentation and paging for the purpose of dynamic storage allocation. The resource allocation algorithms needed to accomplish the above are:

- (i) allocation of channels for I/O requests and
- (ii) paging schemes.

In addition to the above, processor allocation is necessitated by multiprogrammed operation of the system. A discussion of each of these resource allocation algorithms is given in the rest of the section. Computations done in connection with the resource allocation algorithms is a major

part of the 'overhead' incurred. The attempt should be to minimize both space and time required by these programs. This is a desirable objective as the 'overhead' is kept to a minimum.

Interrupt service routines and input-output managerial routines are probably the most difficult sections of the operating system in terms of their logic and coding. This is the general experience with simple batch processing systems. Lack of first hand knowledge about details of operating systems for multi-processor configurations, in which intricate protection schemes etc. are implemented, renders the comment to be somewhat incomplete in the general sense.

The time for servicing an interrupt depends on the type of interrupt and the operations needed to service it. The normal range of service time can be anything from a few machine cycles (i.e. a few micro seconds) to a few milli-seconds (the average page fault service time in the MULTICS system is a few milliseconds [34]). For example, service of interrupts caused by many abnormal program or machine conditions is quite simple (floating point underflow, floating point overflow, storage parity error etc). On the other hand, an interrupt due to say interval timer overflow causes an entry to the processor allocation routine. Depending on the allocation strategy and the operating conditions prevalent at the time of such an interrupt, the time for setting up the next program to take control of the CPU will vary. The minimum time required

is that for saving the 'status' of the current program and loading the CPU registers with the 'status' of the next ready program. Page faults or missing page interrupts cause an entry to the page scheduling routines. Here too, the time for service of such an interrupt depends on the complexity of the paging scheme employed. An interesting account of the detailed operations that occur in a multi-access system is given by Saltzer [16]. He sketches the types of interrupts likely to occur when a user is 'logging in'.

The input-output managerial routines offer to the user, many schemes for making use of the parallelism in input-output (I/O) operations that the system is capable of. Two main features of these routines affect the overall program execution times:

- (i) The buffering scheme employed. And
- (ii) the strategy for servicing I/O requests generated by programs being executed.

Buffering schemes available in IBM's Operating System/360 are outlined by Rosen [6]. User access to these buffers can be either of a sequential nature or of the direct access type. In order to save main memory space, I/O buffers are normally pooled together. This means that only as many buffers required at any time are created and are dynamically assigned to the I/O files. This assignment of buffers can again be either user controlled or be automatically taken care of by the input-output managerial routines.

Normally, when dealing with data in a buffer, the data is either moved to user work area (within his program) or the user is allowed to do the necessary computations within the buffer itself, i.e. the data is manipulated within the buffer and hence this reduces overall program storage requirements (no work area is needed within the program). Yet another way of getting around this problem is to allow the user to set aside in his program, a block of words equal in size to a I/O buffer, and this block of words is used as a I/O buffer or as a work area depending on the need for either.

Two types of buffering schemes are cited in literature [6]. First, the simple buffering technique, is the most flexible technique devised so far. In this technique, one or more buffers are assigned to each I/O file (data set in IBM System/360 terminology). An idea of how parallelism in input-output is used can be had from the explanation of the double buffering scheme explained below. To each logical I/O file two buffers are assigned; these buffers are large enough to accommodate the largest record read from the I/O file, they are assigned to. If the file is an 'input file', the I/O managerial routines issues one read command ahead of the actual input requirements of the current program being executed. That is, while the requirements of input data to the program are met from the contents of one buffer, the second buffer is being filled

with input from the 'input file' concurrently. On the other hand, for an 'output file', the contents of a filled buffer are written out onto the auxiliary storage while concurrently the program being executed, fills the next buffer with output it generates. Design of proper size of buffers to take maximum advantage of the parallelism in the action of hardware, has been discussed by Hellerman [35]. However, he assumes that an extremely good estimate of time for processing a logical record is available; something quite desirable but unrealistic in most cases due to the dependency of the path of computation on input data.

The second type of buffering scheme is known as the exchange buffering scheme. In this scheme, the buffers do not belong to the 'input files' or the 'output files' but subsume the role of an input buffer, an output buffer or a work area depending on the course of computation in the program.

The only model in literature, for a buffering scheme and its effect on performance of the system, is the one reported by Woodrum [36]. This study is of a floating buffering scheme, a case of simple buffering in which the buffers are pooled. The model has a lot of constraints on program conditions (one record of every input file must be available in some buffer or the other etc) and hence results reported are of limited value. This aspect of system operation has been

sorely neglected in the efforts to produce models (analytical or otherwise) for computer systems. That it is an important aspect needs no special emphasis; its inclusion in models for computer systems will lead to the solution of the problem of configuration evaluation.

The other function of I/O managerial routines, i.e. the strategies for allocation of available parallel paths to waiting I/O requests, has not yet been introduced into models for computer systems. One particular case of I/O requests generated due to the paging schemes has been modelled and will be discussed alongwith the problems of paging. Except for this one single instance, this problem has not been studied from the performance evaluation angle.

One of the things to be noted in connection with the study of this part of the I/O managerial routines is that no I/O is initiated directly by a user (especially because, almost all users program in a higher level programming language in which the user is given no control over actual operations but is allowed to specify logically his requirements of input or output and the positional relation between I/O and computation for the sequential execution of his program). In fact, the actual I/O operations depend on the buffering scheme employed. For example, the user program produces output

at the end of intervals of processing data. This output is filled into a buffer of the output file as it is generated. Only when the buffer is full, an output operation need be initiated. However, initiation of this operation is possible if a parallel data path (to that particular part of auxiliary storage where the output file resides) is free. And if a parallel data path is not free, this I/O request is queued with other I/O requests of equal priority and is serviced only when no other request of higher priority or preceding request of equal priority are present. A user cannot keep track of all these conditions in system operation. This fact is accentuated by the presence of many active users who generate I/O requests simultaneously. It is therefore the job of the I/O managerial routines to take care of all these conditions and service the user I/O requests accordingly. The I/O managerial routines are therefore carefully structured into several functional levels.

The user deals with the level which provides him a buffering scheme. Even this may be implicit when he uses a higher level programming language to express his computational procedures. In this case he simply indicates his requirements of input data or generates desired output. It is the responsibility of the language processor of the higher level language to generate or provide the necessary interface between the user I/O specifications and the I/O buffering system.

The buffering system generates at 'run time', calls to the next level of the I/O managerial routines. This level could for example locate the actual position of residence of the I/O file in the auxiliary memory, generate the hardware instructions necessary to accomplish the transfer of information, queue the request after processing it for priority, check for an available data path to the proper peripheral device and either service the request (if no other is waiting for service) or return control to calling program. Another level gets control on the occurrence of an event like the completion of an I/O activity (and hence a data path is now free). This level might check the operation for correct transfer, re-try in case of some hardware malfunction or transmission error, or start off service of a request waiting in queue. Service disciplines for such queues are quite simple. The requests are stored in tables with tags for priority operation. At most one request per peripheral device is entered into this table. Possibly, the entry position for each device is fixed in the table (at the discretion of those who prepared the initial operating system) and a 'top-down' search is made for the entry with the highest priority. This entry is then serviced (the corresponding I/O operation is initiated).

Naturally enough, these routines deprive user programs of 'ready' or 'active' status when they cannot cope with the volume of I/O requested by them (eg, both buffers are full for an output file and yet more output is to be generated or the program needs input and the buffers are empty or one is empty and the other is in the process of being filled - in all such cases, computation by the program comes to a halt and the program is said to be in an 'I/O delay' state or 'dormant' state). On such conditions, an entry to the processor allocation routine to find the next program to be executed, is caused.

The reason for taking a closer look at these functions of the operating system is that in order to be able to model such a system in its entirety, its detailed operation and relationships between its many facets must be understood clearly. It is common knowledge that the 'overhead' incurred by a large, general multiprogrammed computer system is about 30 to 50 percent of the total time the system CPU is operative. In the case of the time sharing computer utilities like the IBM System/360 Model 67 or the MULTICS, these overheads could be more than that (as quoted by Neilson [37] for the IBM System/360 Model 67 at Stanford University, California, USA). A part of these overheads is processor idle time as forced by I/O waits and such like. If such phenomena are to be understood in any measure of completeness and proper counter measures to be designed, an evaluative study or an effort in modelling

the computer system must include these details. However, the argument that usage of such systems is unwarranted, especially in the light of overheads in CPU time incurred, is not totally sound. Firstly, the benefits accrued from such system operation (the provision of powerful features in the virtual machine, like segmentation, for example) are not expressible quantitatively. And secondly, the CPU is only one of many resources available (an important resource no doubt) and at the cost of overheads in CPU time, the rest of the system components have increased utilization (and economically speaking, the rest of the system figures in a large way in the total costs).

Turning to the dynamic memory space allocation problem, Demning [38] has discussed and compared several page scheduling problems. Briefly, the problem of dynamic storage allocation problem can be explained as follows. The user programs in an addressable segmented name space. This means that specification of an address in an instruction is achieved by supplying a segment name and the displacement (page number, line number) within that segment (the set of all possible segments gives the total addressable memory of the virtual machine). Each segment is divided into smaller parts called 'pages'. Normally the size of a page corresponds to the size of a block of main memory (the main memory itself is partitioned into blocks of a convenient size). Normally, all of the segmented name

space (that is required by the active programs within the system) is stored on a fast back up storage medium like a set of drums.

The page scheduling problem is to bring pages of program residing on the drum, into blocks of main memory so that it is available when needed in the execution of that program. There are two ways of tackling this problem. The first is to anticipate the needed pages and initiate 'fetch' operations. This sort of predictive approach leads to several pitfalls which have been outlined by Denning [38]. The second method is to fetch the page from the drum when addresses in that page are generated by the program being executed. This technique is popularly known as 'demand paging' and is the technique used in all paging schemes. A pedagogical notation to explain the commonly used page scheduling algorithms is given by Gecsei et al [39]. Given the dynamic page trace (obtained by monitoring the execution of the program for effective addresses generated and the addresses of instructions in the instruction stream), a measure of effectiveness of a paging algorithm is the number of times a freshly addressed page is found in the main memory. The attempt is to maximize this number. This also means that the I/O traffic, caused by pages being brought into main memory from the drum or written out from main memory on to the drum, is reduced - a very desirable thing as all I/O is considerably slower than computation by CPU. It is to

be noted that there is no difficulty in knowing which page to bring into main memory as an address in that page will be generated by the program in execution, but the problem is to find place in the main memory where the incoming page is to be located. If the main memory is not yet full at that moment, then there is no problem in finding space for the incoming page. However, if the main memory is already full of pages, the job of the paging algorithm is to find a page which can be turned out. This page may or may not have to be dispatched to the drums depending on whether it has been written into or not (there is already a copy of this page in the drums).

Some of the common paging algorithms are the First In First Out (FIFO) scheme and the Least Recently Used (LRU) scheme. The attributes FIFO and LRU refer to the scheme for selecting the page to be turned out. The FIFO scheme is very easy to implement. Just a linear list of pages in core memory need be maintained. The top element of the list indicates the page first brought into the main memory (amongst those present in it). This page is selected to be turned out. As the scheme does not concern itself with actual references to the page during the time of its residence in the main memory, there is no attempt to maximize the success of finding a freshly referenced page in the main memory. Each page resides in the memory for the largest possible duration at the end of which

it is turned out without regard to the behaviour of the program to which it belongs.

In the LRU scheme, when a page not in main memory is referenced, a search is made to find the page that has not been referenced for the longest period of time. Conceptually, a clock may be associated with each page in memory. These clocks are reset whenever the associated page is referenced. When a page is to be turned out, the page whose clock has the maximum value is selected. In practice, a shift register is simulated in the entries for each page in the page table (page tables are maintained in order to aid in the mapping of addresses in the name space, generated by instructions being executed, to main memory locations in which the contents of the page reside). The left most bit of this shift register, known as the 'use' bit, is turned on whenever the corresponding page is referenced. On an entry to the 'page fault' service routine (a page fault is said to occur when a page not in main memory is referenced), the simulated shift register contents are shifted one bit to the right. The contents of all these shift registers are compared and the page associated with the shift register having the least numerical value is selected to be turned out. It is easily seen that a shift register of infinite length is necessary to implement the LRU scheme and

a shift register of zero length gives a scheme equivalent to the FIFO scheme. An interesting experiment on paging was conducted in the MULTICS system to determine a pragmatic length for the shift register [34] and a register one bit long was found to be an economic feasibility.

Another paging scheme is the Least Frequently Used (LFU) scheme. The page selected to be turned out by this scheme is the one that has the fewest number of references to it. Such an algorithm may be implemented in practice by a similar technique to that of LRU technique. A count of references to the page is kept in the appropriate entry of the page table and is used in selecting the page to be turned out.

Denning [38] observes that paging is no substitute for real core memory. Except in the case of the FIFO scheme, overheads are incurred due to techniques of their implementation. Very heavy paging occurs in worst case conditions (in varying degrees for different page turning algorithms). This can lead to instability of system operation (system 'thrashing' or system 'crash' as it is commonly called). Therefore, appropriate hardware need be designed to help in memory management or page turning.

In an effort to achieve economic operation of the system and implement a good page turning scheme, Denning proposed the 'Working Set' model for program behaviour [38]. The

working set is defined as the set of pages referenced in time $t - \tau$ to t , where time t is the present value of time in the execution of the program under consideration and τ is the 'backward window', an interval over which program page-referencing-history is accumulated. A page joins the working set if it is freshly referenced during the interval $t - \tau$ to t . A page, already in the working set, leaves it when it is not referenced for τ seconds. Pages in the working set are considered 'holy' and should not be turned out. And programs are not considered to be in 'ready' status unless at least the pages in the working set are in the main memory.

A quantitative comparison of overhead involved with each of the page turning algorithms is quite possible. Also, it is possible to estimate the size of the routine which carries out the necessary computation for selecting the page to be turned out for each algorithm. However, the success of the paging algorithm in terms of minimization of page traffic, is dependent on the nature of workload. Important design parameters of the paging scheme which depend on the workload are the following:

- (i) The optimal size of a page; a page should be of such size so as to contain effective loops of program code so that the control of program does not pass out of the page too soon - this reduces the number of fresh pages referenced.

Again, the pages should be of such a size so that the utilization of available space in it is a maximum.

- (ii) The magnitude of the 'backward window' τ or other like parameters for the corresponding algorithms. This parameter gives an indication of the extent of program-execution-history to be maintained. For example, in the case of the working set model, if τ is too small then pages pass out of the working set too soon and may have to be recalled soon after they have been turned out. On the other hand if τ is too large, then the working set grows too big and the memory requirements for the program to be in ready status also increases. This limits the number of programs that can be in main memory and hence imposes some restrictions on the number of programs that can be executed in a multiprogrammed fashion at a given time.

The fact is that there is not sufficient critical experience in the details of working of systems sporting such features, to authoritatively conclude about their performance (in terms of

the success of the paging scheme) or determine values for the design parameters mentioned above. Experiments in monitoring paging behaviour are now more frequently reported in literature. Statistics about segment sizes have been reported by Batson et al [40]. Randell has discussed storage Fragmentation (loss of storage utilization due to segmentation and paging) in detail [41]. Belady et al have given instances (found by experiments) in which there is a decrease in running time of programs as a result of decrease in size of main memory [42]! This only goes to show how little is understood of program behaviour and how difficult it is to produce a general model for it.

Finally, a last point of interest in connection with paging schemes. The utilization of the drums and the transfer path between them and the main memory should be a maximum. In general, the problem here is to sequence the list of pages, to be either read out from the drums or written into them so as to make maximum use of the drums. Denning's analysis [43] of this aspect (for a segment turning scheme instead of a page turning scheme) is applicable here. Two scheduling algorithms have been discussed. The first is the FIFO method. Here, fresh requests for pages, to be either read from or written into the drums, are added to the bottom of the list and this queue is serviced by doing the proper operations for the first page in the list. An obviously better algorithm is the Least

Access Time First (LATF) method of sequencing the list of requests. In the execution of this algorithm, the page nearest to the drum position (with respect to the read/write heads) is read out or the first empty space is written into. It is quite possible that though the drum is utilized to a maximum in this manner, certain pages are not read out for a long time. Therefore, in actual implementation a mixed policy may be followed. For example, if a particular page read request is not honoured for a specified quantum of time, then it is assigned a priority over other requests and immediately serviced.

The processor allocation problem has by far aroused maximum interest and prolific attempts to model this aspect is evident in recent literature as indicated in a survey of analytical models for time shared computer systems [44]. A survey of computer processor scheduling methods and counter measures adopted by users to gain favour of the system has been ably presented by Klienrock and Coffman [45]. Further, a pedagogical notation for a whole range of existing time sharing scheduling algorithms and introduction of a new interesting scheduling algorithm is presented by Klienrock [46].

When considering processor allocation algorithms, the important issues to be studied are:

- (i) how is the assignment of priority to users effected? And

- (ii) what is the service discipline used in selecting users from various priority groups for service?

In answer to the first issue, priorities can be bought (in disciplines where bribing is allowed) or earned (by programs exhibiting desirable characteristics) or deserved (due to knowledge about favourable characteristics of program prior to execution). Priority disciplines can be classified according to the properties listed below:

- (i) Pre-emptive vs non preemptive disciplines
- (ii) Resume vs restart for preemptive disciplines
- (iii) Source of priority information
- (iv) Time at which priority information becomes known.

This classification has been proposed by Klienrock and Coffman [45].

Some of the commonly found priority scheduling disciplines are the FIFO discipline, the Shortest Job First (SJF) discipline, the Round Robin (RR) discipline and the multiple level feedback (Foreground-Background, FB) discipline. Mostly implemented priority disciplines in processor allocation are minor variations or combinations of the above. Greenburger [47] has indicated the difficulties in studying even simple priority disciplines analytically. The measures of effectiveness of various processor allocation strategies are the system response time

for various priority groups (or turn around, in other words) and throughput achieved by each of them.

The FIFO discipline has already been explained in connection with paging schemes. As it is not any different in the case of processor allocation, no further explanation is necessary. Again, here too it is the simplest of the various processor allocation algorithms. In all cases, preemptive versions of these allocation schemes cause control to be transferred to incoming high priority programs; the running low priority programs are interrupted in the middle of the processor time slice allotted to them. The SJF scheme causes the shortest (in running time) program in 'ready' status to be executed next. Such a discipline would be particularly useful in a batch processing environment in which users can be assigned priorities prior to submission of their programs, depending on their requirements of processor time. In a multiprogrammed time shared environment, user identification of type of job would be necessary for effective implementation of this scheme. Compared to the FIFO discipline, short jobs clearly have higher priority and faster turn around with the SJF discipline. Long jobs are seen to suffer long waiting times before service. There is no appreciable increase in overhead over that incurred in the FIFO queue, due to the SJF strategy of operation. Hence throughput is the same

for both the FIFO and SJF disciplines.

The round robin scheme (RR) is also seen to favour short jobs and is normally used in time sharing systems. An analysis of this scheme for the CTSS operating conditions, has been presented by Scherr [48]. In this scheme, each job is run for a fixed quantum of time. If the service of the job is complete within this allocated quantum of time, the job simply leaves the system. Else, it is cycled to the end of a single queue of waiting jobs that need yet one or more quanta of time for completion. It is observed that an infinite quantum renders the RR scheme equivalent to the FIFO scheme.

As the quantum is reduced to near zero, the available processor time is divided equally amongst all jobs in the system. In effect, there will be no waiting line and all jobs will be executed in parallel with an effective speed proportional to the number of jobs in the system.

The FB discipline is a little more intricate than the RR discipline. In the implementation of this scheme, the system may be viewed as one with multiple queue-levels numbered 1, 2, 3, ... New arrivals are put in queue-level 1 and executed for a quantum of time as in the RR scheme. After a quantum of service the incomplete jobs are pushed to the next higher level. Jobs in the higher levels are taken up only when a lower level is completely empty. Though the throughput without the FB scheme is lesser than that obtained with the RR scheme,

long jobs are not allowed to interfere or delay excessively the execution of short jobs.

This section and the preceding section cover the study of computer systems from the point of view of performance evaluation. The material presented should provide the necessary pointers in modelling the system as well as its workload.

Section 2.3: Analysis of workload

A point of note at large is that there has been a tremendous increase in the application of computers to solve problems occurring in all forms human endeavour. Computer results are therefore increasingly used everywhere. It is indeed alarming to see such faith in computer output without really having any theory for estimating the reliability with which correct results are produced. That is, there is yet no defined procedure in the offing, which determines with a measure of confidence that a program is well and truly debugged. There have been efforts to produce higher level programming languages (which are universal in computation) in which completeness of logic can be checked out [49] and ambiguities in logic that arise in the execution of programs can be resolved or pointed out at run time. These are indeed very desirable characteristics for programming languages and should be emphasised in their design. Programs constitute the workload of computer systems. And programs are engineering implementations of algorithms developed for solving problems.

A pragmatic analysis of algorithms (and hence computer programs) should take into consideration the following:

- (i) The fact that computers work only with a finite range of numbers in the integer domain (an approximate notation for real numbers - the normalised floating point number notation - is used and such numbers can be mapped onto the set of integers). Investigations in the theory of computation, to produce schemata for representing programs, always examine the behaviour of programs for all input data possible in order to deduce results about their properties (for example, will the algorithm terminate for all inputs). Such an effort has led to some general (but negative) results. Like, there is no algorithm to decide whether any given algorithm will terminate for all input. Such a result is very useful no doubt as it helps in delineating the solvable problems from the unsolvable problems. In an effort to be more positive, attention has been drawn towards finding properties of subclasses of algorithms. For example, what is the set of algorithms that terminate for all inputs. If something very useful has to be said about computer programs, the restriction on the set of all input data to

the set of numbers representable internally in a computer (i.e. finite bound on numbers depending on memory word size etc) should be given due emphasis in the study of their properties.

- (ii) Writing programs is indeed like making machines to automate certain procedures. Therefore a great deal of care in their design and analysis of problem being solved and procedure for its solution is definitely warranted. Moreover, procedures for solution of problems need not be unique. The analysis of such procedures, with a view to find the range or type of data for which they are applicable and efficient, (in terms of the number of steps necessary to arrive at the result) is important. So is the comparison of storage requirements necessitated by each algorithm. The sensitiveness of procedures to input data is all the more an intricate problem in the case of numerical techniques of solution to the problems of classical mathematics. Extensive usage of such techniques blossomed with arrival of computers. It is not uncommon to see people rush to apply these techniques for solution of their problems by using 'ready made' programs

without a careful analysis of the effectiveness of the technique employed by the program in relation to their problem and associated input data. The result is an unpleasant tangle with the idiosyncrasies of the computer used. To illustrate, nobody would knowingly use equipment in the range of operation it was not designed for. The effects could be materially disastrous, something to which immediate heed is paid. Similarly, the result of using a numerical technique unsuited for range of data presented by a particular case of the problem could be disastrous too. The outcome of such usage of computers can be subtle and hence unnoticed or just incomprehensible; in any case it is quite dangerous.

Effecting a transition from algorithms to computer programs is an art in itself. In all seriousness, computer programming is no tool for a high school drop out as is commonly and sometimes derisively opined. It is all the more distressing to note disclaimers tagged on to programs distributed by user groups etc which state that the originator of the program is not responsible for its malfunction for particular types or values of input data, but he would be happy if any bugs found in his program were communicated to him! The user of

such programs is expected to conduct his own analysis with whatever information which can be gleaned from the documentation. Programming seems to need a peculiar acumen which is unique by itself.

A seemingly harmless algebraic expression could run into problems of numerical instability in its computation for some ranges of data, simply due to the manner in which the expression is coded in the programming language used. Being alive to these problems and being able to work harmoniously in ^{the} environment of the computer cannot be emphasized enough. Efficiency of coding is attained by proper factoring and sequencing of expressions and terseness in expression of logic of the program (for example, involved logic can be neatly expressed with the aid of Decision Tables). It is to be noted that normally the coding of programs imposes limitations other than those inherent to the algorithm it implements, which may or may not be easily stretched. Examples are: the insensitiveness to accuracy beyond a certain self imposed limit, of input data; the accuracy with which results are reported; the maximum size of the problem that can be tackled etc.. This further restriction, either artificial or forced, alongwith the rest already listed, should provide the grounds for

an estimate of what can be practically feasible with programs.

The above digression outlines factors to be considered for 'software evaluation'. These ideas have been in the air for sometime. Only, there is a need for a massive effort to be positive and to progress in this direction. In view of the increasingly important role that computers play in day-to-day matters, such investigations are essential. Corbato has analysed touchy or sensitive managerial issues in the design development of large software systems [50]. The success of such efforts depends a great deal on the documentation of the software system. He states that the success of CTSS and the development of MULTICS at Project MAC was chiefly due to the attention given to this problem. In conclusion, the main thing in 'software evaluation' is to enforce a quality control and standardisation of programs that are marketed.

Coming to the main problem to be tackled for computer system performance evaluation purposes, static and dynamic characterization of workload has to be done to this end. Traditionally, static characterisation of programs is achieved by use of flow graphs or directed graphs or connectivity matrices and so on. For a great part, these schemata try to portray the logic of the program and computations which affect the logic. Investigations regarding equivalence of programs naturally need a formalism with which the detailed nature of computations made by the program

can also be specified. For the purposes of performance evaluation, the static characterisation of workload should portray the following

- (i) Control logic of the program: For this purpose, the schemata already available should suffice. Information that can be obtained from this representation of programs is useful in properly segmenting them with a view to retain control within each segment for the maximum possible time in a pass through the program. A point of note is that this information can be readily obtained from the translation stage. As the interest is in characterising a workload rather than individual programs, a statistical measure for this aspect of programs that constitute the workload is necessary. To this end, let α be the number segments in a program. Then $f(\alpha)$ gives the probability distribution function for initial segments in programs and $E(\alpha)$ gives the mean.
- (ii) The size of static programs: This is an indication of the initial size of segments of a program prior to its execution. The size can be expressed in terms of the basic unit of storage in the main memory (bytes, for example). Statistics about this information can be collected during the translation phase. Two factors contribute to the initial size of programs. One is the program code which is normally fixed in size, even during their execution. And the second is the

amount of data area and its associated data structures which need not be fixed and can grow during the execution phase of programs. In such a situation, an effort to separate program code and data area and assign them to different segments can be expected. Therefore, let β be the segment size and $f(\beta)$ and $E(\beta)$ give the probability distribution function of β and its mean respectively.

- (iii) I/O requests: Two static qualities ^{are} of interest here. The first is the number of logical I/O files a program uses (< input file, maximum input records >, < output file, maximum output records >, < work file 1, ∞ records > < work file 2, ∞ records > ... - the use of work files may be obliterated by use of segmentation and paging techniques and is included here only for the sake of generality) and second is the buffering technique employed (<access method, buffering scheme>- the valid combinations were indicated in the preceding section) Let μ and ν be the number of I/O files and buffering techniques used respectively. Then, $f(\mu)$, $E(\mu)$, $f(\nu)$, $E(\nu)$ provide the necessary statistical measures. It must be noted that similar measures are necessary for the values of maximum input and output records for the input and output files.

(iv) Nature of computation: This perhaps is the most difficult parameter to characterise as already seen in section 2.1. For the present purpose, a finite set ϕ' of parameterised kernels will suffice. Generalisations at this stage being tantamount impossible, this will be an initial limitation to work with. Let ϕ be the kernels used by a program, so that $\phi \subseteq \phi'$.

A macroscopic description of a program i is given by the sequence of job steps or tasks $X_i = x_{i1}, x_{i2}, x_{i3}, \dots$ required to service its job request. In the case of batch processing systems, this sequence could be something like - compile routine 1, compile routine 2, ..., load program, execute. In the case of time sharing interactive systems, each user request may be just one job step. The sequence in which job steps are executed is of course important for the proper execution of the job. Each job step is characterised by the static attributes of programs that have been discussed so far and other static and dynamic attributes yet to be discussed. The sequence of job steps required by a job can be obtained from the transition matrix Δ whose elements δ_{ij} indicate the probability of job step j following job step i . Δ is therefore a characterisation of the nature workload in a gross sense. The starting step is always the same.

Lastly, let ζ be the user's estimate of time needed to service him. And so $f(\zeta)$ is the statistical measure for this attribute of the incoming program.

To summarize, the static characterisation of a program is given by the 7 tuple $\langle \alpha, \beta, \mu, \nu, \phi, \chi, \zeta \rangle$. And the workload is characterised by probability distributions of $\alpha, \beta, \mu, \nu, \zeta$ and the set ϕ' and the transition matrix Δ . It would be providential if the dynamic characterisation of the workload could be obtained from the static model. However, this is not the case at present and therefore the dynamic behaviour of the workload must be considered in detail.

Dynamic attributes of programs, important for a dynamic model of the workload are:

- (i) page trace of programs
- (ii) density of I/O requests and the volume of I/O traffic
- (iii) processor time for completion.

It must be noted that these processes have to be tied down in the time scale of program execution.

The notation used here for the addressing pattern in terms of the pages referenced in time (page trace) by program i is adapted from the one developed by Gecsei et al [39]. Let θ represent the page trace generated by a program so that $\theta_i = \theta_{i1}, \theta_{i2}, \theta_{i3}, \dots, \theta_{it}$, where θ_{i1} is the page referenced at program execution time $t=1$ and is drawn from the initial

set of pages in the program. The initial set of pages in the program has a magnitude of $[\alpha_i \times \beta_i]/k$, where k is the page size specified as a number of basic storage units of main memory and α_i, β_i are obtained from static characteristics. This number may or may not grow depending on the dynamic storage requirements of the program, a phenomenon which can be statistically described if the need arises. For the present, the storage requirements magnitude-wise of a program will be assumed to be static. Therefore θ_{it} here is the page referenced at time t in the execution of the program and is drawn from the set of pages defined earlier.

Dynamic attributes of input-output requests of program i are:

- (i) the I/O file selected for use
- (ii) type of operation (read, write, control); and
- (iii) the magnitude of the request (number of characters transferred etc).

To this end, let Ω_i be the logical I/O request trace of program i . So $\Omega_i = \omega_{i1}, \omega_{i2}, \dots$ where ω_i 's are the I/O requests of program i and the second subscript is just the sequence number of the request. The definition of ω_{in} is given by the ordered triple $\langle \rho_{in}, \delta_{in}, \epsilon_{in} \rangle$, where ρ_{in} is the I/O file selected by the request and δ_{in} is the type of operation (control/transfer) and ϵ_{in} magnitude of information transfer. Of these, ρ_{in} is drawn from v

(using an appropriate discrete probability function which characterizes this dynamic behaviour of program i), σ_{in} is given a value of 0 (depends on the probability of a control operation) or 1 (depends on the probability of an information transfer operation) and ϵ_{in} is given value from a probability distribution of the length of I/O requests.

It is very difficult to characterise the dynamic nature of computation done by the CPU. The normal trend has been to concentrate more on the amount of CPU time a program needs. Statistical description of this factor is given by $f(\xi)$ where ξ is the CPU time required by the program. An exponential distribution of ξ is not unusual especially because short jobs are accorded priority in most processor scheduling algorithms. Another important factor is the arrival pattern of job requests. Let γ be the inter arrival time of jobs and hence $f(\gamma)$ characterises this aspect of the workload.

At this stage, it is necessary to tie down page referencing and I/O request generation with the central processor execution of programs. Page traces can be generated from a transition matrix Ψ_i whose elements ψ_{ij} give the probability of page 1 referencing page j . Associated with this matrix is a vector Π_i whose elements π_{ir} (ranging over the set of pages $\{\alpha_i \times \beta_i\}/k$) give the expected time (per unit) for which program control remains within that page. That is to say, $\sum_r \pi_{ir} = 1$ and $\pi_{ir} \leq 1$ for all r . For every instruction executed while the

program control remains within a page (i.e. for page r , the fraction π_{ir} of ξ_i), the page references can be obtained from the appropriate row in the transition matrix.

In the case of I/O requests, it is much easier to put their occurrence in program execution time. Intervals of processing are followed by I/O requests. Inter arrival time of I/O request, say λ_i , is characterised by the probability distribution function $f(\lambda_i)$.

The dynamic model of workload explained above is still crude and needs much more refinement. The chief attempt in the model is to put together several processes that characterise program behaviour. This is in contrast to the general approach of studying one feature of the computer system at a time so that only one feature of program behaviour need be modelled at a time.

Instead of characterising the whole workload of the system in the manner presented here it would be more fruitful to characterise user workload and the workload formed by system programs separately. The usefulness of this approach would be in determining the overheads incurred. Also, system programs (which are executed in response to the job steps specified or depending on the logic of system operation) are peculiar in themselves and such a characterisation would help in understanding their need for computer resources.

Some of the ideas used here in characterising the workload are taken from already available literature [37, 38, 39, 40, 43, 47, 48]. But it is difficult to ascribe motivations for various aspects in the characterisation to particular works seen in it. This is because of an attempt to take a very general stand in modelling all of the workload.

Section 2.4: Conclusions

In this chapter, critical analysis of various resources of computer systems and various processes that occur in the operation of the system are first presented. These analyses culminate in an attempt towards a proposal for a methodology for quantification of workload of computer systems. As the problem of performance evaluation necessitates the development of techniques to relate the demands of the workload to the capabilities of the system, models for computer systems should be developed.

A tutorial survey of analytical models for computer systems is presented by Radhakrishnan et al [51]. One thing stands out as a result of this survey. All these models portray just one of the many processes that characterise system behaviour. Hence any inter relations between these processes are just ignored. One of the outcomes of such procedures of analysis is the comparison of several techniques of management for some of these processes. However,

proper prediction of system behaviour is not possible with the aid of these models as the inter relations mentioned are not taken care of.

A static and dynamic model for system behaviour can in principle be specified with the help of all the information given in sections 2.1 and 2.2. Section 2.3 helps in indicating the detail of the model. As the logic of system operation (and hence the dynamic behaviour of the system) is very intricate, decision tables could be used in specifying it. Besides aiding in a concise and cogent presentation of the logic, this technique proves very useful in checking out its completeness. The complexity of the model makes already available analytical techniques of queuing theory and Markov models (these are the most commonly found techniques used in the modelling of computer systems) quite insufficient in handling the problem. Simulation techniques, explained in the next chapter, are the only other available tools. Even here, the extreme detail of the model makes the use of simulation techniques inadequate.

The main point is: in the execution of a program, other things like page referencing and I/O requests also occur. In view of this, is it justifiable to allocate processor time to a program without any regard to the other phenomena? Denning has made a beginning with his 'working set' model

for program behaviour which accounts for page referencing in program execution time. This concept should also be extended to include I/O behaviour of a program.

CHAPTER III

SIMULATION TECHNIQUES FOR COMPUTER SYSTEM EVALUATION

Section 3.1: Introduction

Simulation techniques can be broadly divided into two classes. Each class is characterised by the type of system being simulated. The two classes of systems are the class of continuous time systems and the class of discrete time systems. Computer systems belong to the latter category, and therefore, only the techniques for simulating discrete time systems are considered in this chapter. In general simulation problems are characterised as being mathematically intractable and having resisted solution by analytic means for a long time. Simulation techniques can no longer be considered as last resort techniques for studying any system. In fact they have come to be the only techniques available for studying large behavioural systems in all complexity. In the ensuing discourse 'systems' will be understood to mean discrete time systems and 'simulation' to mean simulation of discrete time systems.

Discrete time systems are idealised as network flow systems that are characterised by the following:

- (i) the system consists of 'components' or 'elements' each of which performs a prescribed function;

- (ii) interconnections between various components are specified, giving a 'structure' to the system;
- (iii) 'items' flow through the system from one component to another requiring the performance of a function (service) at a component (facility) before the item can move to the next component;
- (iv) the structure of the system governs the flow of items;
- (v) the components have a finite capacity to process items and therefore items have to wait in 'queues' before reaching a component.

Normally, the objectives of studying discrete systems are to study the steady state capacity of the system. For example, consider a simple single server queuing process. The server is a system 'component', the customers waiting to be served are 'items' that flow through the system and the queue discipline is 'first in first out' (FIFO). Given an arrival pattern for the customers and the service time distribution for the server, the steady state server busy period distribution and the customer waiting time distribution and the queue length distribution are to be obtained.

A simulation study of this type mostly deals with keeping track of where individual items are at any particular

instant of time, moving them from queues to components, timing them for processing by components and removing processed items to other queues. The result of a simulation 'run' is a set of statistics describing the behaviour of the system over simulated time.

The study of complex behavioural systems, using techniques as outlined above, has become feasible only due to the advent of large digital computers. Therefore a large class of users are motivated to obtain solutions to problems hitherto untried or solved unsatisfactorily after making gross simplifying assumptions. The computations and other manipulations needed for simulation work being quite different from those required by 'scientific' computations or business data processing applications, a new set of languages tailored for programming simulation problems have been invented.

Section 3.2: Introduction to simulation languages

Before discussing and comparing some simulation languages, an introduction to various terms and concepts used in simulation is in order. When simulating discrete time systems [52], the operations of the system are considered at discrete points in time. These discrete points in time are called 'events' in simulation parlance. In the framework of systems presented earlier, as network flow systems, events occur when an item leaves a component having been processed by it or when an item

joins a queue at a component or when a component starts processing a fresh item. The time intervals between events are called 'activities'. The occurrence of a event changes the 'status' of the system at a discrete point in time. The system is enclosed within a 'boundary' and all that is outside the boundary is called the 'environment'. The environment can influence the system but the system cannot influence or alter the environment in any way.

The simulated system is a collection of 'entities', 'attributes of entities' and 'sets or files of entities' [53]. Any type of unit independently identified in the system is called an 'entity'. The server in a queueing process is an entity. However, he is present throughout the simulated time of the system and hence such entities are termed 'permanent entities'. The customer who waits in a queue and departs when serviced is a 'temporary entity' as information about him need not be retained once he is served. Each type of entity is in turn described by 'attributes'. The attributes of a server can be the time he takes to service a customer. The attributes of a customer can be the time of his arrival and the priority he wields over others in queue. The 'status description' of a simulated system may comprise of any number of entities or even many entities of the same type. Entities are said to be of the same type if their attribute names are identical; the values of the attributes can

be different. Inter relation between entities may be depicted in the status description by grouping entities of the same type into 'sets' or 'files'.

A simulation run is said to terminate or come to a finish when the system assumes or arrives at a particular status description or the simulation lasts a predetermined period of simulated time. For a more complete discussion and illustrative examples to explain various terms, concepts and techniques for conducting a simulation study using digital computers, one is referred to introductory texts and programming manuals of SIMSCRIPT, GPSS III and GASP [53, 54, 52]. Further work on on line simulation languages is given by Greenburger [55] and Buxton [56].

An excellent survey and comparison of simulation languages is given by Teichreow and Lubin [57]. As GPSS III and GASP II A are the only simulation packages implemented at IIT-K, some details, observations and criticisms are given here.

GASP is an acronym for General Academic Simulation Program. GASP II A is not a simulation language. It is a set of FORTRAN subroutines which afford the user a basic set of functions needed to accomplish computer simulation of any discrete time system. Simulation in GASP is essentially an event based study of the simulated system. By this is meant that the simulation of any system is achieved by scheduling

some sequence of events in simulated time. The GASP user, who really programs in FORTRAN, is expected to generate or supply the event schedule and collect statistics about the system status description at points of his interest. Therefore, the GASP user is expected to know a great deal about the basic functions provided by GASP and also about the system he simulates. As the level of programming a simulation model in GASP is low, the user is also given a great deal of flexibility in building the simulation model. For instance, FIFO and LIFO queue disciplines are natural when using GASP and priorities in queue disciplines are very easy to handle. Markov processes can be directly simulated. Besides queuing problems, PERT and CPM networks and inventory systems can also be simulated with the help of basic functions provided by GASP. Any GASP function can be altered by the user to suit his special needs.

The timing mechanism in the executive routine of GASP directly advances simulated time to the time the next event is scheduled to occur. The fact that only one event is examined at any point in time makes the GASP simulation technique conceptually simple. That is, no attempt is made to retain any of the inherent parallelism in the system being simulated; computations required at each event time are done in a sequential fashion. If two events happen to occur at the same point in simulated time, computations required for each event are done one after another in the order scheduled by the user.

Facilities to initialise a system model and conduct multiple runs for a model are available. As all programming by the user is done in FORTRAN, the user can build the simulation model in a parametric fashion and study the sensitivity of the system to change in certain parameters without any reprogramming effort. However, even to simulate a simple queuing situation, the GASP user has to become very familiar with most basic functions of GASP. This seems to be the main drawback in GASP.

GPSS III on the other hand is a higher level language for simulation work and is highly adapted to modelling of queuing situations. When using GPSS, models for systems are constructed as flow diagrams using standard GPSS III 'blocks'. Coding GPSS programs is done by simply transcribing the blocks in the flow diagram in a sequential manner. Parallel paths in the flow diagram are identified by labels in the code produced. Items, called 'transactions' in GPSS terminology, can be created and destroyed by the user and 'flow' through the model from block to block as specified by the flow diagram. Certain blocks, like QUEUE, DEPART, SEIZE etc., are event blocks and change the status of the simulated world whenever transactions flow through them. Statistics about transactions, as they enter and leave such blocks, are automatically kept by GPSS. So are statistics for the behaviour of permanent entities; in GPSS III these are blocks of the form FACILITY, SAVEVALUE etc. An irksome feature of GPSS III is that no computation can be done

at all unless a transaction flows through some block. The shortest operation performed in the simulated system is considered as a unit interval in simulated time and all times for other operations are expressed in terms of this unit. This is a cumbersome artifice a GPSS user has to put up with. As in GASP, GPSS updates simulated time to the time of the next most imminent event.

To simulate the flow of transactions through the system model, transactions are grouped together in different 'chains'. The 'current events chain' holds transactions in 'active scan status' or 'delay scan status'. The overall GPSS scan routine will always try to move transactions in active scan status into some block as dictated by the logic of the model. The 'future events chain' holds transactions being processed by a FACILITY block or any of its like. The user can control transactions by entering or removing them from a 'user chain'. GPSS III simulation is carried out on an interpretive basis and hence is slow.

Both GASP II A and GPSS III have built in trace features and this aids in debugging programs. In both programming systems, modelling of large and complex systems is equally difficult.

Section 3.3: Motivation for simulation of computer systems

Simulation of computer systems with a view to assess their performance is not at all a new phenomenon. Neilson [58] gives a brief history and account of use of simulation techniques

to study computer systems. However, this information needs to be updated as much work has been done in this area since his survey.

One of the earliest usages of computer systems to simulate computer systems was to check out the logical design of large digital systems and compare several logical designs of large digital systems. The need for simulation arose from the inadequacy of available analytical procedures to deal with digital systems of such magnitude. This sort of work was mainly done by the manufacturers of computer systems. The technique proved useful enough to warrant the design of special purpose computer languages for easy specification of such problems [59]. An example of such a language mentioned by Neilson is LOCS, an acronym for Logic and Control Simulation.

Soon enough, computer applications outgrew the capacity and capability of existing computer systems. Design of large computer systems with special hardware, for local and non local concurrency in operation, was contemplated [60]. As no great technological advance was forthcoming, the break through came when new logical organisations for computer systems were proposed. Detailed design of several system components was not possible till a number of system criteria were established. The complexity of the system made analytical studies difficult. Trade off factors for arithmetic speeds, instruction unit speeds, memory speeds and many other parameters had to be established.

To determine the effectiveness of hardware features introduced, timing simulation programs were written.

The input to the timing simulation programs were typical instruction streams representing projected applications of that time. The timing simulator did not actually execute instructions in the arithmetic sense, but timed the execution of instructions by the proposed logical organisation. The output of the program was the total time taken by the proposed system to complete the given tasks. Bucholz [60], in his lucid account of the design of Project Stretch, shows how such simulation programs were used to fix up various design parameters of the system. The most interesting results are the relationships obtained between the various parameters with respect to computer speed.

Upto this stage in the development of computers, the only yardstick for measuring performance of computer systems was the capability of their 'hardware resources' to meet the demand of typical and projected applications of the day. The term 'hardware resources' refers to the computational power of the CPU and the effectiveness of overlap between I/O operations and computation by CPU. In most simulation experiments, as in those conducted for Project Stretch, the computational power of the CPU was the only factor examined. Even so, timing simulation techniques were resorted to for lack of any substantially better tool than an instruction mix.

The only simulation model seen in literature which attempts to study the second factor mentioned above is a simulation program called SCERT. It is considered to be an improvement over the benchmark approach to computer evaluation. SCERT, an acronym for System and Computer Evaluation and Review Technique, is a product of Compress Incorporated, of Washington DC, and like benchmarks it is the brain child of a firm of computer consultants. The result is that very little is known about SCERT except that it is a five phase program which expects a very detailed description of user workload and the computer system of interest from the user. In the long list of things SCERT can do for a prospective customer of the computer market, there is a claim that, for a specified workload, it can evaluate configurations of any given system [61]. It is contended here that the input information to the program about I/O load is insufficient for configuration evaluation purposes. Discussion of this point is held over a later chapter.

With the arrival of operating systems, ideas about computer system evaluation have changed considerably. The computer system, as seen by a user, is no longer just a piece of hardware but an integrated system of hardware and software resources. The resource allocation problem alongwith the interrupt system made the behaviour of computer systems very complicated. As a result even building of simulation models has become a very difficult task; perhaps as difficult as building the

operating system itself. The difficulties faced in modelling and simulating multi-access multi-processor computer systems is given in a later section.

Analytical models attempted range from simple Markov models for batch processing systems to multi server multi queue models for time shared multi processor computer systems. Some of these models have proved to be quite accurate in their predictions about the behaviour of the system modelled. Most analytical models are constructible only after simplifying assumptions are made about the actual system. This is deemed necessary as mathematical analysis is not possible otherwise. Also, due to the complexity of the system, the detail is lacking in the models. All these factors lead to the conclusion that simulation techniques, albeit expensive, are the only tools available presently for studying large and complex behavioural systems like the third generation computer systems.

Section 3.4: Classification scheme for various simulation models

The variety and number of simulation experiments reported in recent literature are numerous. A classification scheme is proposed here so that a uniform appraisal of this work is possible. Before explaining the classification scheme, an analysis of features of computer systems which yield dividends in terms of improved performance when studied with the help of simulation models, is considered.

The management of computer systems can be considered akin to a traffic control problem. It is the work of the operating system assisted by appropriate hardware features to deal with various traffic congestion problems that exist within a computer system. Such circumstances are seen to stem from two basic aspects of computer systems. The first aspect is the fact that there are inherent speed mismatches between various components of the computer systems. The second aspect is that the operating system gives the user a logical environment within which he may conduct his work and this environment transcends any of the actual physical bounds of the system itself.

The design of some features of hardware and software for any computer system partly depends on the various schemes built into the system which try to bridge the speed mismatches or shield the user from the physical nature of the system. For example, the I/O managerial routines which are a part of every executive or supervisor of any operating system, shield the user from the actual nature of the configuration and the physical characteristics of the I/O device that he might chance to use during the course of his computation. These routines also attempt to make much use of the parallelism between I/O operations and CPU computations by queuing I/O operations and servicing them as and when the data paths to the I/O device are free. In short, these routines

offer a logical environment for programming I/O operations and try to bridge the memory speed I/O speed mismatch. Another example could be the dynamic storage allocation feature of the operating system. The user is given a very large virtual memory for his programs. Real core memory space is allocated to him as and when his program needs it. These operations are transparent to the user.

The point made is, the improvement in speed of components due to technological advances is nearing a saturation and so are their physical capacities. Therefore most improvement in performance has been brought about by better and more flexible ways of interconnecting these components and attempting to make programming as independent of physical boundaries of the system as possible. It follows naturally that most simulation experiments have been directed towards examining logical organisations within the system which attempt to bridge speed mismatches or shield the user from physical boundaries of the system.

Most simulation experiments can be classified under the following categories. The simulation models study

- (i) schemes which bridge the CPU speed - main memory speed mismatch,
- (ii) schemes which bridge the memory speed - I/O speed mismatch,
- (iii) schemes which bridge the user arrival rate - computer system service rate mismatch,

(iv) schemes which offer the user a large virtual memory.

As mentioned earlier, the number of simulation experiments reported in recent literature is very large and scattered over a wide range of technical and non-technical journals. As it is not possible to collect and classify all of them, only a few successful simulation models are given here as examples of each category.

The first category, aimed at the CPU - core memory mismatch, consists of simulation studies which examine closely the following type of logical organisations. System parameters studied by the models and the input and output information of simulation programs for them are cited against each case.

(i) Partitioning the main memory into blocks and interleaving addresses (consecutive) between them could be one strategy for reducing the speed gap. Simulation models for studying such logical organisations have tried to study system parameters like the blocksize and number of blocks, speed of main memory and the way instructions and data should be distributed in the main memory so as to make the most effective use of such strategies. Input to simulation programs for studying such logical

organisations is a dynamic instruction trace of typical programs and information about the data storage allocation techniques used by language translator which produced the object code for them. The output of the program could be the computer speed as a function of each of the system parameters being examined. This could help in fixing up values for each of these parameters [60].

- (ii) Another strategy could be to interpose a fast 'scratchpad' memory between the CPU and main memory. Parameters of design studied by simulation models for such a scheme [62] are the size of the scratchpad memory and the algorithm used for loading this scratchpad. Initial work for conducting a simulation study of this type is to obtain a dynamic trace of typical programs which form the workload and analysis of loops in these programs. The outcome of such a study could be the optimum size of the scratchpad and the best replacement algorithm.

(iii) An extension of the above strategy could be an increase in the size of the scratchpad so that it is comparable to a block of main memory. An example of such an organisation is the ^{IBM} System/360 Model 85 which has a 'caché' memory of 16K bytes and a 80 nano second cycle. Simulation techniques were used to determine the size of caché, choice of replacement algorithms [63] and improvement in performance [64]. An interesting result was that the performance of a system with this caché memory was 80 percent of a system with a large main memory of 80 nano seconds.

Most work on study of such schemes is reported under the study of memory hierarchy.

The main memory - I/O operation speed mismatch and schemes devised to get around this problem have hardly been studied by simulation techniques. This fact does not undermine the importance of the problem as such schemes could directly affect the total CPU idle time. The main steps taken to reduce the effect of this mismatch are to use buffering schemes provided by the I/O managerial routines of the executive and overlapped data channels. The problem posed is; how to use such features most effectively? So far, all that has been done is to provide such features and hope for

the best. A more detailed look at this problem is given in a later chapter.

The effects of the third category, the user arrival rate computer system service rate mismatch, are reduced by the introduction of multi programming and priority levels of users for the conventional batch processing systems. Ideas of time sharing and multi processing reduce the effects of this mismatch even more. Simulation studies of such features have tried to compare the task or job scheduling algorithms. For example, Scherr [48] has tried to compare the CTSS scheduling algorithm with the round robin scheduling algorithm. Saltzer [16] has considered the allocation of processor resources as a traffic control problem. Priority assignment to users, shortest job next for example, and its effect on the response and service time for each priority class has also been studied.

When considering the implementation of the dynamic storage allocation feature by the operating system, two scheduling problems are evident. The first one deals with the following decision problem: what page or segment in main memory should be turned out next and dispatched to the drums or disks? The second scheduling problem is, given the pages or segments to be brought into main memory from the drums or disks, find the sequence which makes the best use of the drums or the disks. The fourth category deals with the above problems.

Simulation work on this problem has been mainly directed towards finding the best algorithm for page turning or segment turning.

The problems of allocation of main memory space and processors have been linked together by Denning [38]. Information about the workload, required by simulation model for simulating these aspects of the system, is of the following type. Inter arrival time distribution of jobs or requests for each priority class, probability distribution of time required to service jobs or requests from each priority class and storage requirements by jobs or requests are the necessary pieces of information for conducting such experiments.

For more examples of simulation experiments, one is referred to Bucholz's bibliography on computer system evaluation [65].

At this juncture, it is noted that each of the indicated class of simulation models, examines the behaviour of computer systems at one particular level of detail. To amplify, the behaviour of the computer system is not studied in its entirety but what is examined is the manner in which each of the several aspects of computer systems affects the performance of the CPU or the system as a whole.

There have been a few attempts to study a complex time sharing system like the IBM System/360 Model 67 in great detail [37, 66]. Here too the effect of I/O load

created directly by the user have not been examined. In spite of this simplification there are numerous difficulties in modelling such systems and the next section is devoted to a brief discussion of problems with such simulation models and program difficulties that arise in them.

Section 3.5: Difficulties in simulating multi processor systems

In the preceding section, it was shown that simulation experiments restricted themselves to a single aspect of computer systems at a time and determined its effect on system performance. Interplay between various aspects and their combined effect on system performance could not be studied. The interest in simulating systems in great detail is to find out such results.

Conventionally, the resource allocation or resource management problem in computer systems has been conveniently analysed as extensions of the job shop techniques. In a job shop, jobs arrive and are split into smaller tasks which compete for resources (machines in the job shop). The problem is to schedule operations so as to resolve conflicts for resources in the best way.

Computer systems differ from job shops in the following way.

- (i) In a job shop, the resource allocation is done by an external agent; e.g. the shop manager. In computer systems the resource management problem is tackled by the operating system which itself competes for the system resources like processor and main memory etc. The time required for performing these functions is significant for system performance purposes. This time constitutes the 'overheads' in operating the system as no useful user load is served during this time.
- (ii) Normally, jobs or tasks in a job shop are non interruptible once they are being serviced by a system component. In other words, they seize complete control of a system resource once they have captured it. The system resource is released only on completion of the task. This is not so in computer systems simply because of the flexible interrupt system that is provided with the present day systems.
- (iii) The attributes of a job shop are few and are easily describable. In the case of computer systems however, description of the workload in terms of typical programs

and detailed description of these programs in terms of the resources required to execute it is by no means a simple thing.

It is evident that computer systems cannot be studied in the same way a job shop is analysed. The necessity to examine the detail and also the higher level activities of computer systems have given rise to some problems in simulating such systems. The coming of multi processor systems have added a new dimension to the complexity of the behaviour of the system. This is due to the increase in parallel activity in the system and possible relationship between activity of various processors.

Hutchinson [67] points out three main difficulties in simulating computer systems. These problems are more acute in the case of multi processor computer systems.

- (i) The first difficulty is the degradation of ability to differentiate between time of occurrence of successive events. This point needs some clarification. The detail of the experiment dictates the smallest unit of time by which the simulated time is updated. This unit of time may be looked upon as the time required by the smallest activity in the system. At the end of this activity an

event is scheduled to occur. This is achieved by placing in the event schedule a code for this particular event. An attribute of this event is the time it occurs. This is obtained by adding to the present value of simulated time the time required for completion of this activity. The danger in doing this is that if the simulation has been in progress for quite some time, the addition of a small amount of time to schedule the occurrence of another event may make no difference to the sum. The smallest unit of time which a simulation program can use is therefore dependent on the word size of the computer system being used and the format for storing floating point data. This problem is therefore a technological problem and decreases with increase in word size of the computer system used.

- (ii) The second difficulty faced in simulating computer systems is the occurrence of simultaneous events. This problem can become more severe as a direct consequence of the point made earlier. Such occurrence of events can cause logical problems. This is due to the

fact that most simulation languages do not consider such situations. For example, both GASP and SIMSCRIPT store the event schedule in a linear list. The list is ordered according to the time of occurrence of events. If two events happen to occur at the same time, one of them takes precedence over the other depending on the logic used by the program which updates the event schedule. In simulating the system, events are taken up one at a time and hence any time relation between two simultaneous events is lost. Due to the first difficulty pointed out, it may not be even possible to decide if two events in the event schedule having the same time of occurrence are really simultaneous or a consequence of degradation of the master clock resolution.

- (iii) Another major difficulty with simulation of computer systems in great detail is the representation of workload. Some of the requirements in characterising the user are proper depiction of parallelism in user programs and machine independence in specifying the various activities in the workload.



Due to the unsatisfactory development of simulation techniques and languages to deal with problems posed, Hutchinson observes that the most successful simulation study of multi processor computer systems was programmed in FORTRAN [37]! This program deals with the simulation of IBM System/360 Model 67. SIMSCRIPT and GPSS 360 have also been used to study some multi processor computer systems [66, 68], but have proved to be expensive in computer time. For example, a one minute simulation of IBM 9040, programmed in GPSS 360, took around 30 minutes on an IBM System/360 Model 50. Inferences, about system performance over long periods of operation, drawn from the information obtained from such simulation studies are quite questionable.

It is interesting to note that in the 1967 IFIP Working Conference on Simulation Programming Languages, in the discussion following a paper on on-line incremental simulation [69], there was a general agreement that the use of simulation in studying large and complex behavioural systems is more to understand the system through model building than to obtain some statistics about its behaviour.

When attempting to undertake a simulation study of a computer system, the following two points should be clearly

thought out.

- (i) The first thing is to put down the objectives of the simulation study. For example, a good detailed simulation model should serve the following purposes. It should aid in configuring the system, act as a test vehicle for various resource management algorithms (memory allocation, task scheduling and I/O scheduling algorithms) and help in studying the effect of various job mixes in user workload on system performance.
- (ii) Second, the level of detail of the actual system a simulation model attempts to inspect should be clear from the start. This is important because the characterisation of the user workload is greatly dependent on it. Neilson [37] indicates that the detail of time and space for an IBM System/360 Model 67 simulation study was fixed up at 100 micro seconds and a page of program respectively. Anything an order of magnitude finer in detail than these limits were just ignored in the simulation study.

Section 3.6: Acquisition of data for assessing system performance

So far the emphasis has been on the system and the boundaries that demark it when considered at different levels of detail. No simulation study can be undertaken however, unless the environment within which the system operates is determined [70]. Therefore, acquiring data about this environment forms the first phase of work done when trying to analyse or simulate a system. Design of experiments on computer systems to obtain such data is of paramount importance as the data collected should represent the actual conditions in the system and should not be adulterated by the measurement technique used.

Two classes of techniques for acquisition of data for evaluation purposes are currently being used - software measurement techniques and hardware measurement techniques. Almost all data can be obtained by using software measurement techniques. However, the same cannot be said for the hardware measurement methods. Ease of implementation, ease of use and the ability to obtain data without interfering with work in progress are the principal reasons for using hardware measurement techniques. All measurement techniques can be classified as one of two types. The summary type, which yield a summary of the behaviour of the aspect being monitored at the end of the experiment. And the dynamic trace type,

which record all information about the feature being monitored from the start of the experiment.

Description of hardware monitors and their use in monitoring system performance has been reported in literature [71, 72]. All such references to such work have come from IBM. Hardware monitors feature some registers, counters and logic modules and are programmable by patchboards. Several probes are available which can be introduced at strategic points in the CPU, data channels hardware etc. Measurement of total I/O time for each device, channel busy time, CPU busy time, CPU and channel busy time etc., is quite easy with such equipment. Meaningful dynamic instruction traces can also be obtained as this does not interfere too heavily with normal computation by the processor. The main drawback of hardware monitors is that information that is dependent on the context of work in progress cannot be detected and hence monitored. Examples of such information are job identification, data set identification, origin of requests for system resources etc.

Software measurement techniques can be readily used to obtain all such information. The most natural way of monitoring a system in operation, using such techniques, is by use of 'hierarchical control programs' [73]. As is well known, computer systems operate in two states or modes; the

'supervisor state' or 'privileged mode' and the 'problem program state' or 'normal mode'. It is seen that when priorities are assigned to users, there is a hierarchy established even in the 'normal mode'. This concept of hierarchial structuring of programs is carried a step further for providing a software system monitoring technique. The operating system and user programs are executed in the 'problem program state' with the operating system enjoying the highest priority. The 'supervisor state' executes the software monitor which controls execution of all other programs. Any return of control to the software system monitor causes it to examine the reason for the return of control and this information is logged in if it is of interest.

In computer systems where this is not easily possible, the same end result may be achieved by introducing at strategic points in the operating system, instructions which transfer control to the central routine which handles all monitoring requests. An example of such monitoring operations is Scherr's work in measuring the CTSS users' characteristics [48].

As the level of detail of the information being monitored becomes finer, software measurement techniques prove to be costly in computer time and can also lead to distortion of time sensitive data. Another severe

limitation imposed on the software measurement techniques is that the smallest unit of time measurable by program is the unit interval of the interval timer. This can drastically affect the measurement of detail and of time dependent information. Finally, these techniques being the only ones available to most people interested in evaluative work, a very thorough understanding of the program code (mostly in assembly language) of the operating system is absolutely necessary. Therefore, successful evaluative studies published in literature have been conducted by organisations which have developed their own software support to a large extent. With system documentation being what it is presently, a novice to this field is faced with the unpleasant task of digging through tricky and unexplained masses of assembly language programs which form the operating system.

Section 3.7: Conclusions

The inadequacy of mathematical models to depict satisfactorily the behaviour of computer systems is one of the factors that have made simulation techniques indispensable for assessing performance of systems. Indeed they have been the only tools available to study some of the problems encountered. Much work needs to be done in developing techniques and languages to deal with parallelism in the systems to be simulated.

Choice of proper machine independent measures to portray user workload is a very important problem to be tackled if any standardisation of workload specification is to be

achieved. This objective is indeed a desirable one to obtain and it also necessitates the procurement of information about user load from a wide variety of application areas. So far, most evaluative work has been conducted mostly by academic institutions and most information available about workload concerns the environment of computer systems in such organisations. A lot of field work, in terms of collection of data, especially from computer installations serving a commercial data processing load, is necessary and seems to have evaded the reach of academic institutions and computer system manufacturers so far.

Standardisation of this type can readily lead to the synthesis of the workload of a computer system. Present difficulties in understanding the behaviour of computer systems are bound to recede in the coming years. The major problem in evaluation of performance will be to characterise the environment for input to a computer model.

CHAPTER IV

STUDY OF EVALUATIVE PROBLEMS FOR THE IBM 7044-1401 COMPUTER SYSTEM AT INDIAN INSTITUTE OF TECHNOLOGY KANPUR

Batch processing computer systems are very much in use today. A decade ago, in such systems, unit record equipment like card readers and line printers were attached to the CPU through non overlapped data channels. These factors led to an ineffectual utilization of the computational resource whenever the media for system input and output were unit record equipment. A more effective operation of these systems was achieved by providing off line card-to-tape and tape-to-printer functions using a small off-line satellite computer. Variations and elaborations of such techniques were developed.

The medium-scale present generations batch-processing systems do away with the off-line satellite-computer concept by using a technique called 'spooling'. These machines have all peripheral equipment connected through overlapped data channels to the CPU and are run in the multi-programmed mode. High priority card-to-tape and tape-to-printer system routines are spooled with the rest of the user programs. Such an operation of the system is seen to exhibit high utilization of resources and achieve the same end result economically.

In this chapter, the interest is in evaluative problems of computer systems having off-line satellites to perform peripheral tasks. In particular, the IBM 7044-1401 system at Indian Institute of Technology, Kanpur is singled out for such study. The motivation for this is obvious: IIT/Kanpur has purchased this system from IBM and so, it is very desirable to make the best usage of this resource. Moreover, the system has been operating in saturated conditions, i.e. it services the maximum load possible with its present capabilities. Therefore, it is worthwhile to consider in depth the problem of economically changing the system to increase its capability to service the present workload.

The various investigations reported in this chapter can be applied to other systems like the IBM 7044-1401 too.

Section 4.1: Computational resources at IIT-Kanpur

The major computational resource at IIT/Kanpur is an IBM 7044-1401 system. Currently available hardware facilities and the projected system are both given in Figure 4.1. The problem solving facilities and programming environment is provided by the IBSYS version 9, modification level 9, operating system. In addition to the multifarious utility functions performed by the system [74, 75, 76, 77, 78] several new facilities have been introduced into IBSYS at IIT-Kanpur [79].

CPU 7107

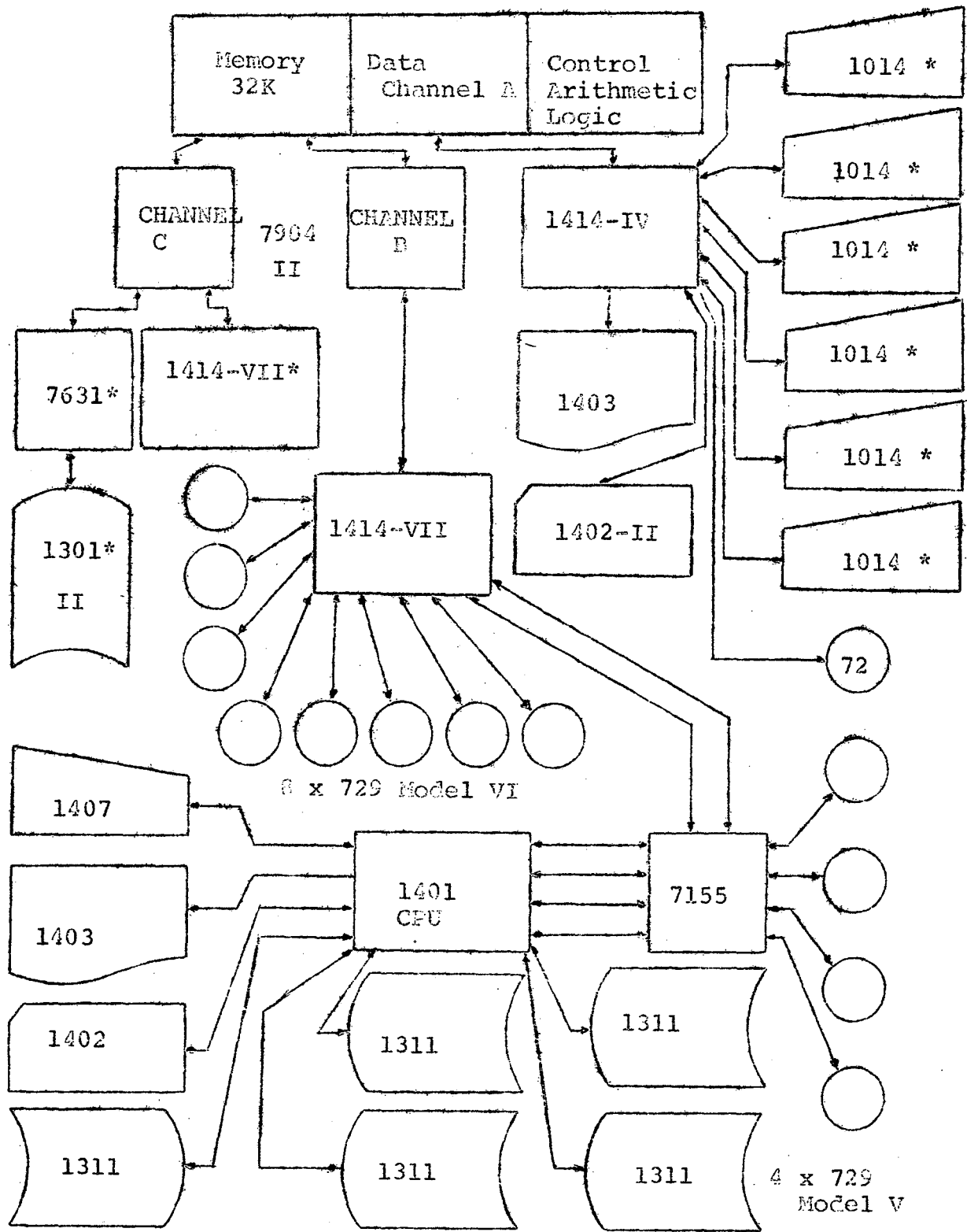


Figure 4.1: The IBM 7044-1401 System configuration at IIT/Kanpur (Legend on next page)

* Equipment on order

Legend for Figure 4.1

1. 7107 CPU, 32K Memory, extended instruction set
2. 7904 II, Two overlapped simplex data channels
3. 2 x 1414 VII, I/O Synchronizers
4. 1414 IV, I/O Synchronizer.
5. 6 x 1014, Remote enquiry units
6. 72, Console typewriter
7. 1402 II, Card reader/punch, 800/250 cpm
8. 2 x 1403, Line printer, 132 ch., 600 lpm
9. 8 x 729 VI, Magnetic tape 90 kc/s.
10. 7631, File control unit
11. 1301 II, Disk, 2 arms
12. 7155, Tape switching unit
12. 1401 CPU, 16K, extended features
14. 5 x 1311, Disks
15. 729 V, Magnetic tapes, 60 KC
16. 1402, Card reader/punch, 800/250 cpm
17. 1407, Console enquiry unit

Section 4.2: Operation schedules for the IBM 7044-1401 system at Indian Institute of Technology, Kanpur

The IBM 7044-1401 system was shifted in June 1969 to the new Computer Centre building, into a single room. The Computer Centre was then servicing nearly 500 jobs a day. The problem of proper sequencing of operations on these machines so as to utilize them best and the problem of system design of various other support services for the efficient operation of the total system demanded immediate attention. Several operation schedules were tried out. An optimal schedule in terms of throughput, turnaround and adaptability of the schedule to actual operating conditions has been developed and implemented. The design of support services that were implemented and an informal argument to prove the optimality of the schedule developed are presented here.

The organisation of a batch processing system is similar to that of the job-shop. The various services that are necessary here are:

- (i) Job identification.
- (ii) Receipt of input jobs
- (iii) Distribution of job output alongwith input card deck
- (iv) System operation schedule.

The design of interfaces between these services is important. The effort was to keep communications between them down to a minimum.

Incoming jobs are assigned priority by its type and resources it needs. The decision table shown below gives the priority scheme. Identification of jobs is according to

	Else		
Special job ?	Y	-	N
WATFOR job ?	-	Y	-
Time < 3 min?	-	Y	N
Time < 8 min?	-	-	Y
Pages < 30	-	-	Y
Cards output <200	-	-	Y
Special job	x		
Express job		x	
Short job			x
Long job			x

their priority. Serial numbers are assigned to jobs in a priority group as they arrive. A card bearing the serial number is introduced into the input card deck. This card bears the following information.

- (i) Proper control characters in columns
1 and 2
- (ii) Priority group
- (iii) Serial number
- (iv) Date

The IBM 519 programmable reproducing punch was programmed to gang-punch the control characters, priority group and date, step-up and punch the serial number and print the serial number and date, all in one pass (the first card bearing the initial serial number and the rest of the information is pre-punched). Duplicate sets of serial number cards are

produced and collated. Identification cards for Long Jobs and Special Jobs are produced for a period of one month at a time as the number is quite small in either case. A fixed number of Short Jobs are serviced everyday and hence corresponding identification cards are produced for a week. However, there is no limit on the number of Express Jobs run in a day. As there is a lot of fluctuation in this number, it is advisable to produce the identification cards for these jobs in small lots for this is quite easy with the set up indicated here.

Input service counter operations are depicted in Figure 4.2 and need no further elaboration. To minimise card and deck handling input jobs are put into card cabinet trays, processed by IBM 1401 and introduced into card cabinets (this is yet to be implemented). Trays are tagged to indicate jobs in them. The output service counter operations are depicted in Figure 4.3.

It is easily observed that the IBM 1401 and 7044 systems have to act in tandem. In other words, the sequence of operations to provide service to a batch of jobs is card-to-tape on IBM 1401, processing on IBM 7044 and tape-to-printer on IBM 1401. So, in a given period of time IBM 1401 is used to prepare a new 'input tape' and print an 'output tape' (a result of the processing by the IBM 7044) while IBM 7044 is used to process, sequentially, all available input tapes.

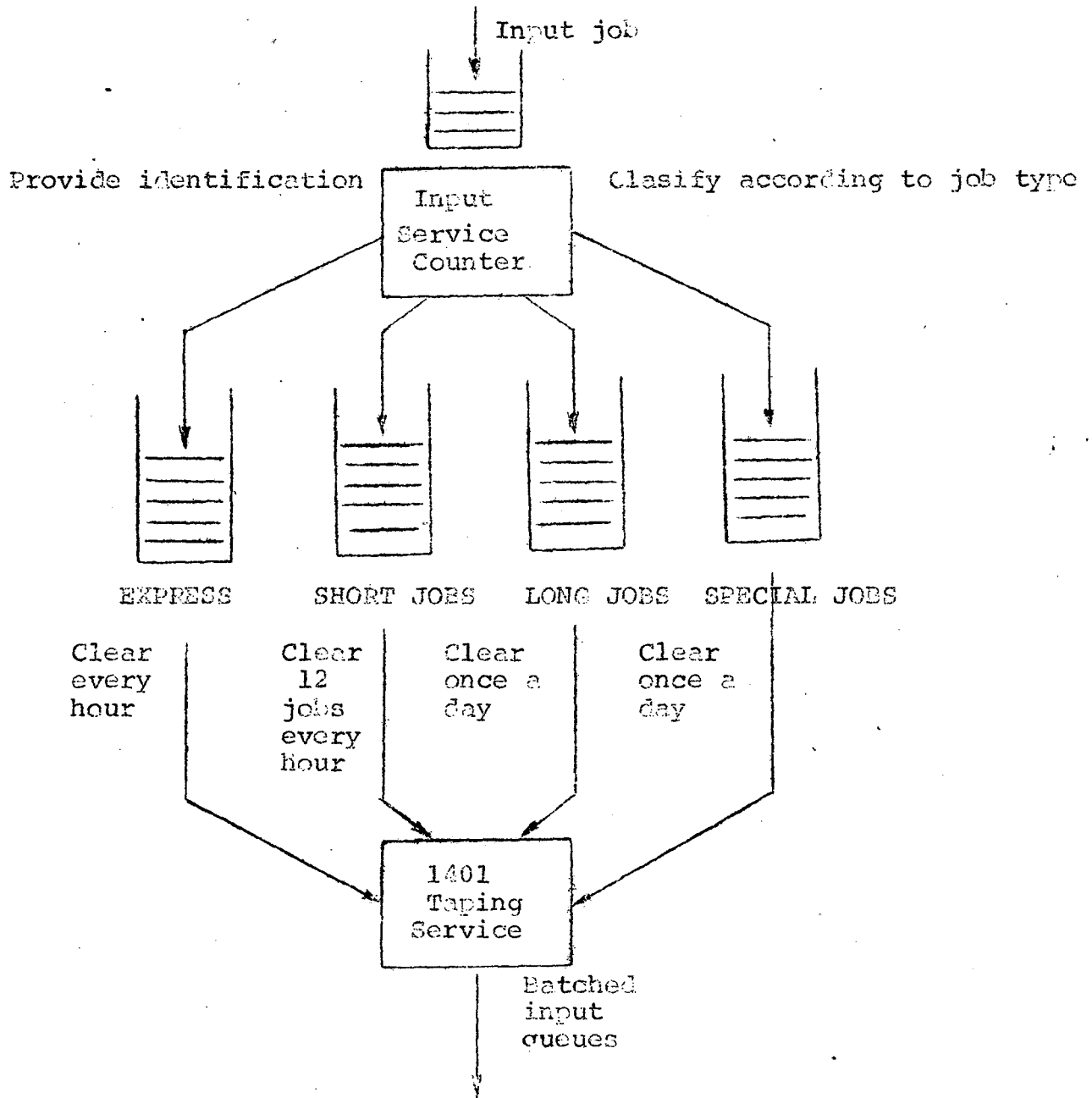


Figure 4.2: Flow of operations for system input

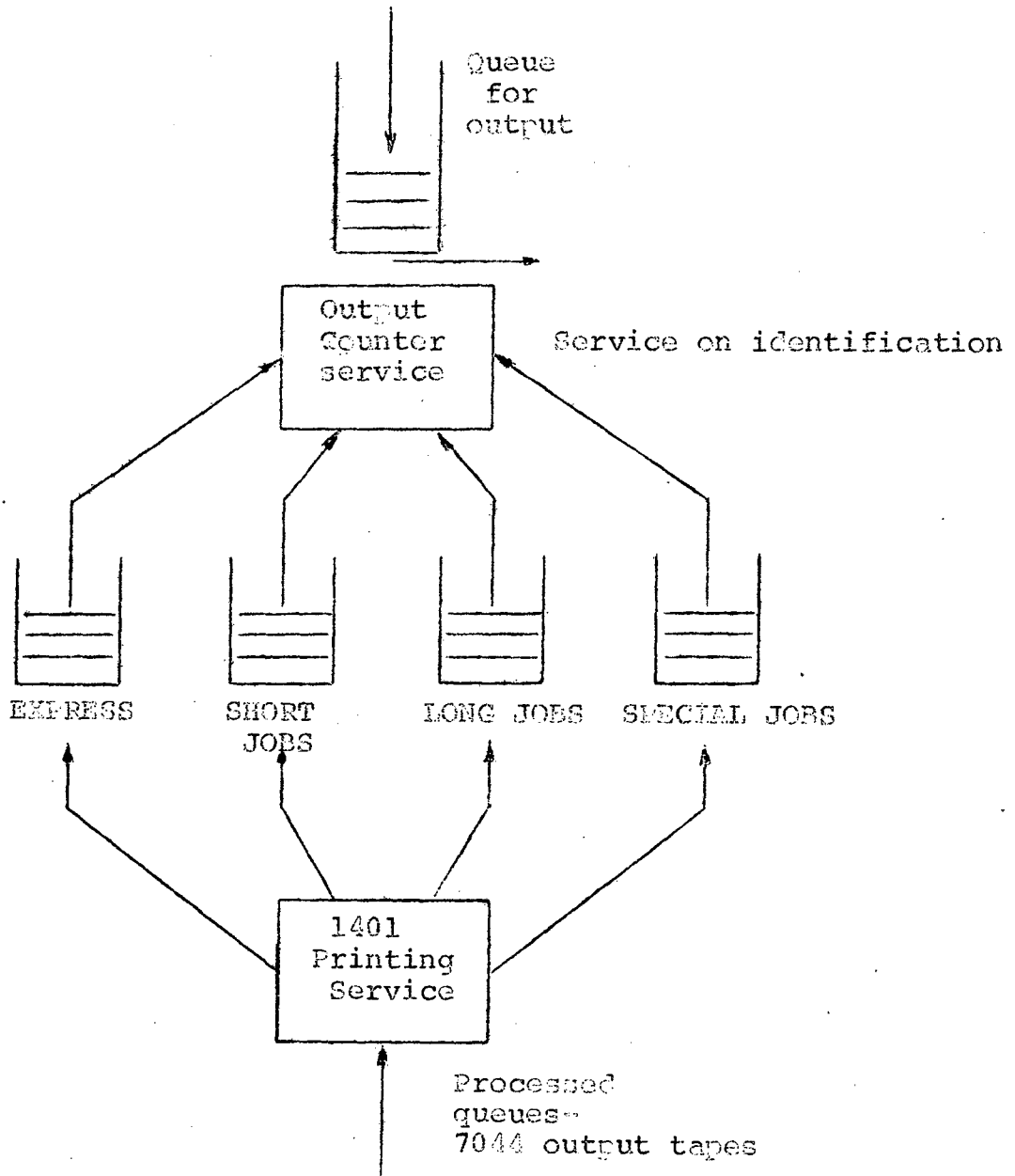


Figure 4.3: Flow of operations for system output

Klienrock has shown with his analytical model for Sequential Processing Machines[80] that optimal (least machine idle time) usage of such systems occurs when both machines take the same time to service a task. This result can be applied to the IBM 7044-1401 system very profitably. The idea is to choose a batch size such that the time taken by IBM 1401 to read input jobs in the batch and tape them and to print the output tape of the batch, matches the time taken by IBM 7044 to process the batch. Maximum throughput is the outcome of such design.

It is indicated in Section 2.2 that short jobs are normally given better service (e.g. SJF, RR, FB service disciplines for processor allocation). The RR and FB service disciplines are suited for the time sharing environment and in multiprogrammed systems. For simple one-job-at-a-time batch processing systems, the SJF service discipline is employed. Therefore at IIT/K, short jobs have high priority. Highest priority is given to Express Jobs, next level of priority to Short Jobs, the third level to special jobs and the lowest to Long Jobs. Hence maximum turnaround is given to Express Jobs and next to Short Jobs. Special Jobs are run once a day. Long Jobs join a queue and are serviced for 3 hours at night. If any Long Jobs are left over, they are queued for the next night.

The problem now is form a batch of jobs from the various priority bins (see Figure 2.2) to satisfy the above constraints of throughput and turn around. Statistical about user-job-running times on IBM 7044 and taping and printing times on IBM 1401 were collected. The observations at that time were the following:

- (i) An average of 13 Short Jobs could be processed in 1 hour by the IBM 7044.
- (ii) A batch of Express Jobs collected over 1 hour took on an average the time taken to process 1 Short Job.
- (iii) The time taken for taping one batch of jobs consisting of 12 Short Jobs and 1 Batch of Express Jobs and the time for printing the output produced by such a batch was 1 hour.

The operation schedule developed turned out to be very simple. 12 Short Jobs and all available Express Jobs are cleared at the end of every hour for taping on IBM 1401. At the end of taping of one batch of jobs, the output tape of the previous queue is printed. These two operations on the IBM 1401 are completely overlapped and matched by the processing of an input batch on the IBM 7044. Special and Long Jobs are cleared at night and serviced during the allotted period. This schedule is partly shown in Figure 4.4.

Time	1401		7044
	Input Tape	Output Print	PROCESS
9.00	Q ₁		
9.30			Q ₁
10.00	Q ₂		Q ₁
10.30		Q ₁	Q ₂
11.00	Q ₃		Q ₂
11.30		Q ₂	Q ₃
.	.	.	.
.	.	.	.
.	.	.	.
00.00			Special
01.00			Long Jobs
04.00			Resume other queues

Figure 4.4: Operation Schedule

Other design factors in the schedule were (i) separate time slots for tape-cleaning at approximately 10 hour intervals and developmental work for the system. This schedule was implemented and found to work very satisfactorily. In fact, an unprecedented service of about 400 Express Jobs, 185 Short Jobs and the usual load of Special and Long Jobs was achieved (at the expense of a supervisor to carefully monitor the operations).

The only snag is that problems of human resource management seem to be much more difficult to handle!

Given these operating conditions, this schedule has some other very pleasing properties. For instance:

- (i) There is least accumulation of jobs at the input counter.
- (ii) There is no accumulation of jobs and output tapes at the IBM 1401 and input tapes at the IBM 7044 than the minimum necessary.
- (iii) The modularity of queues or batches makes it very easy to make decisions in actual operating conditions. For example, when unforeseen machine breakdowns occur, it is very easy to decide how many serial numbers to cancel etc., so as to be able to keep to the schedule.

Other schedules that have been implemented at this Computer Centre do not mix priority levels when batching incoming jobs. Therefore batches of high priority jobs have to be rushed through the sequence of operations to be performed for their service. Therefore, turnaround to the high priority jobs is obtained at the cost of other lower priority jobs. This leads to accumulation of input and output tapes of lower priority jobs (the commitment of turn around to high

priority jobs is seen to cause this) which are waiting for service inside the computer room. To beat the excessive constraints on operations of IBM 1401 caused by this situation, the use of unit record equipment of IBM 7044 is necessitated. This drastically effects the throughput.

An overlap of operations in the I/O operations of IBM 1401 is now possible. Therefore, the schedule has to be re-designed to suit its current capabilities. The principles of design and the constraints remain the same, but a different batch size should be chosen to optimize the operations in terms of increased turnaround.

Section 4.3: Models for IBM 7044 system

In this section are presented some models for the IBM 7044 system. The motivation for producing these models is to help find interesting and economic changes to the system which will result in improved performance in terms of throughput.

The first model is an elaboration of a similar work done to model the University of Michigan (Ann Arbor) Executive System [81]. The behaviour of an operating system can be modelled as Semi-Markov process. Essentially, the model looks like a probabilistic finite state transition graph (i.e. a probability is associated with each transition from a state). Only, there is associated with each state a transition time which depends on the next state. In fact, probability distributions ^{to} characterise each of these transition times are

necessary. For the present case it is sufficient to associate an expected value of transition time with each state.

A probabilistic transition graph for the IBSYS Operating System is given in Figure 4.5. The main difference between this model and Foley's [81] is that states have been included to record the time for fetching various system library programs. This is meaningful under present operating conditions like the residence of system library programs on a tape unit. The chief result that can be obtained from this model is the overhead incurred in fetching various library programs from the system library tape. Naturally, much depends on the position of the system library tape. The position determines the amount of time needed to access the Library program to^{be}/loaded next.

All information about various transition probabilities and times of residence in all the states is not yet known.^{can be got} However, some interesting results/ from the model. For instance, about 180 FORTRAN IV jobs, using the IBFTC (IBM supplied) compiler, are run every day. Each successive \$IBFTC card (command language statement of IBSYS) causes the system library tape to be backspaced so as to fetch the SCAN phase of the compiler. This operation is observed to take 9 seconds approximately. If each job has on an average 3 subroutines, then $2 \times 180 \times 9$ seconds of the total operation time is wasted in recalling the IBFTC compiler. It works out to be

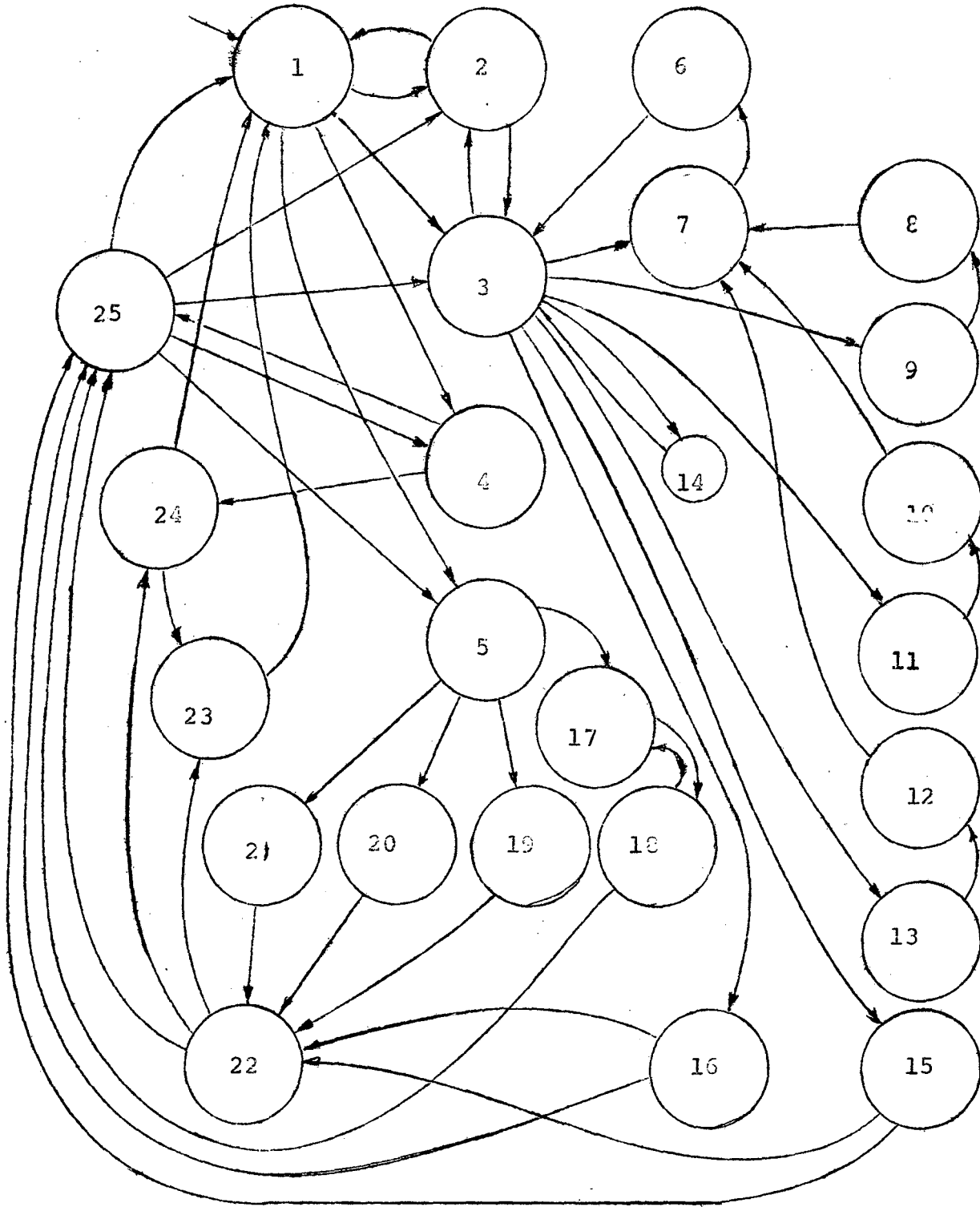


Figure 4.5: A gross model for IBSYS operating system
(Legend on next page)

Legend for Figure 4.3

<u>State</u>	<u>Description</u>
1	Get new JOB, \$ JOB/\$IITEND
2	System Edit, \$IBEDT
3	Processor Monitor, \$IBJOB
4	Sort System, \$IBSRT
5	Installation Programs, \$EXECUTE
6	Assemble MAP program
7	Get MAP from Library, \$IBMAP/Compiler Interface
8	Compile FORTRAN IV program
9	Get FORTRAN from library, \$IBFTC
10	Compile COBOL Program
11	Get COBOL from Library, \$IBCEC
12	Compile ALGOL Program
13	Get ALGOL from library \$ALGOL
14	Receive Binary Program, \$IBLBR
15	Load user (relocatable) object program, \$ENDCHAIN/\$ENTRY
16	Load user (absolute) object program, \$RELOAD/\$RESTART
17	Compile WATFOR program, \$WATFOR
18	Execute WATFOR program, \$ENTRY
19	Get UPDATE from library
20	Get SNOBOL from library
21	Get GPSS III from library
22	Execute Program
23	Check point
24	Dump
25	System Reset, \$IBSYS

nearly 5% of the total operation time! Similarly, the WATFOR compiler, GPSS III, UPDATE, Check point and Dump routines take 20 seconds to fetch. The frequency of occurrence of such operations and total time wasted in such tasks are all obtainable from the proposed model. Furthermore, quantitative improvements available, if the system library resides on a disk instead of a tape, can be easily predicted. Re-organisation of programs in the system library and the relative merits of such efforts can again be quantitatively obtained.

Another problem worth investigating is the effectiveness of the configuration in servicing the I/O load. SCERT (explained in section 3.3) is claimed to be technique which solves this problem. In this technique, file description and volume of I/O expected on each file are the two attributes which characterise the I/O load. This is indeed not sufficient to find the effectiveness of the various parallel paths provided in dealing with the I/O load. An important point missed is that the density and volume of I/O traffic are the necessary attributes besides the file description of each I/O file. This was pointed out and characterised in Section 2.3.

Interesting questions that can be asked in such a study are:

- (i) How does the introduction of one more data channel affect the performance of

the system?

- (ii) What I/O files to shift from one channel to another?

An elaboration of this point is made by discussing the IBM 7044 system at IIT/Kanpur in this context. At present there are 10 tape units on a simplex data channel (only one way operation is possible at a time, either read or write). Another similar data channel is on order and so is an additional synchronizer to connect the available tapes to it. The problem now is to shift some tapes on the existing channel to the new channel so that performance is improved (examination of processor idle time for each of the cases should suffice for this purpose).

In the case of a simple batch processing system like IBM 7044, the behaviour of programs may be simply characterised as a sequence of logical I/O requests that occur in program execution time. The buffering scheme determines the way in which actual or physical I/O requests are generated and whether program execution can continue or not (this depends on whether the I/O request can be scheduled or not). Such a problem can easily be studied by use of simulation techniques.

The model for such a system may be constructed as a multi-level server system. The first-level-server is the buffering system. This level has a maximum queue length of just one request. This occurs whenever the incoming request

necessitates a physical I/O to be initiated and at the same time, the previous physical I/O request on the same file is not yet serviced. The second-level-servers are the various parallel data paths in the system. Queues for this server can have at most one entry per file that can be accessed through it. These entries are put in a list which has a pre-arranged sequence (entry for device 1, entry for device 2, etc...). The order could for example depend on the speed of the device. Service discipline for this server is simple. The top element of this list is serviced first.

The usefulness of this model is that it can indicate the clashes of I/O requests on each device. That is, the service of one is excessively delayed due to the other. Such devices should naturally be separated and connected to different parallel path. If the clash is on the same device, then the buffering scheme provided does not fully serve its purpose. An interesting study of behaviour of system programs which do a lot of I/O in their execution is possible (e.g. IBFTC compiler). The results obtained from this model would be very useful in accounting for processor idle time (time for which the first level server is blocked) and whether it can be easily reduced.

Section 4.4: Specification of a software monitor and a technique of implementation

The lack of any quantitative results by using these models (to justify all their virtues which were extolled in

the preceding section!) would seem to be a glaring omission in the work reported here. However, it is very difficult to obtain data to characterise the workload and obtain statistics about system behaviour. (This point was discussed in Section 3.6). Therefore, only a specification of the software monitor to obtain this information and a technique to implement it in the IBSYS operating system are sketched here.

The software monitor proposed here is a hybrid between the hierarchical control programs [73] and monitoring routines distributed throughout operating system. Control is not transferred to the monitor whenever the system switches to the privileged mode of operation. Routines executed usually in the privileged mode gain control and make an immediate call to a control system monitor. This information is logged in alongwith an identification tag so as to indicate the nature of call. Other system routines also call the system monitor (FORTRAN compiler, for example) to log in data that characterises the IBSYS operating system. A detailed log of system usage is therefore available. External control of the system monitor logic should be provided so as to be easily mask the monitoring operations for certain aspects of system behaviour. For example, it might be of great interest to just get the I/O characteristics of the IBFTC compiler and ignore the rest.

The system monitor has two major parts. The first part logs in data about gross system operation or system behaviour (e.g. IBFTC compile time, MAP assembly time, get IBFTC time, etc). The second part logs in the information about I/O either by a given subsystem of IBSYS or of the user program.

Several things have to be kept in mind when implementing such a software monitor. The first is that this work entails dabbling with the 'nucleus' or 'resident' (IBNUC in IBSYS) of the operating system. Changes in this part of the system should be kept to a minimum. Any change necessitates a fore knowledge of all possible consequences of the system. Also, it must be noted that changes cannot be made frequently in this part of the system because it can cost enormous amount of computer time to effect a change. To be more specific, re-assembly of IBSYS due to a change in IBNUC would necessitate the following steps:

- (i) Get binary deck of IBNUC in absolute mode.
- (ii) Update system edit file with new IBNUC
- (iii) Assemble the whole system library using the new system edit file.

These jobs need at least one and half hours of IBM 7044 time. Therefore, the debugging of software monitors and consequences of changes made to the system are worth a lot of attention.

Another important aspect of implementation is that user programs should not be in any way affected by the software monitor, either in terms of time or space (both the attributes of space, size and address). A memory map of the IBM 7044 system is given in Figure 4.6. From this it is seen that there is a large blank area of core memory at upper end of the protected zone.

The system monitor program and buffers to log in information can be conveniently located in this place. This does not interfere with user core area in any way. In fact, if no I/O has to be monitored, then there is no addition or modification to any nucleus routine at all. A change in the entry for the installation accounting routine (SSIDR) in the list of system transfer points (see Figure 4.6) is all that is required. Normally, an entry to the installation accounting routine is just screened for particular types of call. For these calls the system regains control and for all others, control is immediately returned to the calling program. The only change to this part of the nucleus is that all calls to S.SIDR cause a call to the system monitor. Other changes to be made to IBSYS are also quite simple. The entry to all subsystems of IBSYS, depicted in the Semi-Markov chain model, should cause a call to S.SIDR to log in accounting information.

LOCATION	DESCRIPTION
START	IBNUC
00000	Machine functions/ engineering words
00100	Load area
00135	System Transfer points + data words
	* Symbolic Units Table
	* Unit Control Blocks
	* System Control Blocks
	Nucleus routines (system loader etc)
	IOEX Basic functions
	* Channel tables
	Interrupt schedu- ler
	IOOP Header routines
	Device routines
	Common routines
	IOLS Reels processing Labels check
end at 07777	Blank protected area
10000	IOBS not storage protected
	s.sloc
	File description
	User prog
	Unused core
	I/O buffers + pool control table
end at 77777	User common area

Figure 4.6: Memory Map of IBM 7044

* length depends on actual configuration.

A simple buffer of 100-150 words, to log in information for all calls to S.SIDR, is sufficient. This buffer size depends on the maximum expected number of job steps in a job. The end of a job causes this information to be flushed out onto an 'accounting tape'. The overheads involved in such a scheme for monitoring gross system behaviour is minimal. A point of note is that no change to any user program need be made when implementing this part of the system monitor.

Monitoring I/O is a bit more difficult. In this case, the main difficulty is to keep program execution time separate from the time the system takes in servicing I/O channel interrupts. This is because the characterisation of I/O traffic should be configuration independent. That is to say, all I/O requests should be represented in program execution time which in turn should not reflect any effects a particular configuration may induce. Therefore, a 'software clock' may be associated with the program being executed. Another 'software clock' contains the amount of time spent in the interrupt servicing or delay routines. The latter of course necessitates changes in the lower levels of the input-output managerial routines (IOCS in IBSYS) so as to keep track of above mentioned times. Every I/O request made by the program causes its 'software clock' to be properly updated and information about inter-arrival time of I/O requests,

requested I/O file and magnitude of data transfer needed by the I/O request are logged into a buffer. This buffer is flushed onto an accounting tape when ever it is full so that a concise trace of I/O requests is obtained. The implementation of all these procedures necessitates careful changes to IOCS and a re-conversion of all (system and user) programs in 'absolute mode' (such programs need no relocation and can be loaded into core memory without any changes). The buffer size dictates the overheads that are incurred with this scheme for monitoring. However the data made available by this monitoring scheme is invaluable as such data has not been reported anywhere in literature.

Section 4.5: Conclusions

Particular problems in performance evaluation can be solved very fruitfully with the aid of already available techniques and sometimes with available models for system behaviour. In this particular instance, tangible results were directly achieved by the design of an optimal operation schedule. The model for gross system behaviour proposed here, should when quantitatively solved, give a break down of the manner in which the overall system resources are utilized. Also the direction for reorganisation of the system to improve system throughput can be perceived by analysing such results. Finally, the configuration evaluation problem for a particular simple type of computer system can be solved by using

the I/O model proposed here.

CHAPTER V
CONCLUSIONS AND SUGGESTIONS FOR
FURTHER WORK

Section 5.1: Summary and conclusions

The material presented in this thesis is an attempt at a comprehensive view of the problem of computer system performance evaluation. In retrospect, Chapter I introduces the concept of performance evaluation and brings out the importance of this problem in the study of the economics of computer design, manufacture, selection and usage. Chapter II encompasses an analysis of various important factors in modelling computer systems with a view to predict their performance. It also contains critical discussions of several techniques of performance evaluation developed and an analysis of how and where they can be used. Statistical characterisation of system workload is also presented for this provides the environment for a computer model. Chapter III gives a critical discussion of simulation techniques and their use and limitations in modelling of computer systems. Chapter IV is more down-to-earth; a study of evaluative problems for the IBM 7044-1401 computer system currently operational here at IIT-Kanpur is presented.

The interesting results of this chapter are an optimal schedule of operations for this computer system and a proposal of a model for configuration evaluation of such systems. The latter is an important problem and has not been studied or reported elsewhere in literature.

Objectively considered, this thesis contributes little to the solution of the general problem of modelling large-scale general-purpose computer systems. Hopefully, the understanding of the immensity of problem and difficulties with currently available techniques are cogently presented. The difficulty with such systems seems to be one of improper design. Briefly, the problem is to develop proper resource management procedures which take into account the relationships between page referencing, I/O requests and program execution. Except for the 'working set' model for program behaviour, which relates page referencing and program execution, there has not been effort in this direction. Therefore, an analysis or evaluation of implementations of such design prove to be very cumbersome and difficult.

It is possible to be more positive in the case of the problem of selection of computers. Techniques discussed in Chapters II, III and IV can be used for a detailed analysis of the capabilities of the computer system in question. This should indeed give a comparative estimate of relative capabilities of each computer system for different types of tasks.

Section 5.2: Suggestions for further work

Performance evaluation of computer systems is a rich source of challenging problems. Several references to problems yet to be solved are distributed in the earlier chapters of this thesis. The major problems are presented below:

- (i) Inclusion of I/O in the strategies for resource allocation (other than I/O management, of course).
- (ii) Collection of statistics to characterise user job profiles.
- (iii) Development of simulation techniques to simulate, efficiently, parallel events that occur in a system.

REFERENCES

1. Richards, R.K., 'Electronic Digital Systems', John Wiley & Sons, Inc., 1966.
2. Hassitt, A., 'Computer Programming and Computer Systems', Academic Press, 1969
3. Rao, P.V.S., 'Introduction to Computer System Design', unpublished lecture notes, Computer Group, Tata Institute of Fundamental Research, Bombay, 1966.
4. Mealy, G.H., 'Operating Systems (Excerpts)', from 'Programming Systems and Languages', ed. Saul Rosen, pp 516-534, Computer Science Series, Mc-Graw Hill Book Company, 1967.
5. Kilburn, T., Payne R.B., and Howarth, D.J., 'The Atlas Supervisor', *ibid*, pp 661-632.
6. - 'IBM Operating System/360 Concepts and Facilities (Excerpts)', *ibid*, pp 598-646.
7. Daley, R.C., Neumann, P.G., 'A General Purpose File System for Secondary Storage', Proceedings of FJCC, AFIPS, Vol 27, pp 213-230, 1965.
8. Dennis, J.B., 'Segmentation and the design of multiprogrammed systems', JACM, vol 12, pp 589-602, October, 1965.
9. Wegner, P., 'Machine Organization for Multiprogramming', Proceedings of 22nd National Conference, ACM, pp 135-150, 1967.
10. Saltzer, J.H., 'CTSS Technical Notes', MAC-TR-16, March, 1965.
11. Corbato, F.J. and Vysotsky, V.A., 'Introduction and Overview of the Multics', Proceedings of 1965 FJCC, vol 27, pp 185-196, Spartan Books, 1965.
12. Corbato, F.J., 'System requirements for multiple access, time-shared computers', MAC-TR-3,
13. Dennis, J.B., 'Program Structure in a Multi-Access Computer, MAC-TR-11, May 1964.
14. Dennis, J.B., 'Programming Semantics in Multi Access Computation', MAC-TR-23, December, 1965.

15. Gibson, C.T., 'Time Sharing with IBM System/360 : Model 67', Proceedings of SJCC, vol 28, pp 61-78, 1966.
16. Saltzer, 'Traffic Control in Multiplexed Computing System', MAC-TR-30, Ph.D dissertation, 1966
17. Sackman, H., 'Time Sharing versus Batch Processing : The Experimental Evidence, Proceedings of SJCC, vol 32, pp 1-10, 1968.
18. Joslin, E.O., 'Computer Selection', Addison Wesley Publishing Company, 1968.
19. Sharpe, W.F., 'The Economics of Computers', Columbia University Press, 1969.
20. Rosenberg, A.M., ' Computer usage accounting for generalized time sharing system, CACM, vol 7, No. 5, pp 304-308.
21. Calingaert, P., 'System Performance Evaluation : Survey and Appraisal', CACM, vol 10, No. 1, pp 15-18, January, 1967
22. Meredith Smith, J., 'A review and comparison of certain methods of computer performance evaluation', Computer Bulletin, vol 12, No. 1, pp 13-18, May, 1968.
23. Wickens, R.F., 'A brief review of computer assessment methods', The Radio & Electronic Engineer, vol 36, No.5, pp 286-287, November 1968.
24. Drummond, Jr., M.E., 'A perspective on system performance evaluation', IBM Systems Journal, vol 8, No. 4, pp 252-263, 1969.
25. Arbuckle, R.C., 'Computer Analysis and Thruput Evaluation', C & A, vol 15, No. 1, pp 13 , January 1966.
26. Kerry, D.W., 'Choosing Computers for the Post Office', Computer Bulletin, vol 10, No. 4, pp 12, March 1967.
27. Dijkstra, E.W., 'Recursive Programming', from 'Programming Systems & Languages' ed. Saul Rosen, Computer Science Series, McGraw-Hill Book Company, 1967

28. Dijkstra, E.W. 'Structure of THE Multiprogramming System', CACM, vol 11, No. 5, May 1968.
29. Wilkes, M.V., 'Time Sharing Computer Systems', American Elsevier Publishing Co., 1969.
30. Gruenberger, F., 'Are Small, Free Standing Computers Here to stay', Datamation, April 1966, pp 67-68
31. Knight, K.E., 'A Study of Technological Innovation - The Evolution of Digital Computers', Ph.D. dissertation, Carnegie Institute of Technology excerpts in 'The Economics of Computers' by Sharpe [19], pp 312.
32. Hillegass, J.R., 'Standardized Benchmark Problems Measure Computer Performance', C & A, vol 15, No. 1, pp 16-19, January 1966.
33. Joslin, E.O. and Aiken, J.J., 'Validity of basing computer selection on bench mark results', C&A, vol 15, No. 1, pp 22, January, 1968.
34. Corbato, F.J., 'A paging Experiment with the MULTICS System', MAC-M-384, July 1968.
35. Hellerman, H., 'Digital Computer System Principles', Computer Science Series, Mc-Graw Hill Book Company 1967
36. Woodrum, L.J., 'A model of floating buffering', IBM System Journal, vol 9, No. 2, pp 118-144, 1970.
37. Neilson, N.R., 'The simulation of time sharing systems', CACM, vol 10, No. 7, pp 397, July 1967.
38. Denning, P.J., 'Working Set Model for Program Behaviour', CACM, vol 11, No. 5, May 1968.
39. Gecsei, J., Slutz, D.R. and Traiger L., 'Evaluation techniques for storage hierarchies', IBM System Journal vol 9, No. 2, pp 78-117, 1970
40. Baston, A., Shy-Ming Ju and Wood, D.C., 'Measurement of Segment Size', CACM, vol 13, No. 3, pp 155-159, March 1970.
41. Randell, B., 'A note on storage fragmentation and program segmentation', CACM, vol 12, No. 7, pp 365-369, July 1969.

42. Belady, L.A., Nelson, R.A. and Shedler, G.S., 'An anomaly in spare-time characteristics of certain programs running in a paging machine', CACM, vol 12, No. 6, pp 349-353, June 1969.
43. Denning, P.J., 'Queuing Model for File Memory Operations', MAC-TR-21, Master's Thesis, October 1965.
44. McKinney, J.M., 'Survey of analytical time-sharing models', Computing Surveys, vol 1, No. 2, pp 105-116, June 1969
45. Klienrock, L., and Coffman, E.G., 'Computer scheduling methods and their counter measures', Proceedings of 1968 SJCC,
46. Klienrock, L., 'A continuum of time-sharing scheduling algorithms', Proceedings of 1970, SJCC, pp 453-458.
47. Greenburger, M., 'The Priority Problem', MAC-TR-22, November 1965.
48. Scherr, A.L., 'An Analysis of Time Shared Computer Systems', MAC-TR-18, Ph.D. dissertation, June 1965.
49. Muthukrishnan, C.R., 'Analysis and Conversion of Decision Tables to Computer Programs', Ph.D dissertation, Electrical Engineering Department, Indian Institute of Technology, Kanpur, July, 1969.
50. Corbato, F.J., 'Sensitive issues in the design of multi use systems', MAC-M/383, December 1968.
51. Radhakrishnan, T., Rajaraman, V., Nori, K.V., 'Analytical models for computer systems : a survey', Shortly coming as a Technical Report, Computer Centre, Indian Institute of Technology, Kanpur.
52. Alen, A., Pritsker, B., Kiviat, P.J., 'Simulation with GASP II', Prentice-Hall Inc., 1969.
53. Markowitz, H.M., Hauser B., Karr, H.W., 'SIMSCRIPT; A simulation programming language', Prentice-Hall, Inc., 1963.

54. - 'General Purpose System Simulator III',
Users Manual, IBM Application Program,
Form H.20-0163-1
55. Greenburger, M., 'The OPS-1 Manual', MAC-TR-8, May 1964.
56. Buxton, J.N., (Ed), 'Simulation Programming Languages',
Proceedings of the IFIP Working Conference
on Simulation Programming Languages, North
Holland Publishing Company, 1967.
57. Teichreow, D., and Lubin, J.F., 'Computer Simulation -
Discussion of the technique and comparison
of languages', CACM, vol 9, No. 10, pp
723, October 1966.
58. Neilson, N.R., 'Computer Simulation of Computer System
Performance', Proceedings of 22nd National
Conference, ACM, pp 581, Academic Press,
1967.
59. Singh, C.V., 'Design of an Educational Computer and
General Purpose Logic Simulation Program',
M.Tech Thesis, Electrical Engineering
Department, Indian Institute of Technology,
Kanpur, August 1968.
60. Bucholz, W., (Ed), 'Planning a Computer System; Project
Stretch', McGraw-Hill Book Company, 1962.
61. Ihrer, F.C., 'Computer Performance Projected through
Simulation', C&A, pp 22-27, April 1967.
62. Mathai, Joseph, 'Performance of Memory Hierarchies',
Ph.D. dissertation, Mathematical Laboratory,
Cambridge University, 1968.
63. Liptay, J.S., 'Structural aspects of the System/360, Model 85:
II The Cache, IBM Systems Journal, vol 7,
No. 1, pp 15-21, 1968.
64. Conti, C.J., Gibson, D.H. and Pitkowsky, S.H., 'Structural
aspects of the System/360 Model 85 : I
General Organisation', IBM Systems Journal,
vol 7, No. 1, pp 2-14, 1968.
65. Bucholz, W., 'Bibliography on system performance evaluation,
IEEE Computer Group News, May 1969.

66. Katz, J.H., 'An experimental model for System/360'
CACM, vol 10, No. 11, pp 694, November 1967.
67. Hutchinson, G.K., 'Some problems in simulation of multi-processor systems', Proceedings of IFIP Working Conference on Simulation Languages, Buxton, J.N. (ed), , 1968.
68. Holland, F.C., Merikallio, R.A., 'Simulation of Multi-Processor Systems using GPSS', IEEE Transactions on Systems Science and Cybernetics, vol SSC4, No. 4, November 1968.
69. Greenburger, M., Jones, M., 'On line incremental simulation' Proceedings of IFIP Working Conference on Simulation Languages, Buxton, J.N. (ed), 1968.
70. Rosin, R.F., 'Determining a Computer Centre Environment', CACM, vol 8, No. 7, pp 463-468, July 1965.
71. Apple, C.T., 'The program monitor, a device for program performance measurement', Proceedings of 20th National Conference of ACM, pp 65-75, August, 1965.
72. Bonner, A.J., 'Using system monitor output to improve performance', IBM Systems Journal, vol 8, No. 4, pp 290-298, 1969.
73. Keefe, D.D., 'Hierarchical Control Programs', IBM Systems Journal, vol 7, No. 2, pp 123, 1968.
74. - IBM 7040-7044 Principles of Operation, Form A22-6649-6.
75. - IBM 7040 and 7044 Data Processing Systems; Student Text, Form C22-6732-2
76. - IBM 7040/7044 Operating System (16/32K): Programmer's Guide, Form C28-6318-6
77. - IBM 7040/7044 Operating System (16/32K): Input/Output Control System.
78. - IBM 7040/7044 Operating System (16/32K): System Programmer's Guide, Form C28-6339-5

79. - System Maintenance File, Computer Centre,
Indian Institute of Technology, Kanpur
80. Klienrock, L., 'Sequential Processing Machines (SPM)
Analysed with a Queuing Theory Model
JACM, vol 13, No. 2, pp 179-193, 1966.
81. Foley, J.D. 'A Markovian Model for the University of
Michigan Executive System', CACM, vol 10,
No. 9, p 584, September 1967.

CODES USED

CACM	-	Communications of Association for Computing Machinery.
C & A	-	Computers & Automation
FJCC	-	Fall Joint Computer Conference, AFIPS.
SJCC	-	Spring Joint Computer Conference, AFIPS.
JACM	-	Journal of Association for Computing Machinery.
MAC-TR-	-	Project MAC, Technical Report, MIT, Cambridge, Massachussets, USA
MAC-M	-	Project MAC, Memo, MIT, Cambridge, Massachussets, USA.

ERRATA

<u>Page</u>	<u>Line</u>	<u>Error</u>	<u>Correction</u>
4	4	resource provide	- resource provided
5	1	interchange .bility	- interchangeability
	11	changes percieved	- changes perceived
9	19	competitive offors	- competitive offers
10	8	numerious	- numerous
	9	reference to	- references to
	14	the devising	- devising
11	17	computer is in no way	- computers is in no way
22	12	the steady state	- the steady
	19	the average the average	- the average
23	2	indication the load	- indication of the load
25	7	appraisal	- appraisal
37	14	hence the an...ly	- hence the anomaly
42	17	in compari... CPUs	- in comparison of CPUs
61	3	comment, that	- comment that,
64	21	issues one	- issue one
69	25	evalualive	- evaluative
70	14	problems.	- algorithms.
	15	problem can be	- can be
76	22	sufficient crtical	- sufficient critical
77	14	be a maximum.	- be maximum.
93	22	(rranging	- (r ranging
100	3	qu...s	- queues

119	7	have given	- has given
	8/9	have added	- has added
123	22	were just	- was just
124	21	one of two types. The	- one of two types: The
	22	yield a summary	- yields a summary
	23	experiment. And	- experiment and
125	1	which record	- which records
127	1	tec.nique	- technique
	19	that have made	- that has made
128	2	necessitates	- necessitates
139	1	is form	- is to form
	2	Statistical about	- Statistics about
140	3	imple, ented	- implemented
141	5,7,24	accumulation	- accumulation
	10	than the minimum	- other than the minimum
146	12	to be technique	- to be a technique
149	14	to c..trol system	- to central system
	20	to be easily mask	- to easily mask
150	4	either by a giv	- either by a
151	13	instal ation	- installation
156	13	presented for this	- presented; this
157	9	immensity of problem-	immensity of the problem
	23	comparitive	- comparative