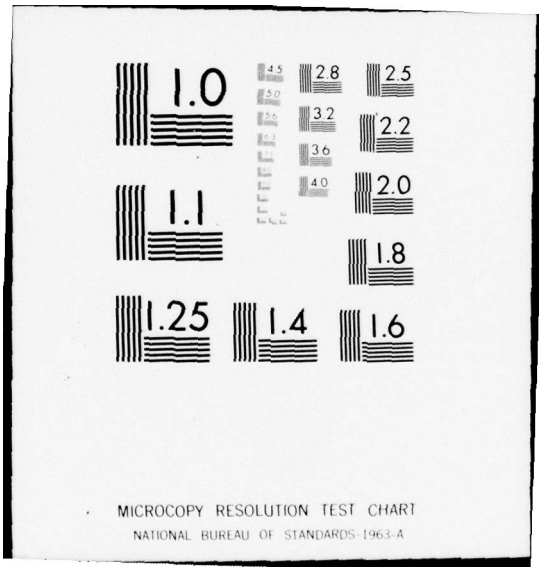
AD-A039 719 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
QUEUEING NETWORK MODEL FOR PERFORMANCE EVALUATION OF THE DECSYS--ETC(U)
MAR 77 L E MCKENZIE

UNCLASSIFIED AFIT/GCS/EE/77-1 NL

1 OF 3
ADA039719


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

UNITED STATES AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

GCS/EE/77-1

QUEUEING NETWORK MODEL FOR PERFORMANCE EVALUATION OF THE DECSYSTEM-10

THESIS

GCS/EE/77-1 Leslie E. McKenzie,Jr.
Captain USAF

AFIT/GCS/EE/77-1

14

6

QUEUEING NETWORK MODEL FOR PERFORMANCE EVALUATION OF THE DECSYSTEM-10.

9 Master's THESIS,

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

10 Leslie E. McKenzie, Jr

Capt USAF

Graduate Computer Science

11 March 1977

12 199 p.

Approved for public release; distribution unlimited.

012 225

Preface

This report is the result of an effort to assist personnel at the Air Force Avionics Laboratory (AFAL), Wright-Patterson AFB, Ohio in the development of tools and techniques which can be used in computer performance evaluation (CPE) of their DECsystem-10. A closed queueing network model which I hope will be beneficial in performance evaluation of the DECsystem-10 at the Avionics Laboratory is presented in this report. Appendix A contains a copy of the FORTRAN program which I wrote to implement this model. Given the constraints and assumptions of the model, this program can be used in performance evaluation of any time-sharing computer system.

I would like to thank Captain Bob E. Baker of AFAL/AAF-2 and Michael E. Price, DECsystem-10 Software Specialist at the Avionics Laboratory, for their support and assistance during this project. Finally, I would like to thank Dr. Gary B. Lamont for his advice and leadership as my thesis advisor.

Leslie E. McKenzie, Jr.

Contents

Contents

Contents

List of Figures

List of Tables

Abstract

A closed queueing network model of the DECsystem-10 at the Air Force Avionics Laboratory (AFAL), Wright-Patterson AFB, Ohio was developed. This model was developed to provide personnel at the Avionics Laboratory with a computer performance evaluation (CPE) tool which they could use to improve the performance of their DECsystem-10.

The hardware configuration and operating system features of the DECsystem-10 are discussed in this report. Features which affect performance are emphasized. Then, a review of computer performance evaluation and the CPE tools and techniques currently available is presented. Of these tools and techniques, *simulation and analytic modeling* are investigated as the two major approaches to computer system modeling.

A closed queueing network model which can be used to evaluate performance of the DECsystem-10 is described. This model can represent different types of time-sharing users and system resources with general service time distributions. The model can also represent several different queueing disciplines for system resources.

Validation experiments indicated that the closed queueing network model described in this report is a viable CPE tool for the DECsystem-10. A possible improvement to the model would be the inclusion of the effect that swapping has on performance measures in time-sharing computer systems.

QUEUEING NETWORK MODEL FOR PERFORMANCE EVALUATION OF THE DECSYSTEM-10

I. Introduction

During the last decade, the use of information processing systems by the Air Force has steadily increased. Today, information processing systems play an essential role in the operation of most Air Force organizations. This increased use of information processing systems in the support of various Air Force missions can be largely attributed to the development and implementation of multiprogramming, multiprocessing, time-sharing, virtual memory, real-time and networking concepts. Although beneficial to the Air Force, the versatility of information processing systems resulting from the implementation of these concepts has been accompanied by the increased cost and complexity of these systems. While the high cost of these systems necessitates their efficient and effective use, their complexity has unfortunately created major problems for Air Force automated data processing managers.

Review of Computer Technology Development

During the 1950s and 1960s, the Air Force's use of information processing systems was limited by the state of the art of computer technology. Computer technology was still in its infancy. The types of computers available were few, and operating systems were practically nonexistent. The

available computers were basically uniprocessor, serial processing, batch systems. The basic hardware configuration of these computers consisted of a memory unit, processor, card reader, line printer, and possibly a number of tape drives.

Hardware configuration and operating system simplicity limited the performance capabilities of computers during this time period. However, this simplicity also provided Air Force automated data processing (ADP) managers with complete managerial control of their computers. Several aspects of this control are as follows:

1. Since all tasks (jobs) were initiated through the card reader, the ADP manager could prioritize and schedule tasks (jobs) in agreement with predetermined organizational objectives.
2. The time required to execute a given workload could be calculated accurately. Since tasks were executed serially, the time needed to execute any workload was simply the sum of the time required to process each task composing the workload.
3. In a similar manner, the maximum workload that the computer was capable of supporting could be calculated.
4. The "turnaround time" for a task could be calculated from its scheduling priority and the processing time requirements of backlogged tasks since the processing time of a given task was fairly constant.
5. Two expressions of computer utilization were also obtainable. The utilization of the computer by a

task was simply the time that the computer processed a given task relative to the total time that the computer processed tasks. Total computer utilization was simply the total time that the computer processed tasks relative to its total processing time, idle time, maintenance time, and repair time.

6. Workloads which exceeded the processing capabilities of the computer were easily measured by the physical backlog of tasks.

These controls and measures enabled ADP managers to control the performance of their computers in processing tasks to accomplish organizational objectives. Using these measures, ADP managers were also able to predict the performance limitations of their computers. Unfortunately, the only solution to workload backlogging was the reduction of the workoad or the purchase of a new computer.

Since the late 1960s, the field of computer technology has exploded. New designs and concepts are barely implemented before they are outdated. In a few years the computer industry has progressed from production of uniprocessor, serial processing, batch systems to production of multiprogramming, multiprocessing, time-sharing systems. In addition, the implemented concepts of virtual memory, real-time processing, and networking have only made complex computer systems more complex.

One key to the successful implementation of advanced computer concepts has been the development of sophisticated hardware.

The development of high speed, low cost memory has made large memory units which support multiprogramming and multiprocessing concepts technically and economically feasible. In addition, a variety of peripheral devices including disks, drums, teletypes, cathode-ray-tube terminals, graphics displays, and plotters have been developed to enhance the input/output (I/O) and communication capabilities of computer systems. The concepts of interrupt driven I/O, separate I/O and data channels, and direct memory access have increased data transmission capabilities in computer systems. Register memories, instruction lookahead, and instruction pipelining have been successfully used to increase the processing speed of the central processor.

Since modern hardware is costly and complex, computer systems are designed modularly. In modular computer systems, various hardware components such as memory units, processors, and peripheral devices operate independently of each other. Communication between hardware components is handled asynchronously, and multiple components may service requests simultaneously. For example, while a processor is executing an instruction it could be fetching a second instruction from memory, and a disk unit could be transferring data into memory for later use by the processor. Thus, the probability of high speed hardware components being idled while waiting for a response from lower speed hardware components is reduced. Since idle times are minimized, the utilization of individual hardware components (system resources) and total system processing capability is increased.

Another advantage of hardware modularization is the configuration flexibility of computer systems. Hardware components can be used in the configuration of various size systems based on workload requirements. As workloads change and technical advances outdate certain hardware components, computer systems can be updated by adding or replacing components without incurring the cost of a completely new system.

A second key to the successful implementation of advanced computer concepts has been the development of sophisticated operating systems. Typically, the operating system controls the tasks within the computer system. It allocates system resources to tasks and handles service requests of the tasks. Service requests may range from providing tasks with the time of day to performing data transmission between the tasks and peripheral devices. Through the allocation of system resources (memory, processing time, disk space, etc.) to tasks, the operating system controls which tasks receive service and which system resources provide that service. Therefore, the utilization of the various hardware components and the workload flow through the system are dependent upon the algorithms which are used by the operating system to allocate resources to tasks.

Many resource allocation algorithms of varying complexity have been developed and implemented within various computer systems. However, all resource allocation algorithms have common goals. Several goals of resource allocation algorithms are to maximize the total processing capabilities of the computer

system, to minimize the time that each task spends in the computer system, and to maximize the utilization of system resources. However, since each organization which uses a computer system has different data processing requirements and priorities, resource allocation algorithms must also provide the owner of the system with the flexibility to alter these goals to reflect his requirements and priorities.

Although the resource allocation algorithms of each computer system must reflect the unique processing requirements of its owner, it is economically infeasible to develop specialized resource allocation algorithms for each organization which owns a computer system. Therefore, resource allocation algorithms are designed so that during processing, the algorithms are controlled by a set of parameters. These parameters usually have default values which have been established by the computer manufacturer to be suitable for the average owner. However, the owner of an individual computer system can alter the default values of these parameters. Examples of the parameters which the owner of a computer system can alter include the following:

1. The maximum number of active tasks allowed in the computer system (multiprogramming level).
2. The maximum amount of memory available to specified users.
3. The minimum amount of time a task must remain in main memory before it becomes a candidate to be swapped out to secondary memory (in-core protect time).

4. The maximum amount of processing time allocated to a task before it is preempted by another task (quantum processing time).
5. The priority of a task relative to other active tasks (priority structure).

By judicious assignment of values to such resource allocation parameters, the owner of a computer system can insure that predetermined processing requirements are achieved. These requirements could include preferential service for certain users, discrimination against tasks requiring unusual amounts of system resources (memory, processing time, etc), or maintenance of high or low utilization levels for certain system resources.

Computer Performance Evaluation as a Management Tool

The increased complexity of hardware and operating systems has greatly increased the processing capabilities of modern computer systems. However, proper management of such computer systems is also more complex. The simplistic controls and performance measures associated with earlier generations of computers are no longer available or viable. Today management of computer systems is less "machine" oriented and more "system" oriented. When using earlier generations of computers, the chief responsibility of Air Force ADP managers was to insure that the computer system hardware was mechanically operational. Today the responsibilities of ADP managers are not as simple. The configuration of the hardware, the resource allocation

algorithms of the operating system, and the processing priorities of various tasks can all be controlled by the ADP manager. Through his control, the ADP manager can insure that the hardware configuration, the operating system, and the mixture of tasks using system resources compliment each other to produce the most effective data processing capabilities possible. Therefore, the performance measurement of computer systems can no longer be based on hardware characteristics alone. Rather, the ability of the Air Force ADP manager to understand and control his computer system (hardware, operating system, scheduling of tasks) plays an increasing role in determining its actual performance.

The ability of Air Force ADP managers to control the performance of complex computer systems is related to their ability to answer various performance questions. Examples of these performance questions are as follows:

1. Since there are many computer systems currently available, how do I select the system that can best satisfy my processing needs?
2. What hardware configuration do I need? How many memory units, tape drives, disk drives, remote terminals, etc. do I need? How do I configure these hardware components to insure maximum performance?
3. How do I set resource allocation algorithm parameters to insure the maximum data processing capability?
4. If my current system can no longer provide required data processing capabilities, what can be done? Can

performance be improved by adjusting operating system parameters, modifying the tasks being processed, or purchasing new hardware? What system resources prohibits expansion of processing capability?

As the cost of modern computer systems and the demands for additional processing capabilities increase, Air Force ADP managers have become increasingly concerned with performance related questions like those listed above. Although the complexity of computer systems makes it difficult to obtain answers to these questions, the field of computer performance evaluation (CPE) does provide ADP managers with assistance in answering such performance questions. Computer performance evaluation has been defined as "any effort directed toward the optimization of the performance of a computing system" (Ref 1:20). Computer performance evaluation is a management strategy which is supported by a collection of tools and techniques. An ADP manager can use these tools and techniques to measure the performance of his computer system, to determine if and how performance can be increased, and to predict performance increases or decreases due to system modifications. These modifications could include changes to hardware configuration, changes to the operating system, or changes in data processing requirements.

Problem Statement

The Air Force Avionics Laboratory (AFAL) at Wright-Patterson AFB, Ohio currently owns a Digital Equipment Corporation DECsystem-10 computer system. This computer system is a multiprogramming, multiprocessing, time-sharing system with real-time processing capabilities. It is used in the development and validation of Air Force avionics software. Recently, the data processing demands of the DECsystem-10 users have been steadily increasing. As a result of this ever increasing workload, the DECsystem-10 is currently exhibiting many of the symptoms which are indicative of performance degradation. These symptoms include excessive response times for terminal users, the lack of unused disk space, excessive swapping, and a high degree of central processor inactivity.

Unfortunately, the causes of performance degradation are not nearly as evident as are the symptoms of performance degradation. The causes of performance degradation can include inefficiencies in hardware configuration, inefficiencies in the resource allocation algorithms of the operating system, and an unbalanced or excessive workload. To identify the causes of performance degradation and to implement the necessary changes to improve performance requires a complete computer performance evaluation of the DECsystem-10.

Scope

The time limitations of this thesis effort prevent the accomplishment of a complete computer performance evaluation

of the DECsystem-10 at the Air Force Avionics Laboratory (AFAL). Therefore, this thesis effort is limited to assisting Avionics Laboratory personnel in the development of computer performance evaluation tools and techniques which can be used in performance evaluation of their DECsystem-10. A review of available CPE tools and techniques with emphasis on modeling techniques (simulation and analytic modeling) is presented in this report. An analytic model of the DECsystem-10 is also presented. This model can be used to optimize the performance of the DECsystem-10 based on its current workload and to predict performance changes due to changes in its workload, resource allocation algorithms, or hardware configuration.

Approach

To assist Avionics Laboratory personnel in developing CPE tools and techniques which can be effectively used in performance improvement efforts, the following activities were undertaken during this thesis effort:

1. The DECsystem-10 was analyzed to determine its major components, to identify the options and parameters which could be changed to improve performance, and to identify the CPE tools and techniques that are currently available at the Avionics Laboratory.
2. Computer performance evaluation was reviewed to determine which CPE tools and techniques would most effectively aid in performance improvement of the DECsystem-10.

3. An analytic model of the DECsystem-10 which can be used in performance improvement efforts was implemented.

4. The analytic model was validated for the DECsystem-10.

Order of Presentation

Chapter II of this thesis describes the DECsystem-10 computer system at the Avionics Laboratory. This description includes the current hardware configuration and the architecture of the operating system. The discussion of the hardware configuration includes a brief overview of the memory, processor, peripheral device, and communication channel components. The major modules of the operating system are reviewed along with its task scheduling and resource allocation algorithms.

Chapters III, IV, and V present a review of the field of computer performance evaluation. This review identifies those CPE tools which are most effectively used in performance improvement efforts. Chapter III reviews the performance measures, tools, and techniques used in computer performance evaluation studies. Chapter IV examines the various simulation modeling techniques, the advantages and disadvantages of simulation modeling, and available simulation languages. Chapter V examines the development of analytical modeling techniques, the advantages and disadvantages of analytical modeling, and a review of applications of analytical models to computer systems.

Chapter VI and VII present the analytical model implemented for performance evaluation of the DECsystem-10. This is a closed queueing network model with different classes of customers and four types of service centers. The description of the model is presented in Chapter VI while Chapter VII presents the computational algorithms used to implement the model and to calculate performance measures.

Chapter VIII presents a description of how the DECsystem-10 computer system can be represented by a closed queueing network model and the results of tests run to validate the queueing model against data collected from the DECsystem-10 at the Avionics Laboratory. Recommendations for continued work are also included in this chapter.

It is assumed that readers of this thesis have had at least one undergraduate course in both computer architecture and queueing theory.

II. DECsystem-10 Computer System

The first step in any computer performance evaluation effort is to gain an understanding of the system being evaluated (Ref 2:6). Therefore, a review of the DECsystem-10 owned by the Air Force Avionics Laboratory (AFAL) at Wright-Patterson AFB, Ohio is presented in this chapter. This review includes a description of the current hardware configuration and operating system architecture of this DECsystem-10. Identification of hardware components and resource allocation algorithms which affect performance is emphasized in this chapter.

The DECsystem-10 computer system at the AFAL is a multiprogramming, multiprocessing, time-sharing computer system with real-time processing capabilities. The DECsystem-10 is located in the simulation laboratory at the Avionics Laboratory. It is used in the development, simulation, evaluation, and validation of Air Force avionics software. In addition to providing local and dial-up time-sharing and real-time processing capabilities, the DECsystem-10 also supports local and remote batch processing. Remote job-entry service is supplied to users located at the China Lake Naval Weapons Center in California and the Armament Development Test Center at Elgin AFB, Florida, via leased common-carrier circuits.

Hardware Configuration

The DECsystem-10 is a dual processor model 1077 computer

system. Fig. 1 is a diagram of the hardware configuration of this system.

Processors. At the heart of the DECsystem-10 are the two KI10 central processing units. These two central processing units are designated the primary processor and the secondary processor. Both of these processors have access to all of the memory units. However, only the primary processor has all of the normal I/O devices attached to its I/O bus. These I/O devices include the line printers, plotter, terminals, disk drives, and tape drives. The I/O bus of the secondary processor is reserved for attachment to I/O devices associated with real-time applications.

The primary processor performs exactly the same operations as it would in a single processor system except for real-time processing. This includes scheduling and execution of user jobs, all I/O operations, and all resource allocation (swapping, core allocation, file management, etc.). The secondary processor handles all real-time processing requirements. When it is not processing real-time applications, the secondary processor executes normal time-sharing user jobs. However, the secondary processor is not allowed to handle I/O requests or any resource allocation. Therefore, it attempts to select for execution compute-bound jobs which are already in core and ready to execute. When a job being executed by the secondary processor requests an I/O operation, the job is stopped, requeued, and marked for execution by the primary processor only.

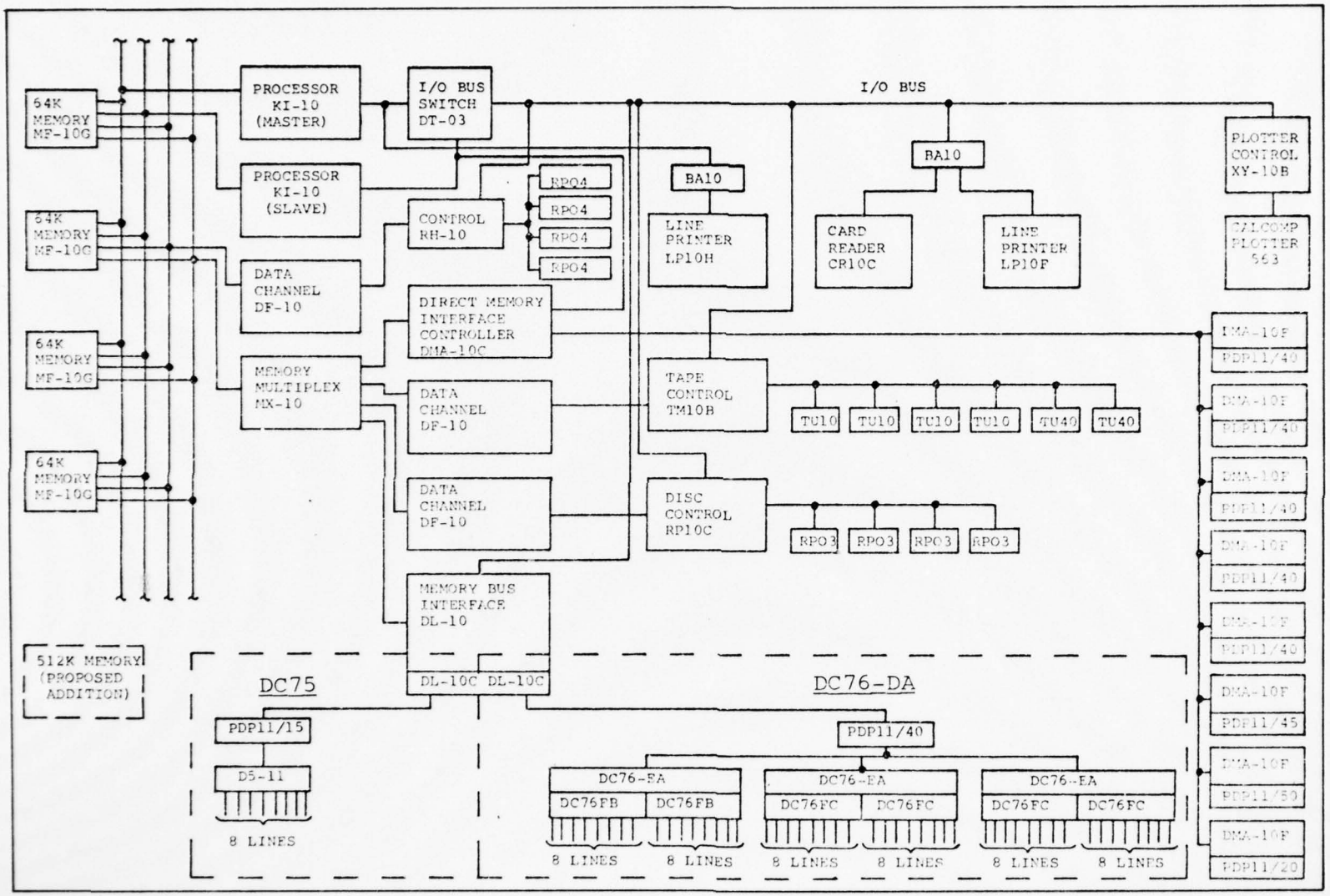


Fig. 1. DECsystem-10 Hardware Configuration

Since the secondary processor is prohibited from performing I/O operations and resource allocation, most of the normal hardware and software overhead associated with true multiprocessor computer systems is eliminated. In addition, a single version of the operating system can be used by both processors. An interlock is located in the scheduler to prevent both processors from trying to execute the same job at the same time. Otherwise, the two processors operate asynchronously while executing the same scheduler. Thus the primary/secondary configuration provides more processing capability than a single processor system but doesn't generate the hardware and software complexity of true multiprocessor systems.

Memory Units. The main memory of the DECsystem-10 system consists of four 64K word (36 bits/word) units of core memory. Each memory unit is further segmented into 512 word pages in support of virtual memory concepts. Each of the four memory units has four memory ports. These memory access ports provide direct access to any combination of processors and/or high-speed data channels. Two of these are connected to the master and slave processors. The two additional memory ports of each memory unit are connected to a DF10 data channel and to an MX10 memory multiplexer. The MX10 memory multiplexer can provide a maximum of seven additional channels (Ref 3:70c-384-01f).

Mass Storage Devices. The DECsystem-10 has four RP04 disk units. Each of these units has a storage capacity of

20 million 36-bit words. The average seek time for these disk units is 28 milliseconds, average rotational delay is 8.8 milliseconds, and the data transfer rate is 178.571 36-bit words per second. The rotational speed is 3600 rpm (Ref 3:70c-384-01g). These four RP04 disk units are controlled by a RH10 disk controller. In addition to handling the interface between the RP04s and the I/O bus and the data bus, the RH10 disk controller also allows overlapped positioner operation. Overlapped positioner operation allows the simultaneously positioning of two or more disk drives. One of the RP04 disk units is used primarily as swapping space while the other three units are used as user disk space.

The DECsystem-10 also has four RP03 disk units. The RP03 disks can store 10.24 million 36-bit words with an average transfer rate of 66,6667 36-bit words per second. Average access time for these disk units is 47.5 milliseconds which includes a 12.5 millisecond average rotational delay and a 35 millisecond head-positioning time (Ref 3:70c-384-01g). Because the memory units have only four memory ports, these disk units are interfaced with the memory units through the MX10 memory multiplexer rather than through a DF10 data channel as are the RP04s. One of the RP03 disk units contains the system files (compilers, assemblers, etc.), while the other three units are used as user disk space.

Real-time Processing. The DECsystem-10 has a DMA10 Memory Sharing Interface System which provides a memory

sharing link between the DECsystem-10 and eight PDP-11 minicomputers. The DMA10 provides the capability for the eight PDP-11's to be used as dedicated device controllers for the DECsystem-10. The DMA10 consists of a DMA10-C I/O controller and memory bus, and a DMA10-F memory interface unit for each PDP-11. A special UNIBUS with differential transmission techniques links each PDP-11 with its memory interface unit. This allows the PDP-11's to be up to 80 feet from the DECsystem-10. Fig. 2 shows the hardware configuration of this DMA10 Memory Sharing Interface System.

DECsystem-10 programs which interact with a PDP-11 must remain in main memory while they are being executed. Such programs can control a PDP-11 by placing necessary task parameters into a common communication area and issuing a PDP-11 interrupt to start the task. As the PDP-11 task executes, it may assemble any output data in the common communications area and interrupt the DECsystem-10 user program when it wishes the user program to take action. Thus, the PDP-11 may be controlled from the DECsystem-10 user program.

Hardware Effect on Performance

The hardware configuration of the DECsystem-10 plays an important role in the performance of the system as viewed by time-sharing users. Several characteristics of the hardware configuration described in previous sections are possible causes of performance degradation and should be investigated in

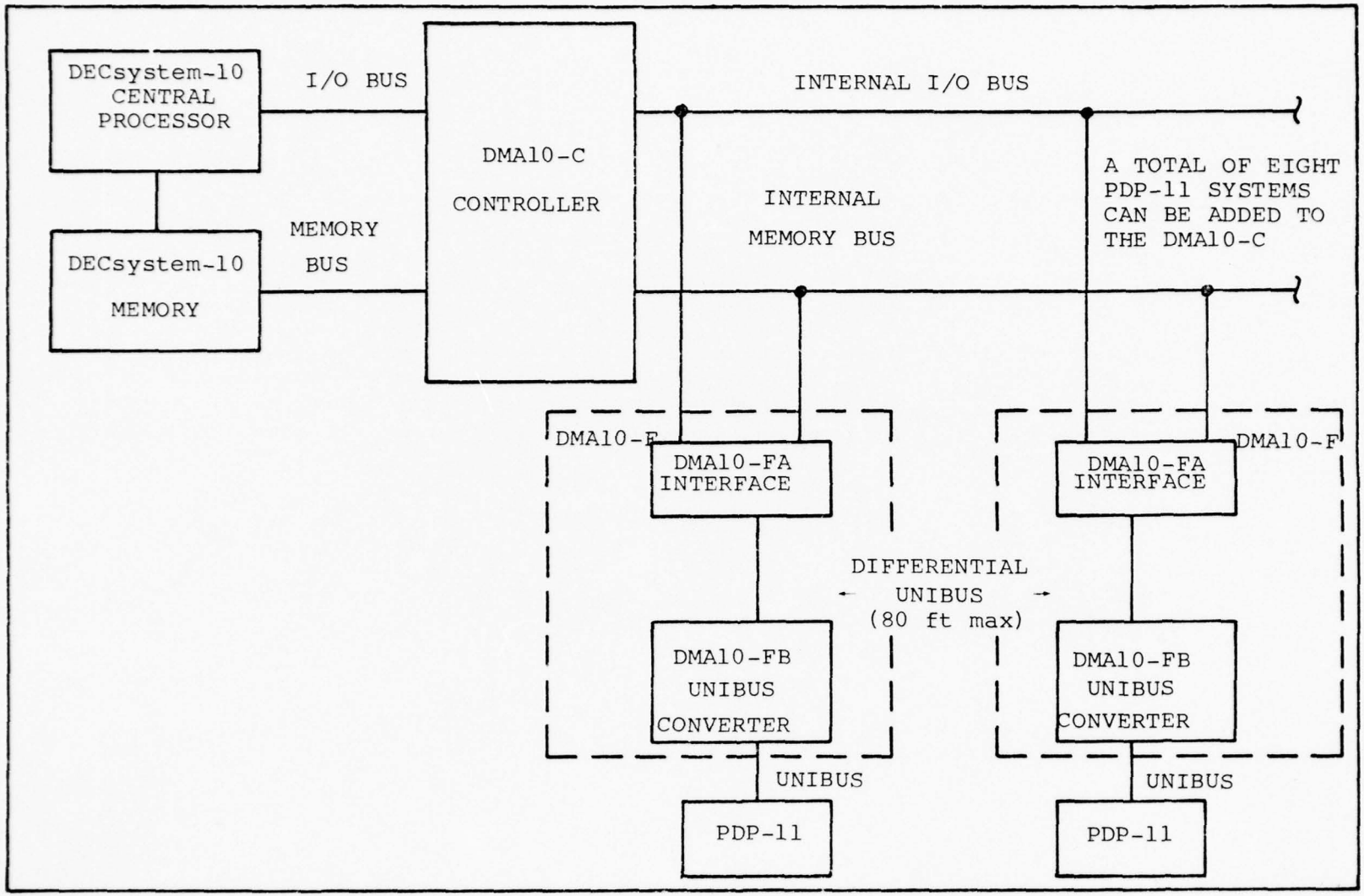


FIG. 2. DMA10 Memory Sharing Interface System (From Ref 4)

a performance evaluation of the DECsystem-10. Some possible causes of performance degradation which are related to the hardware configuration of the system are as follows:

1. When real-time jobs are being executed, they must remain in main memory. They also control the secondary processor. Therefore, a limited amount of main memory and only one processor is available to service time-sharing users when real-time jobs are being executed.
2. When real-time jobs are not being executed, the secondary processor executes time-sharing jobs. However, the secondary processor cannot handle the I/O requests of time-sharing users. The overhead that is required to interrupt the primary processor to handle all I/O operations may cause significant performance degradation if compute-bound jobs are not selected for execution on the secondary processor.
3. The amount of main memory available for time-sharing users may not be sufficient to contain enough jobs to keep the two processors active.
4. The limited number of memory ports requires that the swapping RP04 disk unit and the user RP04 disk units be connected to main memory through a single data channel. Since swapping I/O receives priority over time-sharing user I/O, swapping increases the delay for time-sharing user I/O. This delay may affect system performance significantly since jobs must

remain in main memory while their I/O requests are processed. Thus, main memory may be dominated by jobs which cannot be executed until their I/O is completed. Such a situation would lead to processor inactivity.

Operating System

The operating system of the DECsystem-10 is called the TOPS-10 MONITOR. The MONITOR is a collection of system programs which schedules and controls the operation of user programs, performs overlapped I/O, provides for context switching (overhead involved when a processor changes the user program it is executing), and allocates resources so that the computer's time is efficiently used. The MONITOR maintains control of all processing through a clock located in each processor. Each clock interrupts its processor 60 times a second which guarantees that the MONITOR regains control of the processor from user programs at least 60 times per second.

The MONITOR consists of many separate and more or less independent routines which are called according to events which occur within the system. Some of these routines operate on a regular cycle based on the clock interrupt (after each clock interrupt). Others are called only in response to system events such as I/O interrupts. Fig. 3 shows the relationship among the major modules of the MONITOR.

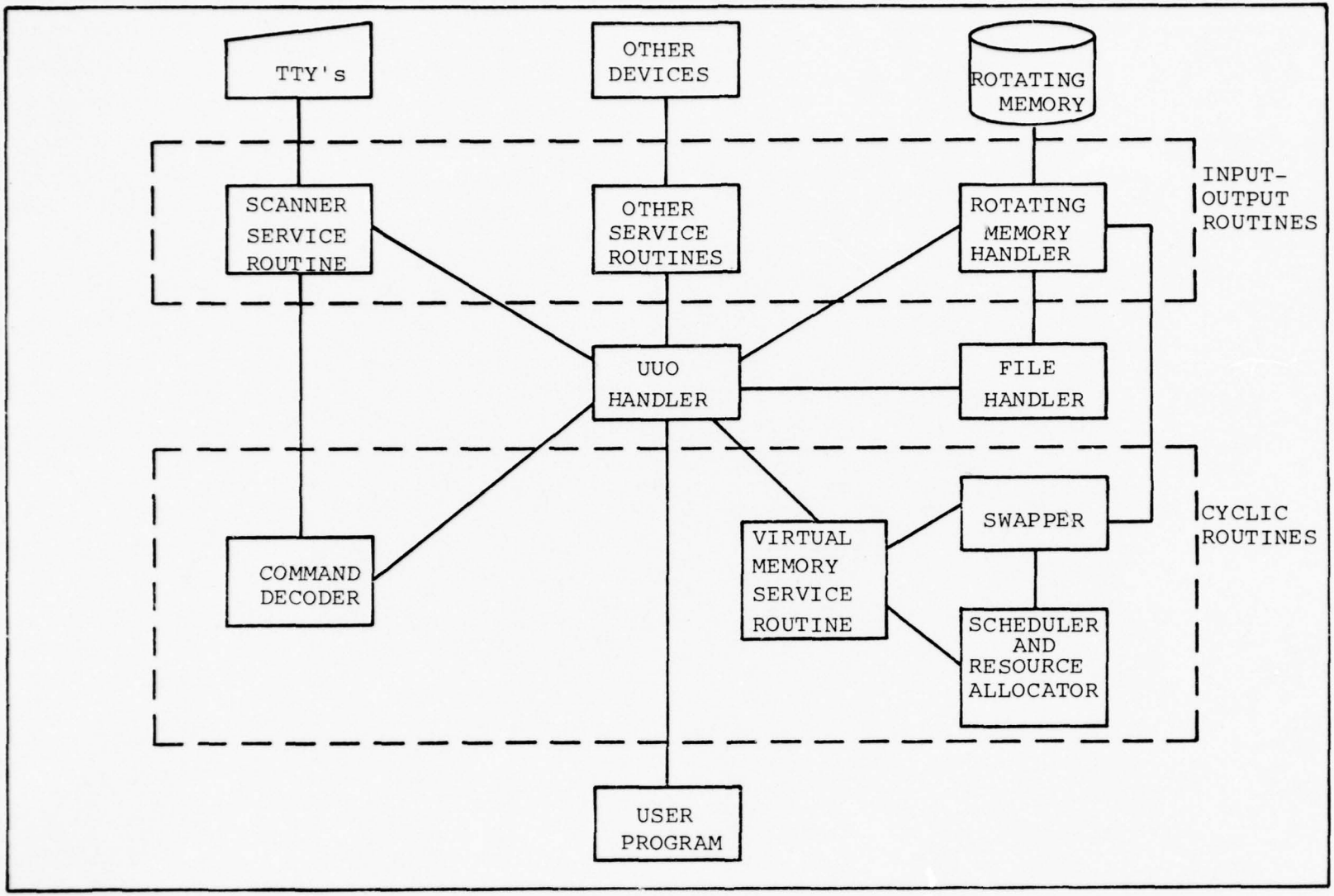


Fig. 3. DECsystem-10 MONITOR Structure (From Ref 4)

Cyclic Routines. After each clock interrupt, the MONITOR executes a predetermined set of routines. Fig 4 illustrates this MONITOR cycle. The first routine to be executed is the command decoder. The command decoder is the communications link between the user at his terminal and the operating system. When the user types commands and/or requests on his terminal, these commands are stored in an input buffer in the operating system. On each clock interrupt, control is given to the command decoder to interpret and process one command. If more than one user command is waiting to be serviced, they will be serviced at subsequent clock interrupts. If the command requires lengthy service time compared to the MONITOR cycle time of 1/60 second, it is initialized and processed in subsequent cycles as a regular task.

After the command decoder has been executed, the scheduler is executed. Since the DECsystem-10 is a multi-programming system, it allows several user tasks to reside in core simultaneously and to execute sequentially. It is the job of the scheduler to determine which user job will execute for the remaining time slice (remaining 1/60 second).

All jobs in the computer system are retained in ordered queues. These queues have various priorities that reflect the status of each job at any moment. The queue in which a job is placed depends on the system resource for which it is waiting; and, because a job can wait for only one resource at a time, it can be in only one queue at a time. Jobs may be

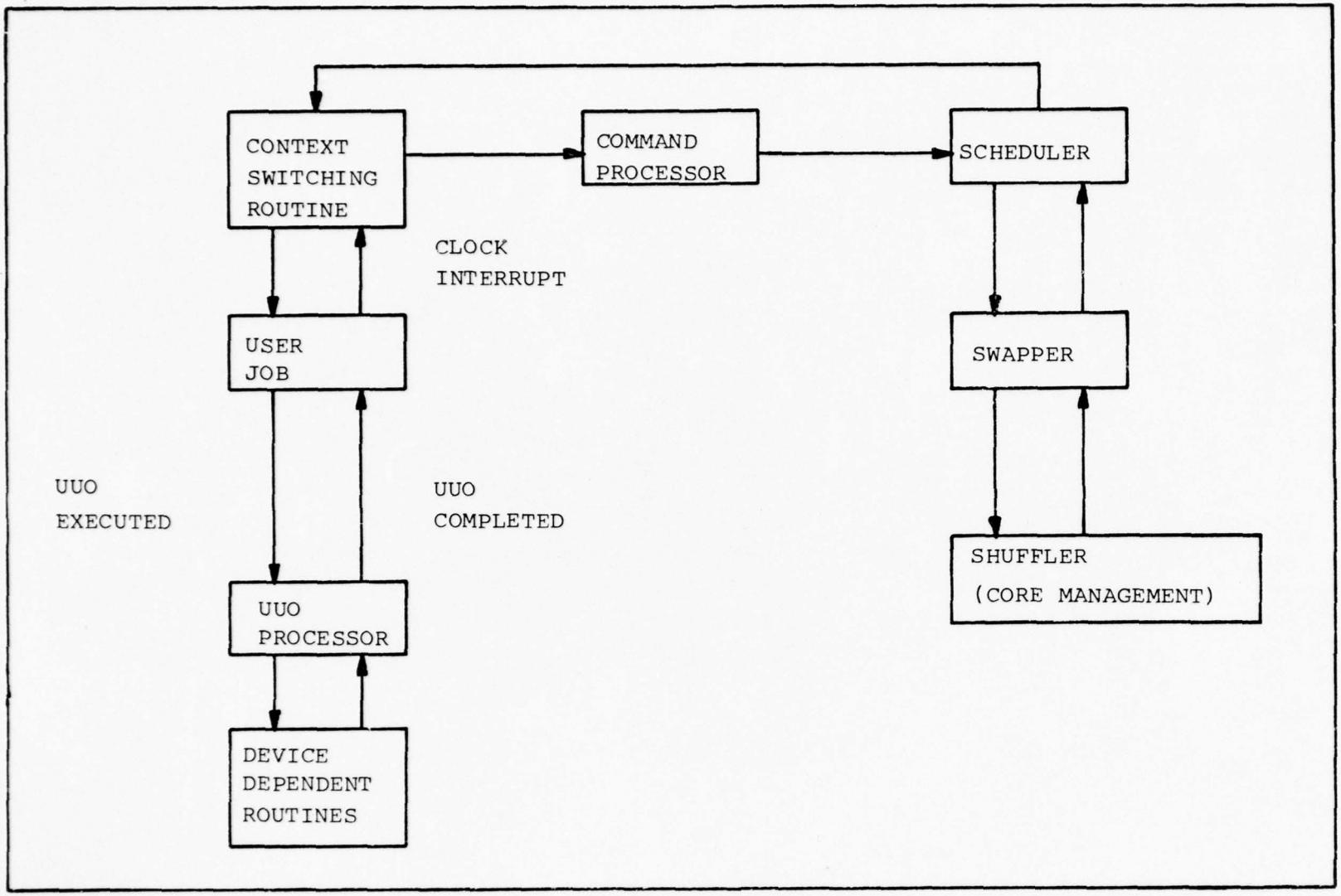


Fig. 4. DECsystem-10 MONITOR Cycle (From Ref 4)

in run queues, I/O wait queues, I/O satisfied queues, terminal I/O queues, sleep queues, etc.

The scheduler examines the various run queues in order to determine which job will be executed in the remainder of a time slice (remaining 1/60 second). If the scheduler is selecting a job to execute on the primary processor, it scans the run queues in the following order (Ref 4:302):

1. HPQ1 through HPQn are scanned forward for non-real-time jobs.
2. PQ1 is scanned forward.
3. PQ2 is scanned forward.

If the scheduler is selecting a job to execute on the secondary processor, it scans the run queues in the following order (Ref 4:302):

1. HPQ1 through HPQn are scanned forward for real-time jobs.
2. PQ2 is scanned forward.
3. PQ1 is scanned forward.

HPQ1 through HPQn are high priority queues specified at system generation time. Only privileged users' jobs are found in these queues, and they are reserved almost solely for real-time applications. If the scheduler finds a job in one of these queues, it is selected to execute.

PQ1 is one of the two queues holding normal jobs. Jobs are placed in the back of PQ1 when they first start running, come out of terminal I/O wait, or command wait. Some jobs that have been "sleeping" for over one second will also be

queued at the back of PQ1. The purpose of PQ1 is to provide fast response to jobs requiring a small quantum of central processing time. Once a job exceeds its quantum processing time, it is requeued to the back of PQ2. The scheduler attempts to execute jobs from the PQ1 run queue on the primary processor because they are more likely to request an I/O operation after a shorter period of processing than jobs in the PQ2 run queue are.

PQ2 is the queue for all jobs that have exceeded their quick-response quantum processing time. Jobs that reside in PQ2 are treated in a normal round-robin manner with a constant time-slice. Jobs will always requeue to the end of PQ2 when they exhaust their processing time slice. The scheduler attempts to execute jobs from the PQ2 run queue on the secondary processor because they are usually compute-bound jobs.

As part of the scheduling procedure, the swapper is executed during each clock interrupt. The swapper attempts to keep in core the jobs most likely to be runable. It determines if a job should be in core by scanning the various run queues in which a job may be. If the swapper decides that a job should be brought into core, it may have to take another job already in core and transfer it to secondary storage. Therefore, the swapper is not only responsible for bringing jobs into core but is also responsible for selecting jobs to be swapped out if necessary. In order to determine which jobs to swap in, the swapper scans the run queues in

the following order (Ref 4:302):

1. HPQ1 through HPQn forward.
2. PQ1 forward.
3. PQ2 forward.

In order to determine which jobs to swap out, the swapper scans the run queues in the following order (Ref 4:302):

1. PQ2 backward.
2. PQ1 backward.
3. HPQ1 through HPQn backward

Although the swapper attempts to keep in core the jobs that are most likely to be runable, the activities of the swapper are limited by the following three factors:

1. The swapper is never allowed to swap real-time jobs and other high priority jobs out of main memory.
2. A job can never be swapped out of main memory when it has an incompleted I/O request outstanding.
3. Once a job is swapped into main memory, it cannot be swapped out of main memory until its in-core-protect-time has been exhausted (as long as it remains in the run queues). The in-core-protect-time for a job in the DECsystem-10 is its Minimum Core Usage Function (MCUF). The MCUF is defined as MCUF = PROT * job size + PROTO (Ref 5:245). PROT is the Minimum Core Usage (MCU) multiplier and PROTO is the Minimum Core Usage (MCU) constant. When a job is in main memory, its MCUF value is only decreased when it is being processed or waiting for an outstanding I/O request to be completed.

Once the scheduler has selected a job to execute, the MONITOR then performs context switching. This is the last MONITOR action of each clock interrupt. Context switching is the restoration of all hardware registers to the conditions of the selected program when it was last interrupted. Once control is given to a user program after context switching, it may run until the next clock interrupt, or until it becomes "blocked".

Non-Cyclic Routines. The UUO-handler is responsible for accepting requests for service by the user tasks via software-implemented instructions known as programmed operators (UUOs). The UUO-handler is the only means by which a user task can give control to the operating system in order to have a service performed. Contained in the user program are operation codes which, when executed, cause the hardware to transfer control to the UUO-handler for processing. After the UUO request has been processed, control is returned to the user's program following the UUO.

Input/Output routines are called by the UUO-handler when needed as specified by the user programs. The I/O routines in cooperation with the UUO-handler allows buffered as well as unbuffered I/O. Since the operating system handles communication between the user program and the device, the user does not need to know the peculiarities of the various devices on the system.

The file handler manages user and system data. Thus, this data can be stored, retrieved, protected, and/or shared among other users of the computing system. All files in the system are uniquely identified not by physical address, but by name. Thus they can be accessed and identified by logical name rather than by physical address.

Operating System Effect on Performance

The resource allocation algorithms of the operating system of the DECsystem-10 play an important role in the performance of the system. Examples of the resource allocation algorithms of the operating system are those which define the scheduler and swapper. These were described in a previous section. The values of several parameters associated with the resource allocation algorithms of the operating system can be changed to control the performance of the system. Some of these parameters are as follows:

1. The total number of users that can be active at any time. This parameter is presently set at 40 at the Avionics Laboratory.
2. The quantum processing time that is given to a job in the PQ1 run queue before it is requeued into the round-robin PQ2 run queue. This parameter is presently set at 1/20 second at the Avionics Laboratory.
3. The quantum processing time that is given to a job is the PQ2 run queue before it is requeued to the back of the PQ2 run queue. The value of this quantum processing time is a function of the size of the job.
4. PROT and PROTO. These parameters can be set to control the swapping rate of the active jobs in the system.

Survey of Tools and Techniques Available

The Air Force Avionics Laboratory (AFAL) currently has three software programs that are used to measure the performance of the DECsystem-10. These three software programs are described in this section.

SYSTAT. The SYSTAT program is a system program which provides a summary report of the current state of the DECsystem-10 (Ref 6). The basic function of the SYSTAT program is to identify all current system users and to describe the status of these users. Information provided by SYSTAT includes the number of users currently logged onto the system, the identity of each user, the terminal identifier for each user, the program each user is executing, the size of the program, its state (swapped, executing, teletype, I/O wait, etc.), and the total execution time for each user. In addition, SYSTAT provides information concerning the status of all busy devices, file structure statistics, disk performance statistics, and the status of all remote stations.

TRACK. TRACK is a software program which can be used to monitor the progress and performance of individual jobs and the performance of the entire DECsystem-10. TRACK interrupts the system at time intervals specified by a CPE analyst and accumulation statistical data concerning system performance. The length of time TRACK is active and the number of interrupts per report are also controlled by the CPE analyst. Information gathered by TRACK during each report period is presented in a statistical distribution format. TRACK currently provides the following performance information:

1. Percent of time each processor is idle, and the percent of processor time that is lost because no runable jobs are in main memory.
2. The number of times context switching occurs.
3. The number of times user programs trap to the operating system.
4. The number of users logged on and their status (runable, teletype I/O, disk I/O, swapped, etc.).

METER. METER is a group of modules which allows CPE analysts to implement software into the operating system of the DECsystem-10 cleanly and efficiently and with a minimum of interference with the rest of the system (Ref 7). The METER facility provides the interface necessary for performance measurement modules to be added to the MONITOR as needed. The METER facility consists of three types of modules: 1.) meter points, 2.) meter point routines, and 3.) meter channel routines. Meter points are very small sections of code that act as sources of data when enabled, and there can be any number of these scattered throughout the operating system. Each time an active meter point is encountered, one word of data is supplied to a meter point routine which processes the word of data. This processing might include attachment of meter point or time identifying information to the word of data. The meter channel routine then delivers this information to a user program where it is stored in a buffer which is under control of the CPE analyst.

Summary

An overview of the DECsystem-10 at the Avionics Laboratory was presented in this chapter. Emphasis was placed on the hardware configuration and operating system features which play significant roles in determining the performance of the DECsystem-10. Several parameters which can be set to control the activities of the resource allocation algorithms of the operating system were identified. In addition, several programs which personnel at the Avionics Laboratory currently use to measure the performance of the system were discussed. It should be noted that these programs can only be used to measure the performance of the DECsystem-10. Therefore, in following chapters, the emphasis will be on CPE tools and techniques which can be used to optimize the performance of the system for a given workload and to predict performance changes due to changes in the workload, hardware configuration, or operating system of the DECsystem-10.

III. Computer Performance Evaluation

Since the early 1970s, interest in computer performance evaluation (CPE) has burgeoned. This interest has closely paralleled the development of computer technology. The increased variety, complexity, use, and cost of computer systems developed through the application of computer technology has greatly stimulated interest in computer performance evaluation. Both economic and data processing requirement considerations have necessitated that managers make wise decisions in the procurement and management of computer systems. Computer systems must adequately satisfy the data processing requirements of an organization while being operated cost-effectively and within budgetary constraints.

Unfortunately, the interest in and the economical importance of computer performance evaluation has not yet lead to the development of a theory of computer performance evaluation. There are still a number of obstacles to overcome before computer performance evaluation can be considered a science. These obstacles include "considerable differences among workers in performance in the definition of various performance measures, the scarity of analytic methods for relating various performance measures to each other, and the lack of general methods of scaling performance measures from one system to another" (Ref 8:2).

The variety of hardware configurations, operating systems, data processing requirements, and goals of computer performance

evaluation has proven to be a major hindrance to the establishment of a methodology for computer performance evaluation. Computer performance evaluation is conducted within a specific environment and for a specific purpose. Each environment, composed of the hardware configuration, the resource allocation algorithms of the operating system and the workload, is unique; and the reasons for evaluating computer performance vary. Therefore, it has been difficult to establish procedures and methods for computer performance evaluation which apply to all situations.

Although development of a theory of computer performance evaluation or even a methodology for computer performance evaluation has proven to be difficult, efforts in this area have contributed to the understanding and development of computer performance evaluation. Work in the CPE field within recent years has lead to the identification of several problems which are common to all CPE efforts, to the classification of computer performance evaluations based on their purpose, and to the development of a number of CPE tools and techniques which can be used by CPE analysts. To illustrate how recent advances in the CPE field can be applied in performance evaluation of the DECsystem-10 and to identify CPE tools and techniques which can be used most effectively in performance evaluation of the DECsystem-10, an overview of computer performance evaluation is presented in the remainder of this chapter. This overview includes a review of workload characterization and the selection of performance measures as

problems which are common to all CPE efforts, a description of the purposes of computer performance evaluation, a review of the performance goals of an organization, and a summary of the tools and techniques which can be used in computer performance evaluation.

Workload Characterization

"Computer performance evaluation is a complex process that involves many analyses and decisions" (Ref 9:1). The initial analysis that is required in all CPE efforts is the definition of the environment in which computer performance evaluation is to be performed. Computer performance evaluation is meaningless unless discussed in relation to a particular data processing environment. This environment is defined through the description of the hardware configuration of the computer system to be evaluated, its operating system (especially the resources allocation algorithms of the operating system), and its workload. If sufficient vendor documentation is available, the hardware configuration and operating system of any given computer system can be adequately described. The description of the DECsystem-10 presented in Chapter II is an example of the hardware configuration and operating system description required before any CPE effort is undertaken. Unfortunately, obtaining a description of the workload is not easy.

The workload processed by a computer system is a representation of the data processing requirements of the using organization. The workload is characterized by the type and

amount of system resources that are demanded by the using organization. Examples of parameters which are used in workload characterization are presented in Table I.

In modern computer systems, the demands for system resources vary widely and are very unpredictable. Resource demand fluctuation is related to the number of users of a computer system, the variety of their data processing requirements, their access to the resources of the system, and the resource allocation algorithms of the operating system. Workload fluctuations tend to be greater in time-sharing systems like the DECsystem-10 than in batch systems because each user has direct access to the computer system. This unpredictable workload fluctuation makes workload characterization extremely difficult.

Measures of Performance

The performance of a computer system is determined by its ability to respond adequately to the data processing requirements of its user organization (Ref 10:3). The ambiguity of "adequate response" allows different people to use different criteria to measure the performance of a computer system. While a hardware engineer may measure the performance of a computer system by the utilization levels of its various hardware components, an ADP manager may measure computer system performance by the total number of jobs executed in a given time period, and a programmer may measure computer system performance by the time required to process his program. All of these measures of performance

Table I

Workload Characterization

Workload Parameters	Description
Job CPU time	Total CPU time requested by a single job*
Job I/O requests	Total number of I/O operations requested by a single job
CPU service time	CPU time required to process a single CPU task
I/O service time	I/O time required to process a single I/O task
Interarrival time	Time between two successive requests for a system service
Priority	Priority assigned to a job by the user
Blocked time	Time a job is incapable of receiving CPU service
Memory requests	Amount of memory requested by a single job
Working set size	Number of pages of a single job that must be kept in the main memory
Locality of reference	Times for which all memory references made by a single job remain within a single page or a set of pages
User response time	Time needed by a user at an interactive terminal to generate a new request
User intensity	Processing time per request / user response time
Number of simultaneous users	Number of interactive users logged concurrently
Number in the system	Number of jobs or tasks being serviced or waiting in queues for system resources
Instruction mix	Relative frequencies of different types of instructions the system must execute

*In an interactive system, each terminal command is assumed to represent a new job.

(From Ref 9:12-13)

are valid. Therefore, an initial decision of every computer performance evaluation is the establishment of a set of performance measures upon which the evaluation can be based.

Performance measures are usually described as measures of effectiveness or efficiency. Effectiveness measures describe how the computer system handles data processing requirements from the viewpoint of the user. Efficiency measures describe how the computer system uses its resources to satisfy resource demands. Some performance measures of system effectiveness are presented in Table II. Some performance measures of system efficiency are presented in Table III.

Classification

The performance measures that are selected for use in a CPE effort depend on the purpose of the evaluation. Computer performance evaluations are undertaken for a number of reasons. The classical classification of the purposes of computer performance evaluation is selection evaluation, performance projection, and performance monitoring (Ref 11:79-80).

Selection Evaluation. Selection evaluation is the use of computer performance evaluation techniques to evaluate the performance of the hardware and system software of an existing computer. Selection evaluation can be used to classify computers according to performance and to select a computer system from a vendor when performance is a major criterion. Selection evaluation is thus limited to existing computer systems.

Table II

Measures of System Effectiveness

Performance Measure	Description
Throughput	Amount of useful work completed per unit of time with a given workload
Relative throughput	Elapsed time required to process given workload on system S1 / elapsed time required to process the same workload on system S2
Capability (capacity)	Maximum amount of useful work that can be performed per unit of time with given workload
Turnaround time	Elapsed time between submitting a job* to a system and receiving the output
Response time	Turnaround time of requests and transactions in an interactive or a real time system
Availability	Percentage of time a system is available to users

*In an interactive system, each terminal command is assumed to represent a new job.

(From Ref 9:16)

Table III

Measures of System Efficiency

Performance Measure	Description
External delay factor	Job turnaround time / job processing time
Elapsed time	Turnaround time of a job under multiprogramming / turnaround time of this job when it is the only job in the system
Gain factor	Total system time* needed to execute a set of jobs under multiprogramming / total system time needed to execute the same set sequentially
CPU productivity	Percent of time a CPU is doing useful work (used as a measure of throughput)
Component overlap	Percent of time two or more system components operate simultaneously
System utility	Weighted sum of utilization of system resources
Overhead	Percent of CPU time required by the operating system
Internal delay factor	Processing time of a job under multiprogramming / processing time of this job when it is the only job in the system
Reaction time	Time between entering the last character on a terminal or receiving the input in the system and receiving first CPU quantum
Wait time for I/O	Elapsed time required to process an I/O task
Wait time for CPU	Elapsed time required to process a CPU task
Page fault frequency	Number of page faults per unit of time

*System time is the time when at least one of the system's processors is busy.

(From Ref 9:17-18)

Performance Projection. Performance projection refers to the use of performance evaluation techniques to predict the performance of a nonexistent computer system. Performance projection is beneficial in the design of new system hardware and software. Using performance projection techniques, the performance of new designs can be evaluated during development rather than after implementation. Performance projection is also beneficial in predicting the performance impact of changes in the hardware, operating system, or workload of an existing system before the changes are implemented.

Performance Monitoring. Performance monitoring describes the use of computer performance evaluation techniques to collect data on the actual performance of an existing computer system. Such data can also be used to determine whether a computer system is operating within control limits established by the managers of the using organization. If control limits are being exceeded, this data can assist in determining the causes of the control limit excesses (Ref 12).

Performance Optimization. In addition to selection evaluation, performance projection, and performance monitoring, performance optimization is another important purpose of computer performance evaluation efforts (Ref 13:22). Performance optimization techniques are used to improve the performance of a given computer system without changing the hardware configuration or the workload of the computer system. Performance is optimized with respect to the adjustable parameters of the operating system (in-core-protect-time, quantum run time, multiprogramming level, etc.).

Performance Goals

Automated data processing (ADP) managers will sometimes initiate a computer performance evaluation for one of the specific reasons listed above. For example, the ADP manager might initiate a CPE effort to tune the resource allocation algorithms of the operating system for a given workload (performance optimization). However, the normal performance goals of an organization are not usually this limited in scope. The performance goals of an organization tend to be broad in scope and related to the performance measures of system effectiveness. For example, the ADP manager might wish to increase the response time (time-sharing system) or throughput level (batch system) of his computer system for a specific workload. Alternatively, he might wish to maintain an acceptable response time or throughput level while increasing the workload on the system.

To successfully achieve his performance goals, the ADP manager's CPE effort might necessitate performance optimization, measurement, and projection evaluations if not selection evaluation. To improve response time the ADP manager might initiate a performance optimization evaluation. If this approach proved unsuccessful, he might then conduct a performance projection evaluation to identify performance improving and cost-effective system upgrade alternatives. However, successful performance optimization and performance projection evaluations both include many of the concepts and techniques associated with performance monitoring. For example,

it is impossible to evaluate the performance effects of changes to the resource allocation algorithm parameters without measuring the performance of the system before and after the changes have been implemented. Thus, the achievement of the normal performance goals of an organization requires the development and use of a variety of tools and techniques which support computer performance evaluations conducted for a number of specific reasons.

Tools and Techniques

After the environment in which computer performance evaluation is to take place, the performance measures, and the performance goals have been established, various CPE tools and techniques can be used to identify the factors within the computing environment which affect the performance measures. Computer performance evaluation tools and technique can also be used to describe the relationship between these factors and the performance measures. The techniques used in computer performance evaluation are similar to analysis and evaluation techniques used in other fields of science and include statistical sampling, regression analysis, simulation, etc. (Ref 9:1).

A critical decision in any computer performance evaluation is the selection of the CPE tools and techniques to be used in the evaluation. Many of the CPE tools and techniques are complementary, and the achievement of performance goals usually requires the use of several of these tools and techniques.

The choice of the CPE tools and techniques to be used in a particular computer performance evaluation and the utility of these tools in the evaluation are determined by the performance goals of the using organization and the experience of the CPE analysts handling the computer performance evaluation. A summary of current CPE tools and techniques is presented below.

Instruction Mixes. Instruction mixes are used in the evaluation of the performance of central processing units (CPUs). The performance measure obtained from instruction mixes is the "average instruction time" for the CPU being evaluated. "Average instruction time" is obtained from the execution time of each instruction in the instruction set of the CPU weighted by a frequency distribution of the occurrence of the instructions in a workload (Ref 14:64). The frequency distribution of instruction usage can be obtained through statistical sampling techniques if the CPU exists within a computer system which can process the given workload. Otherwise, the frequency distribution of instruction usage for the workload must be estimated.

Instruction mixes have limited usage as CPE tools. Instruction mixes can be used in selection evaluation to compare the relative performance merit of various CPUs. They can also be used in performance projection to predict the performance of a new CPU relative to a given workload. However, instruction mixes have several major limitations. Generation of frequency distributions of instruction usage

is subjective unless the workload can be sampled while being processed on the CPUs being evaluated. Instruction mixes consider only one of the many performance characteristics of computer systems, and input/output instructions are usually not included in instruction mixes (Ref 11:82).

Kernal Programs. Kernal analysis is a technique which is used to model workloads. In kernal analysis, programs which represent a certain subset of the workload are actually coded. Portions of the workload selected for representation in kernal programs are activities that take the most time to execute, are executed the most frequently, or are critical to the execution of the rest of the workload (Ref 14:79-81). Kernal programs are timed based on a manufacturer's published execution times for each instruction contained in the kernal programs. These execution times are used as a performance measure in computer performance evaluation.

Although kernal programs are more beneficial in computer performance evaluation than instruction mixes, their utility as CPE tools is limited. Kernal programs include some performance characteristics of computers that aren't included in instruction mixes. Some of these performance characteristics are memory addressing logic, indirect addressing capabilities, and the number of available index registers (Ref 11:82). The performance measure of execution time can be used in the selection evaluation of various CPUs and the performance projection of new CPUs. Problems in using kernal programs as CPE tools

are the difficulty in determining the appropriate subset of the total workload to be represented in the kernal programs and the influence of the programmers' experience on the execution times obtained from the kernal programs (Ref 14:82-83).

Benchmarks. Benchmarking is the traditional technique used in computer performance evaluation to model workloads. A benchmark is "a job or set of jobs that represents a typical workload" of the computer system to be evaluated (Ref 9:55). When executed, a good benchmark should generate the same resource demands within the computer system that the workload being modeled would generate. Benchmarks are usually composed of jobs that already exist and are part of the workload. Selection of the jobs to be included in a benchmark can be accomplished through cluster analysis. All jobs in the workload are grouped into various classes based on their resource demands (workload characteristics). The job whose resource demands most closely approximate the resource demand characteristics of its class is selected for inclusion in the benchmark. The selected job is also weighted according to the percentage of the workload that it represents.

Benchmarks are commonly used in computer performance evaluations related to selection evaluation and performance optimization. Benchmarks can be run on different computer systems to obtain performance measures which can be used to compare the systems. Benchmarks can also be executed before and after resource allocation algorithm parameters are changed in order

to determine the effect that the changes have on performance measures. Benchmarks also have a limited application in performance projection studies. If proposed changes to a computer system are the only architectural difference between it and a second computer system, a benchmark could be executed on the second system to predict the effect that the proposed changes would have on performance measures. Established benchmarks can also be altered and executed to predict the effect that workload changes would have on performance measures.

Benchmarks have several disadvantages and limitations as CPE tools. The workload of modern computer systems is usually complex and variable. Therefore, development of reliable benchmarks is difficult and costly. Usually a benchmark must represent numerous types of jobs, and seldom is there one job that is truly representative of a job type. In addition, many jobs can not be used in benchmarks because they make permanent changes to the computer system when they are executed (update permanent files). Benchmarks are also usually system dependent and must be modified for each computer system evaluated (Ref 9:56, 11:84).

Synthetic Jobs. Benchmarks which are composed of synthetic jobs rather than application jobs can be used in computer performance evaluations. Synthetic jobs use resources and perform functions, but they do not do any useful work. A set of parameters which can be used to control the type and amount of resources that a synthetic job will use is

usually incorporated into the programs which represent synthetic jobs. These parameters usually control the amount of memory, amount of CPU time, number of I/Os, etc. that are requested by a synthetic job. Once the workload classification according to resource demands is determined, synthetic jobs with appropriate control parameter settings can be used to represent the "typical" job described by each resource demand class.

Some of the technical problems normally associated with benchmarks are solved through the use of synthetic jobs rather than application jobs in workload characterization. Benchmarks composed of synthetic jobs are more flexible and less costly to develop and maintain than benchmarks composed of application jobs. It is also easier to transport synthetic benchmarks between computer systems.

Software Monitors. Software monitors are programs which monitor the activities within a computer system. Since software monitors reside in memory, they have access to any information contained in the memory or CPU registers of the computer system being monitored. Software monitors can thus examine any of the tables maintained by the operating system. From these tables software monitors can obtain information which defines the state of the system at any given time. Software monitors can gather information on memory usage, length and contents of system queues, status of peripheral devices, etc.

Software monitors are either time-driven or event-driven. Time-driven software monitors interrupt the central processor

at specified time intervals and gather information concerning the state of the system at that time. Although the sampling rate is constant, the samples can be considered to be random since the sample rate is much slower than the rate at which the state of the system changes. Between samples, the state of the system is assumed to have unpredictably changed states many times. "Therefore, in computer sampling, randomness is obtained through the large and unknown number of changes of state between samples rather than through random timing of examinations " (Ref 15:24). The programs TRACK and SYSTAT, which were described in Chapter II, are time-driven software monitors for the DECsystem-10.

Event-driven software monitors are executed only when a specified event occurs. An event-driven software monitor gathers data related to the event which caused its execution or the state of the system when the event occurred. The use of event-driven software monitors requires that the operating system be modified to include traps (code to transfer control) to the software monitor. The implementation of such traps requires intimate knowledge of the operating system, and they may generate errors if they are not carefully coded and tested. Although event-driven software monitors are more difficult to implement than time-driven software monitors, they must be used if deterministic results rather than statistical results are required. The program METER, which was described in Chapter II, is an event-driven software monitor for the DECsystem-10.

The ability to obtain measures of performance is a vital requirement in almost every computer performance evaluation. For this reason, and because they are relatively cheap and easy to develop, software monitors are a very popular CPE tool. However, they do have several disadvantages. Since software monitors are programs, they must be supported by the system being monitored. They require memory, CPU time, and I/O capabilities. Thus, the resource demands of a software monitor may distort the very data that it is gathering. This phenomenon is called monitor artifact (Ref 9:82). To minimize monitor artifact, most software monitors gather data on-line but do any necessary data reduction or analysis after the completion of the monitoring period. Monitor artifact also limits the amount of data that can be gathered and the sampling rate of time-driven software monitors. Another disadvantage of software monitors is that they are system dependent. Since they obtain their data from the tables maintained by the operating system, they must be modified whenever the operating system is modified. They also can't be used on a system with a different operating system. The concepts of software monitoring are transportable but the code isn't.

Hardware Monitors. Hardware monitors are other tools which can be used to monitor the activities of a computer system. Hardware monitors have traditionally been self-supporting devices that use high impedence probes to sense electronic signals in the circuitry of the computer system being monitored. Thus, a hardware monitor is theoretically

capable of measuring any event which is represented by some electronic signal.

The basic hardware monitor consists of high impedence probes to sense signals, a combinational logic unit, counters, a magnetic tape for storage of data, and possibly some comparison capabilities (Ref 16:25-31). After a signal is sensed, it is routed through the combinational logic unit. In the logic unit, logical operations are performed between the signal and various other signals to indicate an event. For example, if one probe which measures processor activity and one probe which measures channel activitiy are fed into a "AND" unit, then the resultant event signal represents processor and channel activity overlap. These event signals trigger counters which accumulate the number of occurrences of the various events being monitored. Periodically, the contents of the counters are stored on the magnetic tape for later reduction and analysis. If a hardware monitor has comparison logic and bit-parallel probes, it can compare the contents of CPU registers (memory address register, instruction counter, etc.) with the contents of bound registers located in the hardware monitor to indicate the occurrence of an event of interest. This event could be the execution of a routine, for example.

Within recent years, the sophistication of hardware monitors has increased beyond the simple signal recorder described

above. Hardware monitors are currently available that are incorporated with a minicomputer. Memory modules are used as banks of counters, and the combinational logic, which had been programmed through plugboards, is software controlled. Data reduction and analysis are performed on-line to support comprehensive real-time displays of the recorded data.

Hardware monitors are important CPE tools because, like software monitors, they support the measurement requirements necessary in almost every computer performance evaluation. Hardware monitors also have two advantages over software monitors. Since hardware monitors only sense binary signals, they can be used with a variety of operating systems and a variety of computer systems. Although CPE analysts must be familiar with each system being evaluated, a new measurement tool does not have to be developed for each system. Since hardware monitors operate independently of the computer system being monitored, they do not inject any monitor artifact into the performance of the system. The major disadvantage of hardware monitors is their relatively high cost and the need for experienced engineers who can attach the probes to the appropriate locations in the hardware architecture of the computer being monitored. In addition, hardware monitors can't access some of the data available to software monitors (queue lengths, state of individual jobs, etc.).

Models. Modeling is one of the most popular and most effective CPE technique currently available. Some of the reasons that modeling is such an effective CPE technique are as follows (Ref 17:11):

1. Modeling provides a framework for testing the desirability of system modification.
2. Models of the system are easier to manipulate than the system itself.
3. Modeling expediates the speed with which an analysis of the system can be accomplished.
4. Modeling permits control over more sources of variation than direct study of the system would allow.

Simulation and analytic modeling are the two techniques used to model computer systems. Both are important techniques of computer performance evaluation. They can be used in CPE efforts related to selection evaluation, performance projection, and performance optimization. Simulation and analytic modeling are important because they provide insight into the functioning of the computer system being evaluated. Unlike other techniques of computer performance evaluation, the modeling techniques are not solely used to provide workload characterizations or performance measurements. Rather, they also attempt to provide insight into the cause and effect relationships in a computer system which influence performance. A discussion of simulation will be presented in Chapter IV, and a discussion of analytic modeling will be presented in Chapter V.

Summary

An overview of computer performance evaluation has been

presented in this chapter. This overview included a review of workload characterization, performance measures, performance goals, and CPE tools and techniques. As illustrated in this chapter, the available CPE tools and techniques could be and should be used by AFAL personnel in arriving at a solution to the performance problems of the DECsystem-10 at the AFAL. Although the AFAL has several software monitors (TRACK, SYSTAT, and METER), they do not have any simulation or analytic modeling tools which could be used in performance optimization and performance projection studies. For this reason, the techniques of simulation and analytic modeling will be presented in the following chapters. Following these chapters will be a description of an analytic model of a time-sharing computer system that could be used by the AFAL for computer performance evaluation of the DECsystem-10.

IV. Simulation of Computer Systems

To solve the performance problems of large-scale time-sharing computer systems like the DECsystem-10 usually requires CPE efforts related to performance optimization and performance prediction as well as performance monitoring. Successful performance optimization (the optimization of the performance of a computer system for a given workload and hardware configuration) and reliable performance projection (the prediction of performance changes due to hardware configuration, operating system, or workload changes) both require a knowledge of the cause and effect relationships that exist in the computer system being evaluated. Unfortunately, the complexity of modern computer systems makes it difficult for CPE analysts to understand the behavior of these systems. Modeling can be used as a computer performance evaluation tool by CPE analysts to identify and study the factors and relationships in a computer system that influence performance.

As stated in Chapter III, one of the major approaches to computer system modeling is simulation. Simulation has been defined as "the act of representing some aspects of the real world by numbers or symbols which may be easily manipulated to facilitate their study" (Ref 18:1). Simulation is widely used as an investigative tool to study the behavior of systems as diverse as oil refineries, traffic control patterns, and relations between nations. As used in this thesis, simulation means the act of representing a computer system by a symbolic

model that can simulate the behavior of a specified computer system for a specific workload and collect the necessary data required for performance analysis. Unlike analytic modeling, the other main approach to computer system modeling, simulation is not restricted to the use of models of computer systems that are based on closed form mathematical expressions of system behavior. Therefore, simulations may include any level of detail necessary to reproduce system behavior realistically since the system simplifications that are required in analytic modeling are not necessary in simulation.

Computer System Model

In simulation, a computer system is modeled as a collection of related entities characterized by attributes (Ref 17:4). Entities are objects. They may be jobs, tasks, requests for system resources, peripheral devices, peripheral device controllers, CPUs, etc. Entities which represent system resources (CPUs, memory units, etc.) are also sometimes referred to as facilities. Each entity in the model is described by its attributes. The attributes of a job entity might include its priority, memory requirement, processing time requirement, etc. The attributes of a disk unit entity might include its storage capacity, its rotational speed, its average seek time, etc.

In addition to their attributes, entities are characterized by sets (Ref 14:146). Sets are groups of entities which have at least one common characteristic. For example, an entity

representing a disk controller could be a number of a set of entities representing all disk controllers in the system. An entity representing a job could be a member of a set of entities representing jobs requesting main memory.

In addition to belonging to sets, entities may also own sets. Set ownership implies control of the entities in the set. For example, the entity representing the disk controller could be the owner of a set containing entities representing all disk units attached to the disk controller. Likewise, an entity representing a processor could own a set composed of entities representing jobs which are waiting to be executed. In this case, the set would be equivalent to a CPU queue in the actual computer system. The DECsystem-10 run queues, PQ1 and PQ2, could thus be realistically represented as sets in a simulation of the DECsystem-10.

Model States

The collection of entities, the values of their attributes, and their membership in sets is a static model of a computer system. This entity structure models the "current" status of the workload, the hardware components, and the queueing structure of a computer system during simulation. In simulation, the state of a computer system at an instant in simulated time is defined by the entity structure. The state changes of a computer system are modeled by the changes which occur to the entity structure over simulated time. In addition, since the behavior of a computer system is determined by its state changes

over elapsed time, the behavior of a computer system is also modeled by the changes which occur to the entity structure over simulated time.

Changes which occur to the entity structure during simulation include changes in the number and type of active entities, the value of entity attributes, and the set assignment of entities (Ref 14:146-147). Entities which represent the workload are usually created and destroyed to model the initiation and subsequent completion of jobs or tasks. Although entities which represent system resources (facilities) are established at the start of simulation, they may also be destroyed. For example, an entity which represents a disk unit might be destroyed to model its hardware failure. The value of the attributes associated with each entity are changed to model computer system activity. If a job entity has a processing-received attribute, then the value of this attribute would be increased to indicate any processing received by the job entity.

Sets which model the hardware configuration of the computer system are not usually changed during simulation. However, sets associated with job or task entities are dynamic. Job or task entities are included in sets or deleted from sets to model their flow through the computer system. For example, a job may be associated with a set of all jobs needing core. However, once a job receives core, it is removed from the set since its characteristics no longer agree with the characteristics of the other members of the set. In a simulation of the

DECsystem-10, the addition of job entities to sets and the deletion of job entities from sets could be used to represent the transfer of jobs among the various system queues (run queue, teletype I/O queue, disk I/O queue, etc.).

State Control

Although the entity structure represents the state of the computer system model at any instant in simulated time, another structure is required in simulation to control the state changes of the model. Whereas the entity structure models the workload, hardware configuration, and queueing structure of a computer system, this additional structure models the resource allocation algorithms associated with the operating system of the computer system. This structure is composed of two sets of rules. The first set of rules describes the mathematical relationships that exist between the attributes associated with entities (Ref 17:23). For example, a mathematical relationship exists between the memory-allocated attribute of a job entity and the memory-available attribute of the memory unit entity. The modeling of core allocation requires that the memory-allocated attribute of the job entity be increased and the memory-available attribute of the memory unit entity be decreased by the amount of core allocated to the job entity. This mathematical relationship thus controls the state change associated with core allocation.

The second set of rules that controls state change in

the computer system model defines the logical relationships that are used to determine the actions simulated in the model (Ref 17:24). The logical relationships define the actions to be taken if a condition exists and the alternative actions to be taken if a condition does not exist. The logical relationships actually control the flow of jobs or tasks through the computer system model. For example, if a CPU becomes idle, the next action associated with the CPU is either the execution of another task or the initiation of a wait state for the CPU. The condition which controls these two actions is the number of entities (tasks) grouped in a set reserved for entities (tasks) that require processing time. If no tasks require processing, the CPU is placed in the wait state. Otherwise, one of the jobs waiting for processing time is selected for execution, and the CPU is placed in an execution state. Logical relationships also define the order in which entities are placed in and removed from sets. Examples of these logical relationships or task selection rules are first come first serve, last come first serve, longest in queue first serve, etc.

Discrete Event Simulation

Changes in the state of a system may occur continuously over time or at discrete points in time. Since the activities of large-scale computer systems like the DECsystem-10 are controlled by a system clock, state changes in computer systems occur at discrete moments in time. Therefore, in simulation,

computer systems are modeled appropriately by discrete event simulation. Discrete event simulation is the modeling of computer systems in which state changes that occur as a result of changes in the entity structure of the model are represented by a collection of discrete events. Discrete events are the actions that result from the application of the logical relationship rules of the computer system being simulated to the entity structure of the model. Thus, discrete simulation implements the concepts of an entity structure and a mathematical and logical relationship structure to model computer systems.

Queueing Models. Discrete event simulations of computer systems normally model computer systems as queueing systems. A queueing system consists of a network of service centers, tasks which demand service at the service centers, and a set of rules that govern the order in which tasks waiting for service at the service centers are selected for servicing. The validity of such a modeling approach for computer systems is supported by the structure of the operating system of computer systems like the DECsystem-10.

A discrete event simulation of any queueing system, whether the simulation models a workshop, bank, or computer system, must model the tasks to be performed, the service center structure, and the selection for service rules associated with the service centers. For a computer system, a discrete event simulation must therefore model its workload, its hardware configuration, and its resource allocation

algorithms. Although the hardware configuration and resource allocation algorithms of a computer system are fixed and can be modeled precisely, the workload can not be modeled precisely, because it is constantly changing and is unpredictable. Therefore, the workload is modeled probabilistically, or it is modeled from trace information obtained from the actual computer system that is being simulated.

Stochastic Simulation. Discrete event simulations of computer systems that model the workload of a computer system probabilistically are called stochastic simulations. (Ref 9:64). The parameters normally associated with workload characterization (see Table I) are generated as random samples of specified distributions in stochastic simulations. These distributions may be standard mathematical distributions (exponential, uniform, etc.), or they may be distributions established from workload measurements taken directly from the computer system.

Trace-Driven Simulation. Discrete event simulations of computer systems that model the workload of a computer system by a "deterministic sequence of resource demands" are called trace-driven simulations (Ref 9:66). The deterministic sequence of resource demands that is used to characterize the workload of a computer system consists of a time-ordered list of resource demands obtained from a trace of the activities of a computer system. This time-ordered list of resource demands may have been obtained from the normal workload of the computer system or from a benchmark.

Simulation Languages

Obtaining the description of the hardware configuration, the workload, and the resource allocation algorithms of a computer system that is necessary in modeling is extremely difficult because the operating system must be studied and the workload must be measured. However, once this data is gathered, the actual building of a model may be just as difficult. Building a data structure that will support the concepts of an entity structure (integrated data base), controlling and coordinating all events in time-ordered fashion, and collecting data needed for performance evaluation can be costly, time consuming, and discouraging. Fortunately, several special purpose discrete event simulation languages are available which facilitate the actual building of discrete event models. These simulation languages are a major reason for simulating computer systems with discrete event modeling techniques. A review of three of the most popular and most powerful of the discrete event simulation programming languages follows:

GPSS. GPSS is an IBM-developed interpretive simulation programming language. Users of GPSS need little programming experience because the statements of the language are limited but powerful. Each statement of the language corresponds to an assembly language subprogram which is interpreted and executed each time the statement is encountered. The powerful macrostatements of GPSS enable users who are unfamiliar with simulation to use the language. Some versions of GPSS provide

the user with only limited options to perform specialized computations. However, GPSS V allows users to write additional subroutines in PL/I. GPSS also automatically performs data collection, statistical analysis, and printing of formatted summaries.

SIMULA. SIMULA is an ALGOL-based simulation language that is currently available for use on the CDC6600 at Wright-Patterson Air Force Base, Ohio. SIMULA implements simulation concepts that allow a collection of programs, called processes, to conceptually operate in parallel. During execution, new processes may be described and dynamically generated. Data local to one process can be shared by all processes, and a central time routine is available to control the activity flow within the various processes.

SIMULA provides the user with a variety of sampling and data analysis procedures. These procedures allow the user to generate pseudorandom numbers, Bernoulli trials, continuous and discrete uniform deviates, etc. SIMULA also provides procedures for constructing histograms of accumulated data.

SIMSCRIPT II.5. SIMSCRIPT II.5 is another simulation programming language that is available for use on the CDC6600 at Wright-Patterson Air Force Base, Ohio. SIMSCRIPT II.5 is the most comprehensive discrete event simulation programming language currently available, and it is divided into five levels of complexity. The first level is a simple teaching language designed to introduce programming concepts to non-programmers. The second and third levels are comparable to

FORTRAN and ALGOL respectively. The fourth level provides the support for entity-attribute-set capabilities needed to build an entity structure and manage it during simulation. The fifth level contains statements for time advance, event-processing, and generation of statistical variates, random number, etc.

SIMSCRIPT II.5 provides a number of standard routines that can be used to support simulation. SIMSCRIPT II.5 provides a routine which controls the execution of events based on their scheduled time of execution, routines to accumulate data during simulation, and routines to analyze accumulated data and print the results in formatted summaries.

Advantages

Simulation is a very powerful and useful computer performance evaluation tool. Simulation can be especially effective in the areas of computer performance projection and computer performance optimization. Some of the advantages of simulation as a CPE tool are as follows:

1. Simulation provides a framework in which evaluation of a computer system can occur without impacting the operational capability of the computer system.
2. Simulation can be used as a method of investigating the effect that changes in the workload, the hardware configuration, or the operating system have on performance measures.
3. Since only those events that change the state of the computer system must be simulated to obtain valid

performance data, simulation can usually compress a large interval of simulated time into a much smaller interval of real time.

4. A system can be modeled at any level of detail necessary to achieve a realistic model of a computer system. No simplifications must be made to insure mathematical exactness as in analytic modeling.
5. A simulation model can be composed of system component descriptions that represent different levels of detail.
6. A simulation can be designed to provide any performance measures that could be gathered from the real system.

Disadvantages

Although simulation is a very powerful and flexible computer performance evaluation tool, it does have several disadvantages. Some of the disadvantages of simulation as a CPE tool are as follows:

1. The level of detail needed in a simulation is hard to determine. Not enough detail leads to invalid results. Yet, as the detail of a model grows, the time that must be spent in preliminary study of the actual computer system and the time that must be spent in building the model grows at a nonlinear rate.
2. As the detail of a model grows, cause and effect relationships become harder to identify although the goal of simulation is to identify and study these relationships.

3. If a very detailed level of simulation is required, it may be time consuming and even impossible to identify all of the lower level resource allocation algorithms in the operating system.
4. It is difficult to determine how long it takes a given model to reach a stable state. Therefore, it is difficult to determine when to start taking performance measurements.
5. Simulations tend to be system dependent. Simulations usually only apply to a single manufacturer's computer system. Major modifications are also necessary when the hardware configuration or the operating system change.
6. The cost and effectiveness of simulation depends on the available discrete event simulation programming languages and the ability of the CPE analyst to use these languages. The analyst might have to learn a new programming language before he could build a simulation model of his computer system. If simulation languages are not available, the cost and time required to build a model in a language like FORTRAN may be prohibitive.

V. Analytic Modeling of Computer Systems

As stated in Chapter III, modeling is a valuable computer performance evaluation technique, especially when applied to the analysis and solution of computer performance problems through performance optimization and performance projection. The two major approaches to modeling computer systems are simulation, which was reviewed in Chapter IV, and analytic modeling. Analytic modeling is the act of representing a computer system by a mathematical description of its performance behavior (Ref 9:34).

The state of a computer system describes the status of its system resources and its workload at any given time, and state transitions represents the flow of tasks through the computer system over a time interval. The performance of a computer system is usually evaluated by studying the state changes which occur in a computer system. As explained in Chapter IV, simulations model system performance by simulating the state changes which occur in a computer system. Thus, the measures of performance provided by simulations are measurements of reproduced state changes over a simulated time interval.

Solutions for analytic models which describe the time-dependent behavior of computer systems are complex or have not been found yet. Therefore, most analytic models are based on the premise that the probability of a computer system being in a particular state approaches a limit as time approaches infinity. These probability limits describe a computer system at

statistical equilibrium or steady state. The distribution of these probability limits across all possible states constitutes a stationary probability distribution (Ref 19:155). Although the state of a computer system changes with time, the probability that the system is in a given state is therefore assumed not to change in time. Using this concept of steady state, analytic modeling relates the workload, the hardware configuration, and the resource allocation algorithms of a computer system to its steady state performance through mathematical expressions. Both the workload and the steady state performance can be characterized by stationary probability distributions (Ref 9:30). Therefore, analytic models attempt to represent the performance behavior of a computer system accurately; but, unlike simulation, they also retain mathematical tractability.

The limitations of analytic models to mathematically tractable expressions of computer performance behavior requires that many simplifications be made when modeling modern, complex computer systems. Simplification of computer system characteristics is usually necessary in analytic modeling because solutions can not be generated currently for arbitrarily complex models. Because solutions for arbitrarily complex models can't be generated currently, the level of detail required to realistically represent computer systems can not always be achieved through analytic modeling techniques. Therefore, simulation, which can model a computer system at any level of detail necessary to achieve realistic modeling, is the more popular of the two

modeling techniques. "In recent years, however, a number of general models and methods of analysis have been developed which have greatly increased the range of applicability of analytic modeling" (Ref 20:946). Thus, analytic modeling may be a viable alternative to simulation in some computer performance evaluation efforts.

The performance problems associated with the DECsystem-10 owned by the Air Force Avionics Laboratory correlate closely with performance problems that can be analyzed and solved through performance optimization and performance projection CPE efforts. Because modeling plays a major role in performance optimization and performance projection CPE efforts and because analytic modeling is one of the two major approaches to modeling, the remainder of this chapter will consist of a review of analytic modeling. The purpose of this review is to familiarize the reader with the basic concepts of modeling computer systems analytically, and the capabilities and limitations of analytic modeling as a CPE tool. Analytic modeling techniques which could be used to support performance optimization and performance projection studies of the DECsystem-10 will be identified. The review of analytic modeling in this chapter presents the development of analytic modeling as a CPE tool and several models which can be applied to time-sharing computer systems. Past efforts to apply these models to actual computer systems and the advantages and disadvantages of analytic modeling as a CPE tool for performance optimization and performance projection are also presented.

Queueing Theoretic Models

As stated in Chapter IV, one of the most common and most applicable ways to describe a modern computer system (batch or time-sharing) is to represent the system as a queueing system. This representation is realistic because jobs (tasks) do "travel" through a computer system seeking service at the various system resources (service centers). If these resources are busy when a job requests their service, the job must normally wait until the system resource being requested is available. This delay is usaully referred to as a queueing delay, and it can have a significant impact on various performance measures. Therefore, many of the basic designs of modern computer systems are compatible with the traditional concepts of queueing theory.

However, memory is one resource which does not follow the "one request at a time" service rule. Since most modern computers have multiprogramming capabilities, more than one job can occupy main memory simultaneously. In addition, once a job obtains memory, it may request service from other system resources. For example, a job which has been allocated memory may request service from the CPU. Therefore, a single resource (memory) in a computer system can service more than one job (task) concurrently and a job (task) can request and receive service at two or more system resources (service centers) concurrently. Since the DECsystem-10 is a multiprogramming system that allows buffered I/O, a single job can actually be requesting service at three resources (memory, CPU, disk)

concurrently in this system. In spite of these conflicts with the basic concepts of queueing theory, the similarities between queueing theory and the flow of jobs through a computer system have led to the extensive use of queueing theory principles to model computer systems, their individual components, and their activities.

Infinite Population, Single Server Models. Prior to 1970, analytic models that were used to evaluate the performance of computers were almost exclusively single server models (Ref 21:11). Single server models are characterized by their representation of a single service center (system resource) and its queue in isolation. The service center (CPU, disk, etc.) satisfies some resource demand (CPU time, I/O, etc.) of tasks, and the queue holds tasks which are waiting to be serviced. One of the classical single server models found in queueing theory is the infinite population, single server model shown in Fig. 5. Models of this type require the following as inputs:

1. The arrival rate of tasks at the service center
2. The service time distribution for tasks at the service center
3. The scheduling algorithm of the service center

The performance measures obtained from infinite population, single server models are the average response time for tasks and the utilization level for the modeled resource under steady state or equilibrium state conditions. Given the three inputs listed above, analytic equations are used to determine the

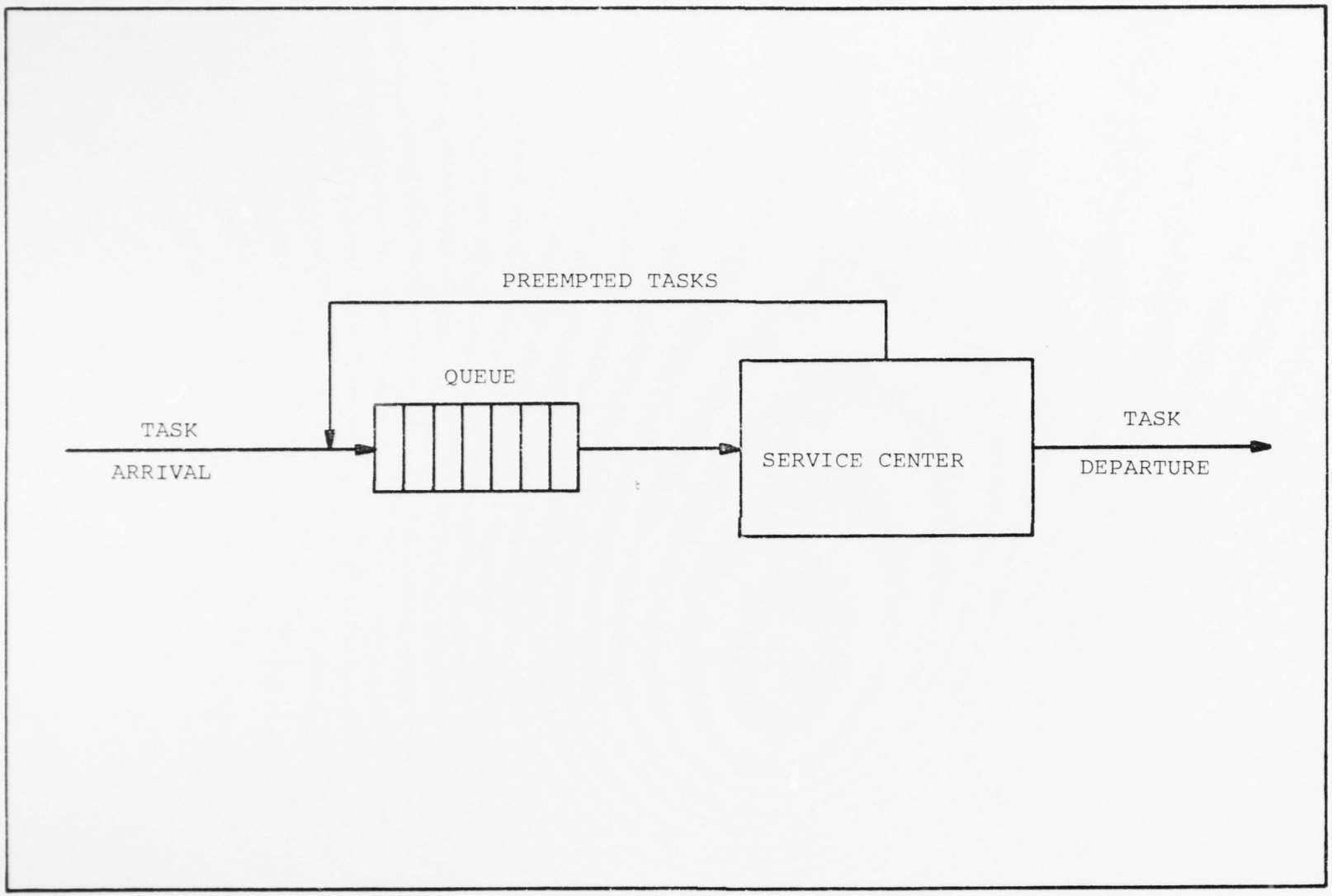


Fig. 5. Infinite Population, Single Server Model (From Ref 21:12)

stationary probability distribution for the number of tasks at the service center. To obtain an analytic solution for this stationary probability distribution under steady state conditions usually requires that certain assumptions concerning the arrival rate of tasks and the service time for tasks be made. The arrival rate of tasks is almost always assumed to be Poisson, and the service time distribution is often assumed to be exponential (Ref 21:13). Once it is determined, the stationary probability distribution for the number of tasks at the service center is used to calculate the performance measures of average response time for tasks and resource utilization. The average response time for a task is its average service wait time (average queueing delay) plus its average service time. For example, if a task must wait for service an average of 20 seconds and requires an average of 5 seconds of service, then its average response time is 25 seconds. Since the service center is active when it is servicing a task, the utilization of the resource that is being modeled is simply the stationary probability that at least one task is at the service center.

The stationary probability distribution for the number of tasks at the service center also defines the steady state of the service center. The steady state of the service center is represented by the average number of tasks at the service center. Since the population that is represented by the model is infinite, the steady state of the service center is not bounded above. Therefore, models of this type are also referred to as

infinite state, single server models.

The infinite population, single server model has been used traditionally to compare the effect of different scheduling algorithms on the response time for tasks at various resources of a computer system under given conditions. For example, CPUs have been modeled to predict the response time for tasks composing a given workload at a CPU under different scheduling algorithms. Seldom have infinite population, single server models been used to predict the performance of real computer systems. The limitation of infinite population, single server models as performance prediction tools is the assumption that the arrival rate of tasks at the service center is Poisson. This assumption implies that the number of active tasks in the system is potentially infinite and that the average arrival rate remains the same no matter how many tasks are queued at the service center. However, this assumption is usually not valid for computer systems.

Because of the Poisson task arrival rate assumption, infinite population, single server models can only be used to model batch systems if the number of users is very large and if there is no restriction on the number of jobs that each user can input into the system. For time-sharing systems, however, the Poisson task arrival rate assumption is not generally valid. The number of active jobs in a time-sharing system tends to be proportional to the number of physical terminals available. When a time-sharing user submits a job (task), he must wait for this job to be completed before he submits a second job.

Therefore, as the queue length increases, the arrival rate of new jobs into a time-sharing computer system must decrease. Although the DECsystem-10 has batch as well as time-sharing processing capability, the maximum number of jobs which can be active in the system at any given time is controlled by a parameter in the operating system. For the DECsystem-10 at the Air Force Avionics Laboratory, this parameter is currently set at 40. Therefore, the Poisson task arrival rate is not applicable to the DECsystem-10 at the Air Force Avionics Laboratory.

Finite Population, Single Server Models. Finite population, single server models are better representations of time-sharing systems and time-sharing system components than infinite population, single server models because the number of active tasks in such models is fixed. A finite population, single server model is shown in Fig. 6. A finite population, single server model represents a finite, fixed number of tasks interacting with a service center. Tasks are not allowed to arrive at the service center from outside the system; and, rather than leave the system after servicing as they do in infinite population, single server models, tasks are recycled back to the service center after a specified delay. Models of this type require the following as input:

1. The number of active tasks in the system
2. The service time distribution of tasks at the service center
3. The scheduling algorithm of the service center

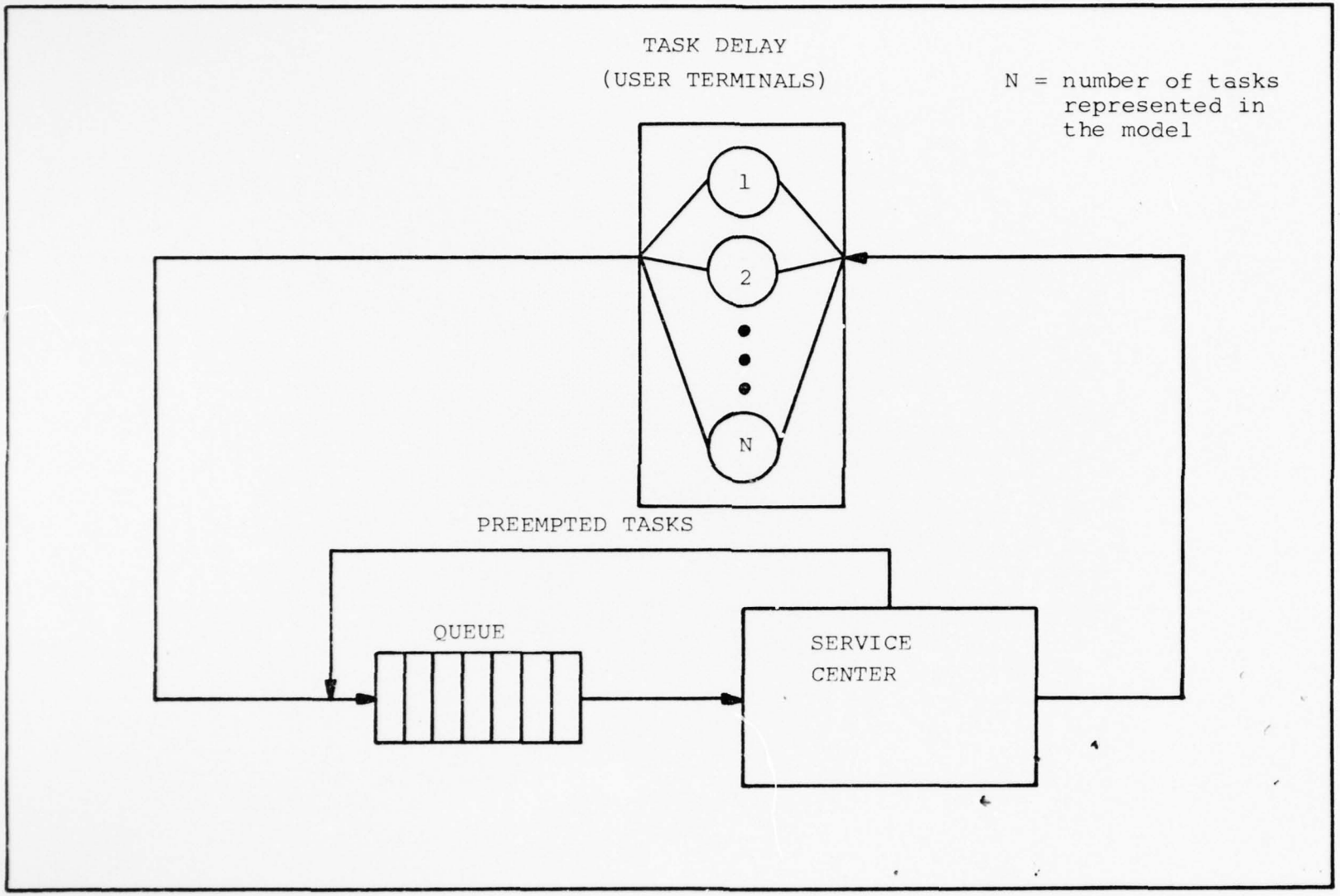


Fig. 6. Finite Population, Single Server model (From 21:15)

4. The delay time of tasks between the time that they leave the service center and the time that they arrive back at the service center

The performance measures and the methods used to obtain these performance measures for finite population, single server models are the same as those previously described for infinite population, single server models. However, the arrival rate of tasks at the service center is not restricted to a Poisson arrival rate. The service time distribution is still often assumed to be exponential. The steady state of the service center is also still represented by the average number of tasks at the service center. Since the number of tasks that are represented in the model is fixed, the state space (all possible states) of the service center is bounded. For example, if there are five tasks represented in the model, then the service center can be in one of six different states. These six states represent the possible number of tasks at the service center (0-5) and compose the state space. Thus, finite population, single server models are also referred to as finite state, single server models.

A finite population, single server model is easily applied to a time-sharing computer system. The tasks which flow through the model represent requests for service issued by users at terminals. The service center represents the computer; and, the average response time for tasks at the service center represents the time delay between the time that a user issues a service request and the time that he receives a response

to this request. Thus, the response time for tasks at the service center equates to the computer system response time for the users at their terminals. After tasks are serviced, the delay which they encounter before being requeued for additional service represents the think time of users at their terminals between the time that they receive a response to their request and the time that they issue another request.

Although finite population, single server models are better representations of time-sharing systems than infinite population, single server models, finite population, single server models are limited as computer performance evaluation tools. Since they contain only one service center, finite population, single server models must represent an entire computer system as a single resource. The use of finite population, single server models in performance optimization and performance projection CPE efforts is therefore severely restricted because they can not model multiple system resources and their interrelationships.

Closed Queueing Network Models. A third type of queueing theoretic model which can be used to represent computer systems analytically is the closed queueing network model. Closed queueing network models are similar to finite population, single server models except for one major difference. Closed queueing network models can be used to represent queueing systems with an arbitrarily large, but finite, number of service centers. Therefore, closed queueing network models are more realistic representations of computer systems than single server models because

they can model the system as a collection of interacting, yet distinguishable, service centers. A closed queueing network model of a hypothetical time-sharing computer system is shown in Fig. 7.

A queueing network model consists of a number of nodes and a linkage structure among the nodes. Each node represents a service center and its associated queue. The linkage structure represents the paths which tasks may take as they move among the nodes. The queueing network is also closed if the number of tasks in the network is fixed. New tasks can not arrive from outside the network, and tasks can not exit from the network.

The information which closed queueing network models require as input is as follows (Ref 22:39):

1. The number and characteristics of the tasks in the network
2. The rules governing the service time for tasks at each node
3. The rules governing the scheduling algorithms at the service centers
4. The rules governing the transition of tasks between nodes

The service centers which are represented by the nodes in closed queueing network models are characterized by their queueing discipline and the number of servers which they contain. The queueing discipline is synonymous with the scheduling

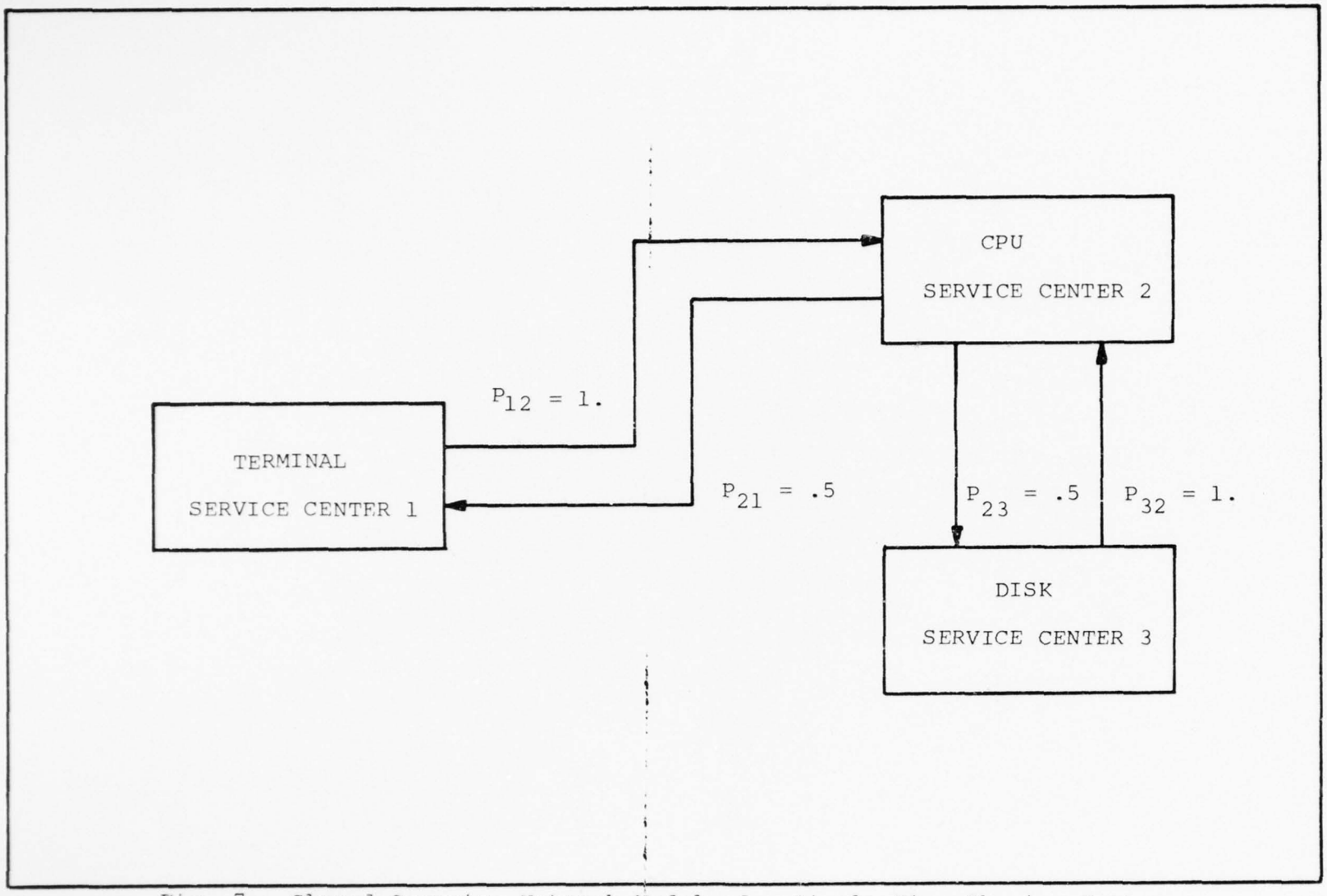


Fig. 7. Closed Queueing Network Model of a Simple Time-Sharing System

algorithm of a service center. It determines the position of tasks in a service center queue and the order in which tasks are serviced. Several examples of queueing disciplines are first come first serve, last come first serve, and round robin.

The servers of each service center actually satisfy the service demands of the tasks, and the number of servers in each service center can vary. The only restrictions on servers are that each service center must contain at least one server and that all servers in a service center must have the same queueing discipline. The service centers are classified as being single server, multiple server, or infinite server service centers. These three types of service centers are shown in Fig. 8. Single server service centers contain only one server. Multiple server service centers contain more than one server but not as many servers as there are tasks in the model. Infinite server service centers contain at least as many servers as there are tasks in the model. Such a service center is referred to as an infinite server service center because there can never be any queueing delay at its node in a closed queueing network model.

The number of tasks in a closed queueing network model is fixed. These tasks may be grouped into different task types based on their resource demand characteristics. Each task type may be further divided into several classes. Tasks within a task type can transition into and out of the classes associated with their task type as they move among the service centers.

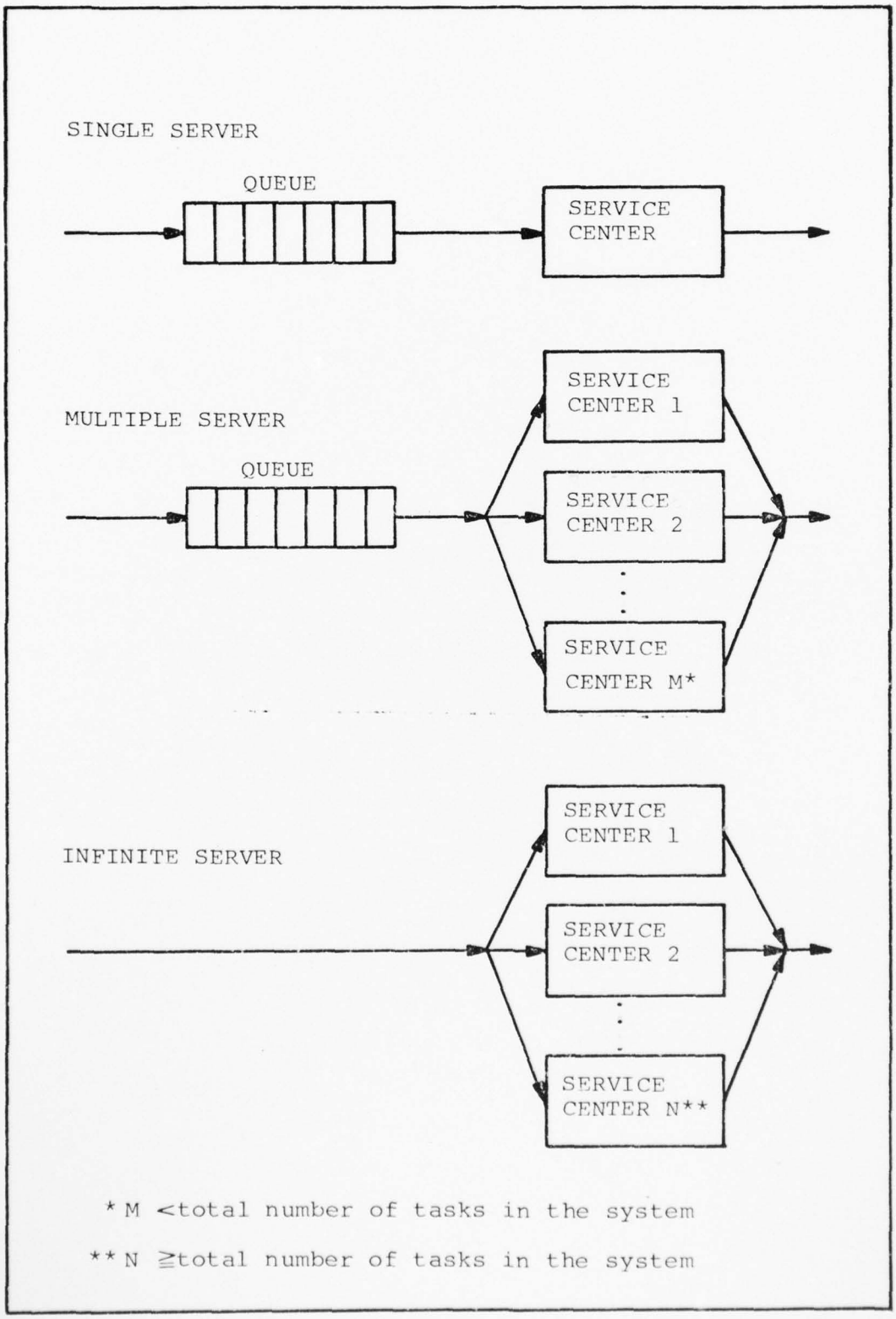


Fig. 8. Three Types of Service Centers

The movement of tasks among the service centers (nodes) of the network model is controlled by a matrix of transition probabilities $P = \left[p_{i,r;j,s} \right]$ where $p_{i,r;j,s}$ is the probability that a task in class r, leaving service center i, will move to service center j and change from class r to class s. This transition matrix defines the linkage structure of the network model. Since the network model is closed, the following is true:

$$\sum_{\text{All } (j,s)} p_{i,r;j,s} = 1 \quad \text{for all } (i,r) \qquad (1)$$

"The transition matrix $P = \left[p_{i,r;j,s} \right]$ defines a Markov chain with states identified by the pair (i,r), and this Markov chain is assumed to be decomposable into $m \geq 1$ ergodic subchains" (Ref 21:23). Each ergodic subchain corresponds to a type of tasks. If task types are not allowed to change class

$$p_{i,r;j,s} = 0 \quad \text{for } r \neq s \qquad (2)$$

If there is only one type of task represented in the network, the notation for the transition matrix reduces to the matrix $P = \left[p_{ij} \right]$. In this matrix, p_{ij} is the probability that a task leaving service center i will move to service center j. The transition probabilities shown in Fig. 7 are examples of the reduced transition matrix described here.

The performance measures and the methods used to obtain these performances measures for closed queueing network models are similar to those previously described for single server models. Similar to single server models, the performance

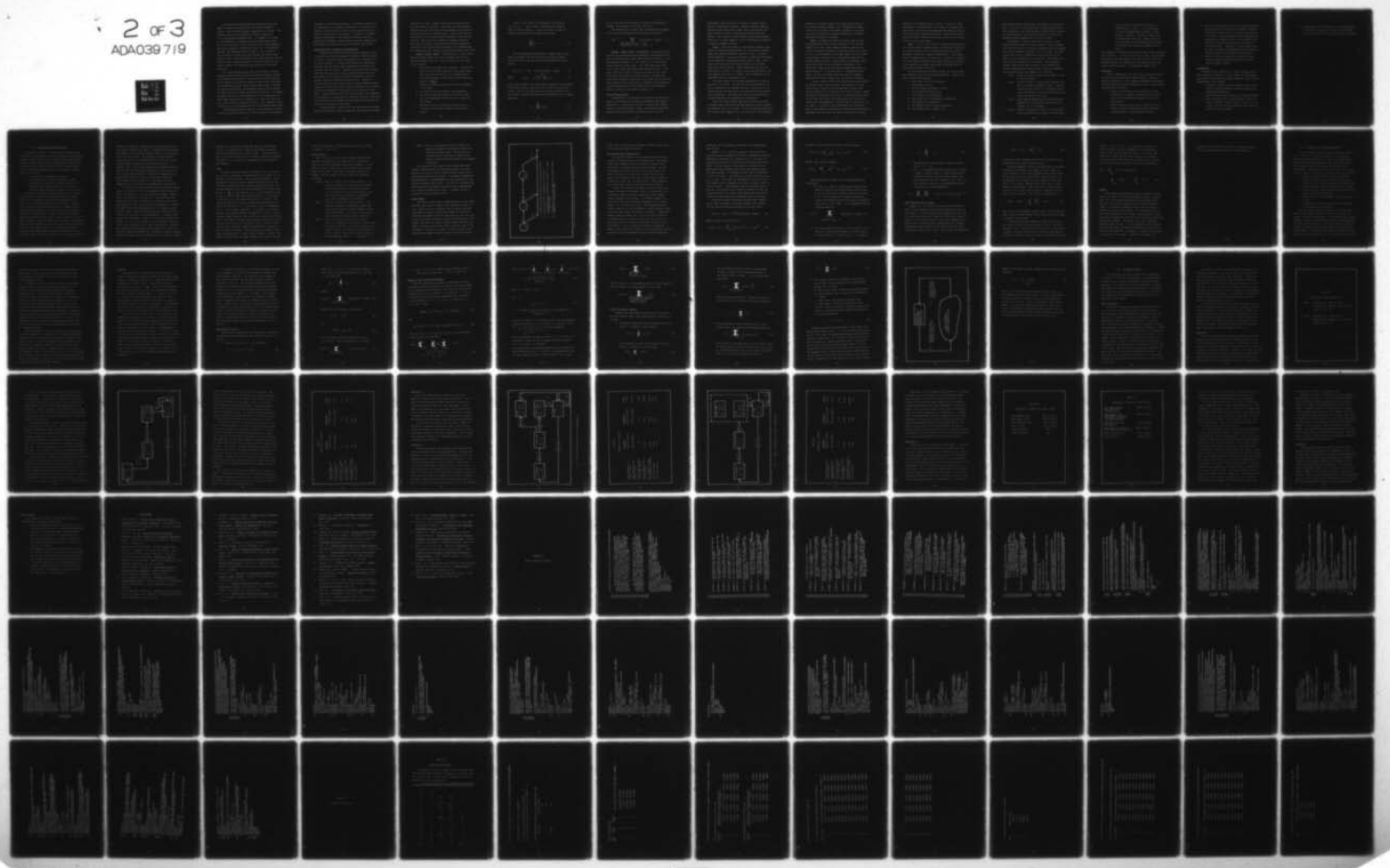
measures which can be obtained from closed queueing network models are calculated from the steady state or equilibrium state probability distribution of tasks among the service centers of the network model. The state of the network model is simply a vector describing the state of each service center in the network model. The state of each service center is defined as the number of tasks at the service center. If several types and/or classes of tasks are modeled in the network, the state of each service center is a vector describing the number of each type and/or class of tasks at the service center.

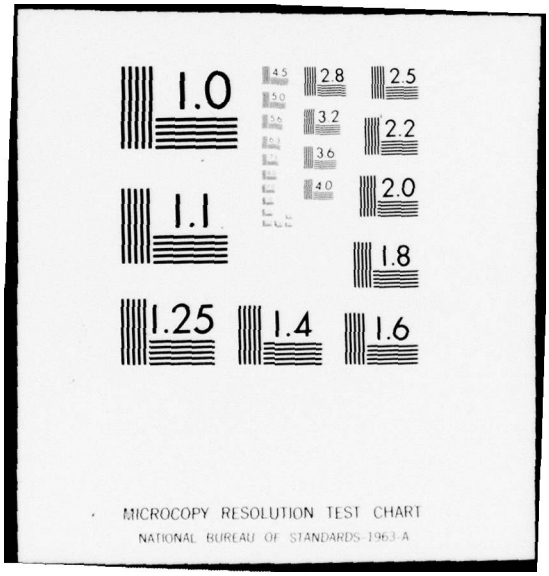
From the equilibrium state probability distribution of the network model, the average queue length at each service center, the average response time for different task types at each service center, and the utilization of each service center can be calculated. It is important to note that when time-sharing computer systems are being modeled, the response time for a task at any one service center can not be equated to system response time as it was for the finite population, single server model. However, if one of the service centers in the network model represents a time-sharing user (terminal), average system response time for this user can be calculated using the transition matrix and the equilibrium state probability distribution. In this case, average system response time is the average amount of time that it takes a task to move through the network (based on its transition probability) and arrive back at the user's service center once the task leaves the user's service center.

AD-A039 719 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
QUEUEING NETWORK MODEL FOR PERFORMANCE EVALUATION OF THE DECSYS--ETC(U)
MAR 77 L E MCKENZIE

UNCLASSIFIED AFIT/GCS/EE/77-1 NL

2 OF 3
ADA039719




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Closed queueing network modeling techniques have great appeal as modeling techniques for time-sharing computer systems. The individual resource components of a time-sharing computer system can be modeled as different service centers in a closed queueing network model. For example, in a model of the DECsystem-10, different service centers might represent the active users at their terminals, the master processor, the slave processor, the RP04 disk units, etc. The programs, commands, and requests issued by users at their terminals can be modeled as tasks which move through the network model based on their transition probabilities, being queued at various service centers (system resources), obtaining service at the service centers, and eventually arriving back at the users' terminals.

The various design options that are available in closed queueing network modeling allows time-sharing computer systems to be modeled realistically. Active users at their terminals can be modeled as an infinite server service center, processors can be modeled as single server service centers with a variety of queueing disciplines, and peripheral devices can be modeled as single server or multiple server service centers. For example, the RP04 disk units of the DECsystem-10 can be represented as four single server service centers or as a multiple server service center with four identical servers. Thus, different levels of detail can be incorporated into the time-sharing system model.

Closed queueing network modeling also increases the effectiveness of analytic modeling as a CPE technique for performance

evaluation of time-sharing systems. Time-sharing systems can be represented as a network of interacting, yet distinct, resources, and the cause and effect relationships which exist among the resources can be analyzed. The effect of workload, hardware configuration, and resource allocation algorithm changes on performance measures such as average response time and resource utilization can also be studied (Ref 20:946).

Equilibrium State Probability Distributions

Although closed queueing network modeling appears to be a versatile and flexible approach to modeling computer systems, it has two limitations that have prevented its widespread use as a CPE technique. To obtain exact results for performance measures, the equilibrium state probability distribution for the model must be derived. Analytic solutions do not currently exist for the equilibrium state probability distribution of many models with interesting queueing disciplines, state dependent transition probability matrices, etc. Therefore, simplifications usually must be made in closed queueing network modeling to insure that current analytic solutions for the equilibrium state probability distribution can be applied to the model. Even when such simplifications are valid, the complexity of the analytic solutions and the resources required to calculate the equilibrium state probability distribution may prohibit the use of this technique in many cases.

Although analytic solutions for the equilibrium state probability distribution of many interesting closed queueing network

models do not exist, analytic solutions have been derived for some models of interest. Two models for which analytic solutions for the equilibrium state probability distribution have been derived are presented here. These models are simplifications of the general model described in the previous section. Both reduce the complexity normally associated with exact solutions for the equilibrium state probability distribution by characterizing the model as a representation of a continuous time, discrete state Markov process (Ref 22:35).

Jackson, Gordon, Newell. The analysis of a basic closed queueing network model was given by Jackson (Ref 23) and Gordon and Newell (Ref 24). The limitations of this model are as follows:

1. All tasks are assumed to be identical. They all belong to the same task type and do not change class. They have the same service time distributions at the various service centers and the same transition matrix $P = \left[p_{ij}\right]$.
2. There are M service centers and N tasks represented in the model.
3. All service time distributions are exponential.
4. The service time distribution at a service center may not be dependent on the state of a second service center.
5. The service rate at each service center may be a function of the number of tasks at the service center.

States in this model are represented by the vector $(n_1, n_2, n_3, \ldots, n_M)$. Each n_i represents the number of tasks at service center i. Since the number of tasks is fixed, valid states of the model must satisfy:

$$\sum_{i=1}^{M} n_i = N \tag{3}$$

For the assumptions listed above, Jackson (Ref 23) and Gordon and Newell (Ref 24) derived the following product form solution for the equilibrium state probability distribution:

$$P(n_1, n_2, \ldots, n_M) = C(N) F_1(n_1) F_2(n_2) \ldots F_M(n_M) \tag{4}$$

where,

$$F_i(n_i) = e_i^{n_i} \Big/ \prod_{j=i}^{n_i} u_i(j) \tag{5}$$

Each $u_i(j)$ represents the instantaneous departure rate of tasks from service center i when there are j tasks in this service center. The e_i's are the mean arrival rate of tasks at service center i and are determined by the following set of linear equations.

$$e_i = \sum_{j=1}^{M} e_j p_{ji} \tag{6}$$

The e_i's can only be determined to within a multiplicative constant, any solution can be used in Eq (4).

The normalization constant C(N) is determined by summing all of the state probabilities and equating the sum to 1.

$$C(N)^{-1} = \sum_{\substack{\text{All states } (n_1, n_2, \ldots, n_M) \\ \text{such that } n_1 + n_2 + \ldots + n_M = N}} F_1(n_1) F_2(n_2) \ldots F_M(n_M) \qquad (7)$$

Baskett, Chandy, Muntz, and Palacios. The analytic solution for the equilibrium state probability distribution of the basic closed queueing network model derived by Jackson, et. al. was extended by Baskett, et. al. (Ref 25) to a more general closed queueing network model. This model eliminates many of the restrictions associated with the earlier model. Different types and classes of tasks are allowed in this model. First-come-first-served, processor sharing, and last-come-first-served queueing disciplines are allowed, and service time distributions are not restricted to exponential distributions. A more detailed review of this model and its corresponding analytic solution is presented in Chapter VI.

Previous Applications

Traditionally, the application of analytic modeling techniques to the modeling of computer systems has been largely theoretical. In theoretical studies, no particular computer system is actually modeled. Rather, these studies deal with general models that attempt to provide insight into the

performance characteristics of a number of systems having similar architectural features. However, analytic modeling techniques have been used to model entire time-sharing computer systems in several computer performance evaluation efforts. The results of these studies indicate that analytic modeling is a viable approach to use in performance evaluation of specific computer systems.

Scherr. The first effort to use analytic modeling techniques to evaluate the performance of a time-sharing computer system was reported by Scherr (Ref 26) in 1967. He used a finite population, single server model to study the performance of the Compatible Time-Sharing System (CTSS) at MIT. Such a simple model was realistic because CTSS was an extremely simple time-sharing system. CTSS allowed only one job into main memory at a time. To achieve a time-sharing effect, jobs were swapped into and out of memory, and the swapping was not overlapped with the execution of jobs.

The single server in this model represented the central processor of the CTSS. The service time associated with the server represented the swapping time for a job as well as its execution time, and the service time distribution was assumed to be exponential. The think time for the time-sharing user was also assumed to be exponentially distributed.

Over a two month period, data were collected from CTSS to obtain input values for the model parameters and to measure the response time of the system. Using the measured values for CPU service time, swapping time, and user think time the model

predicted an average response time surprisingly close to the measured average response time. Although this model is not valid for modern time-sharing systems, its use demonstrated the validity of analytic modeling as a computer performance evaluation technique.

Moore. Although the results of Jackson (Ref 23) and Gordon and Newell (Ref 24) were reported in 1963 and 1967 respectively, their closed queueing network model was not used in a computer performance evaluation effort until 1971. In 1971, an investigation of the applicability of this model in performance evaluation of a time-sharing system was reported by Moore (Ref 22). Moore modeled the Michigan Terminal System (MTS) at the University of Michigan. This system was a large-scale time-sharing system running on an IBM 360 Model 67. It had two CPUs, 1.5 megabytes of main memory, two IBM 2301 drums which were used for swapping, three IBM 2314 disk storage units, and over 90 terminals.

In Moore's model of MTS, the CPUs and disks were modeled as multiple server service centers, the drums as a single server service center with state dependent service rates, and the terminals as an infinite server service center. Moore was forced to assume that all users in the system were identical and that the service time distributions at the service centers were all exponential (restrictions of the Jackson, et. al. model). In reality, both assumptions were inaccurate. Moore's measurements of the system revealed that the exponential assumption was very crude for some of the system resources,

especially the swapping drums. However, in spite of these assumptions, the predictions of the model for response time and resource utilization compared favorably with the measured values for these performance measures. Predicted values for these performance measures were usually within ten to fifteen percent of the actual values.

Chien. In 1974, Chien (Ref 27) reported the application of a closed queueing network model in performance evaluation of a dual processor PDP-10 time-sharing system at Digital Equipment Corporation, Maynard, Massachusetts. This system had two swapping drums, six disk units, and over 70 terminals. The closed queueing network model which Chien used was based on the modeling techniques established by Jackson (Ref 23) and Gordon and Newell (Ref 24).

In this performance evaluation study, five sets of data were gathered while the system was operational. Each set of data included the following:

1. The number of jobs in the system
2. The average job size
3. The main memory available to users
4. The average CPU service time
5. The overhead in job switching
6. The average CPU time needed per interaction
7. The average user think time
8. The number of disk I/Os per interaction
9. The average system response time

The closed queueing network model was applied to each set of data. Even with the identical user and exponential service time assumptions, the closed queueing network model predicted an average response time that was within 15% to 20% of the measured average response time in all five cases.

The purpose of Chien's study was to develop a closed queueing network model with state dependent routing probabilities for the study of time-sharing systems which use swapping as a memory management strategy. In such a model, the probability that a user is swapped would be a function of the number of users competing for main memory. Since exact solutions are not available for such models, Chien proposed a method to approximate their solution. His algorithm for solving closed queueing network models with state dependent routing probabilities for swapping is as follows:

Step 1: Assume an initial value for the number of jobs in the think mode.

Step 2: Use this number to calculate the probabilities that a job must be swapped in after it leaves the CPU service center or its terminal service center and before it arrives at the CPU service center for another time slice.

Step 3: Treat these probabilities as fixed transition probabilities and solve the model using the Jackson, et. al. technique.

Step 4: Compare the number of users in the think mode that is predicted by the model with the number

that was used to calculate the transition probabilities in Step 2. If the difference is within a predetermined tolerance, the algorithm is complete. Otherwise, repeat Step 2 through Step 4 using the number of users in the think mode that was predicted by the model to calculate the transition probabilities in Step 2.

Chien applied this algorithm to the five sets of data collected from the PDP-10. In four out of the five cases, his algorithm predicted an average response time that was closer than the average response time predicted by the classical fixed transition probability model to the measured response time.

Advantages

Analytic modeling has not been used to evaluate the performance of many operational computer systems. However, analytic modeling is a viable computer performance evaluation technique which has several advantages as such. Some of these advantages are as follows:

1. Analytic modeling provides a framework in which evaluation of a computer system can occur without impacting the operational capability of the computer system.
2. Analytic modeling often yields insights into the cause and effect relationships between workloads, system parameters, and performance measures.

3. The cost of developing and running computer programs which perform the calculations necessary in analytic modeling is relative low (in comparison with simulation costs). These programs are also independent of any particular computer system. The computer being modeled is completely described by the inputs to the program alone. Therefore, a single program can be used to evaluate the performance of any number of computer systems with different architectures.
4. Analytic models can be used to calculate the performance measures normally associated with time-sharing computer systems.

Disadvantages

Although analytic modeling is a viable computer performance evaluation technique, it has a number of disadvantages as a computer performance evaluation technique. Some of these disadvantages are as follows:

1. Arbitrarily complex computer systems can not be analyzed by analytic modeling techniques without many simplifying assumptions.
2. Exact solutions exist for only a few analytic models.
3. Many of the features of modern computer systems can not be included in analytic models. The ability of a job to use more than one system resource simultaneously is one such feature.

4. Few people have the mathematical and queueing theory background to effectively evaluate the performance of computer systems with analytic modeling techniques.

VI. A Closed Queueing Network Model

As stated in Chapter V, closed queueing network models are realistic representations of the structure of time-sharing computer systems and the flow of tasks through these systems. Although network models can not represent the concurrent use of more than one resource by a single task, they can represent the following characteristics of time-sharing computer systems (Ref 21:18):

1. A collection of independent resources
2. The sequential use of these resources by tasks
3. The concurrent servicing of different tasks by the resources (Note that the concurrent servicing of the same task by different resources is not allowed.)

Although closed queueing network models have an inherent attractiveness as models of time-sharing computer systems, their use in performance optimization and performance projection of time-sharing systems has been limited. The main reason that they are not used more often in computer performance evaluation efforts is the lack of analytic solutions for arbitrarily complex models. Analytic solutions for network models are usually obtained only by making a number of simplifying assumptions about the system being modeled. Unfortunately, these necessary assumptions often contradict the true characteristics of time-sharing systems. For example, the analytic solutions for network models derived by Jackson (Ref 23) and Gordon and Newell (Ref 24) are based on the assumptions that all

tasks in the system are identical and that the service time for tasks at each service center is exponentially distributed. When applied to time-sharing computer systems, these assumptions are seldom valid. The demands for system resources associated with each interaction between a user and a time-sharing computer system are actually quite diverse. In addition, Moore (Ref 22:143) showed that although exponential distributions are reasonable approximations for the service time at some system resources, they may be inappropriate as descriptions of service time at other system resources. Moore showed that the service time distribution for the swapping drum in the MTS was not exponentially distributed.

In an effort to overcome some of these unrealistic assumptions, an extension of the closed queueing network model developed by Jackson, et. al. was reported by Baskett, Chandy, Muntz, and Palacios in 1975 (Ref 25). This model does not assume that all tasks are identical or that the service time distributions at the service centers are exponential. Rather, this model allows the representation of any number of types of tasks (jobs), several different queueing disciplines, and a broad class of service time distributions. A product form solution for this model, similar to the solution described by Jackson, et. al. for their model, is also reported. The flexibility and versatility of this model, along with its product form solution for equilibrium state probabilities, make it an attractive modeling framework for time-sharing computer systems. Because this model offers a framework for more realistic analytic

modeling of time-sharing systems than has been possible previously, its potential as a computer performance evaluation tool for the DECsystem-10 is investigated. A summary description of this general model and its analytic solution as reported by Baskett, et. al. (Ref 25) is presented in the remainder of this chapter.

Tasks

The model which was reported by Baskett, et. al. (Ref 25) can be used to represent an arbitrary but finite number, R, of different classes of tasks. For time-sharing systems, a task represents the system resource demands of one user during one interaction between the user and the time-sharing system. The resource demands of the users are represented by the transition matrix $P = \left[p_{i,r;j,s}\right]$ where $p_{i,r;j,s}$ is the probability that a task in class r, upon leaving service center i, will proceed to service center j and change from class r to class s. The transition matrix thus defines a Markov chain whose states are labeled by the pairs (i,r). This Markov chain is decomposable into m $(m \geq 1)$ ergodic subchains. In a model of time-sharing systems, each ergodic subchain corresponds to a type of request (resource demand sequence) that can be initiated by time-sharing users. One ergodic subchain might represent requests associated with text editing and file maintenance, while a second ergodic subchain might represent requests associated with compilation and execution of programs. Several different classes of tasks may be associated with a single ergodic subchain. However, the number of tasks within an ergodic subchain must remain constant.

A detailed discussion of ergodic Markov chains can be found in Chapter II of Ref 28.

Service Centers

While the tasks of this closed queueing network model represent the demands of time-sharing users, the service centers of the model represent the system resources. An arbitrary but finite number M of service centers can be represented in the model. Four types of resource centers are allowed. The four types of service centers are described as follows:

Type 1: This is a single server service center. Its queueing discipline is first-come-first-serve (FCFS), and all tasks have the same service time distribution at this service center type. The service time distribution is assumed to be a negative exponential distribution.

Type 2: This is a single server service center. Its queueing discipline is processor sharing, and each class of tasks is allowed separate service time distributions. All service time distributions must have rational Laplace transforms.

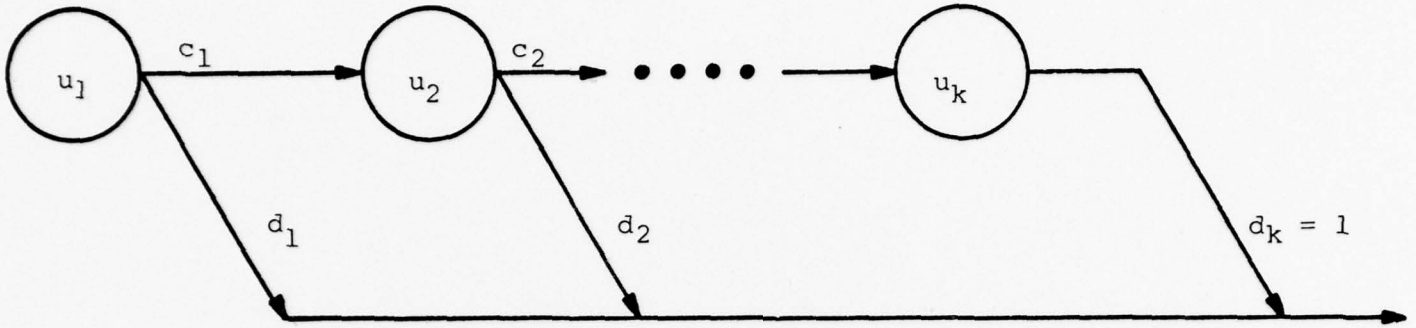
Type 3: This is an infinite server service center. There is no queueing at this service center since the number of servers is at least as great as the number of tasks. The service time distribution of each class of tasks may be unique as long as the distributions have Laplace transforms.

Type 4: This is a single server service center. Its queueing discipline is preemptive-resume last-come-first-served (LCFS). The service time distributions of each class of tasks may be unique as long as the distributions have Laplace transforms.

The restriction that service time distributions have Laplace transforms does not necessarily prevent the representation of service times at various system resources in a time-sharing system realistically. For example, exponential, hyperexponential, and hypoexponential distributions all have Laplace transforms (Ref 25:251). Cox (Ref 29) has shown that distributions having Laplace transforms can be approximated by a network of exponential stages. A diagram of such a network of stages is shown in Fig. 9.

Model States

The state of this model is represented by a vector which describes the status of tasks at the individual service centers. The state of an individual service center describes the number of tasks of the various classes at the service center and the stage of service that they are in or their position in the queue associated with the service center. For example, the state of a type 1 service center describes the number of tasks at the service center and their order in the FCFS queue. The state of a type 2 service center not only describes the number of tasks of each class at the service



k = number of stages of service in the service center

u_j = mean service rate (exponential) at state j

c_j = probability that a task moves to stage j + 1 after it leaves stage j

d_j = probability that a task departs the service center after the j th service stage

Fig. 9. Set of Exponential Stages of Service (From Ref 29)

center but also describes the number of tasks of each class at the various stages of service.

Equilibrium State Probabilities

The analytic solution of this model requires the derivation of its equilibrium state probabilities. The equilibrium state probabilities for the model are based on the solution of the balance equations for the model. The balance equations reflect the fact that at equilibrium, the rate of transition of tasks into a state from all other states must be equal to the rate of transition of tasks out of the given state into all other states. These equations are called global balance equations by Chandy (Ref 30).

Because the states of the model are rather complex and because the number of different states is exceedingly large, calculating equilibrium state probabilities with global balance equations is difficult. An alternate approach to calculating the equilibrium state probabilities is used by Baskett, et. al. (Ref 25:251-254). This approach involves finding solutions for the independent balance equations rather than the global balance equations of the model. Independent balance equations equate the "rate of flow into a state by a customer entering a stage of service to the flow out of that state due to a customer leaving that stage of service" (Ref 25:252). Here, customers and tasks are synonymous. A global balance equation is simply a sum of independent balance equations, and the independent balance equations represent

sufficient but not necessary conditions for global balance (Ref 25:252).

Baskett, et. al. (Ref 25) presents a solution for the equilibrium state probabilities based on independent balancing. Although this solution has a product form, it is really unmanageable because of the detailed state descriptions of the model. They do, however, also present a solution for the equilibrium state probabilities of the model based on aggregate system states. An aggregate system state is defined as a vector $(S_1, S_2, S_3, \ldots, S_M)$ where S_i is another vector $(n_{i1}, n_{i2}, n_{i3}, \ldots, n_{iR})$. Each vector S_i represents the state of a service center and n_{ij} represents the number of class j tasks at service center i. This aggregate state description retains the basic description of the conditions within the model but eliminates much of the detail which makes numerical calculations of the equilibrium state probabilities so difficult.

Using the concept of aggregate system states, Baskett, et.al. (Ref 25) report the following product form solution for the equilibrium state probabilities of their model:

$$P(S_1, S_2, \ldots, S_M) = C(\vec{N}) F_1(S_1) F_2(S_2) \ldots F_M(S_M) \qquad (8)$$

where for type 1 service centers,

$$F_i(S_i) = n_i! \left(\prod_{r=1}^{R} \frac{1}{n_{ir}!} (e_{ir})^{n_{ir}} \right) (1 / u_i)^{n_i} ; \qquad (9)$$

for type 1 service centers and type 4 service centers

$$F_i(S_i) = n_i! \prod_{r=1}^{R} \frac{1}{n_{ir}!} (e_{ir} / u_{ir})^{n_{ir}} ; \qquad (10)$$

and for type 3 service centers,

$$F_i(S_i) = \prod_{r=1}^{R} \frac{1}{n_{ir}!} (e_{ir} / u_{ir})^{n_{ir}} \qquad (11)$$

These equations are based on the following supportive definitions:

1. $\vec{N} = (N_1, N_2, \ldots, N_R)$ is a vector describing the total number of tasks in the closed queuing network model. N_i is the total number of type i tasks in the model.
2. $C(\vec{N})$ is the normalization constant obtained by summing all feasible states of the model and equating their sum to 1 (Ref 21:48). It is represented by the following equation:

$$C(\vec{N})^{-1} = \sum_{\substack{\text{For all} \\ S_1+S_2+S_3+\ldots+S_M = \vec{N}}} F_1(S_1)F_2(S_3)\ldots F_M(S_M) \qquad (12)$$

3. The total number of tasks at a service center is defined as n_i and the total number of tasks of type r at service center i is defined as n_{ir}. Therefore,

$$n_i = \sum_{r=1}^{R} n_{ir} \qquad (13)$$

4. The mean service time of class r tasks at service center i is $1 / u_{ir}$.
5. The e_{ir} 's represent the mean arrival rate of class i tasks at service center r. They can only be determined to within a multiplicative constant and any solution can be used in Eqs (9), (10), and (11). The solution for the e_{ir}'s are determined from the following set of linear equations:

$$e_{ir} = \sum_{j=1}^{M} \sum_{s=1}^{R} e_{js}\, p_{j,s;i,r} \quad \begin{matrix} i = 1,2,..,M \\ r = 1,2,..,R \end{matrix} \qquad (14)$$

State Dependent Service Rates

Baskett, et. al. (Ref 25) also report three different types of state dependent service rates which can be used in their model. The first type of state dependent service rate is an arbitrary but positive function of the total number of tasks at a given service center. Let $f_i(n_i)$ represent the service rate at service center i when there are n_i tasks at the service center relative to the service rate when there is one task at the service center. Then $F_i(S_i)$ is replaced by

$$F_i'(S_i) = F_i(S_i) \quad / \quad \prod_{j=1}^{n_i} f_i(j) \qquad (15)$$

This form of state dependent service rate is useful in representing multiple server service centers.

The state dependent service rates may also be arbitrary but positive functions of the number of class r tasks at service center i. This state dependent service rate allows the service rate for a class r task at service center i to be dependent on the number of class r tasks at the service center. Let $f_{ir}(n_{ir})$ represent the service rate of class r tasks at service center i when there are n_{ir} tasks present relative to the service rate of class r tasks when only one class r task is present. Then $F_i(S_i)$ is replaced by

$$F_i'(S_i) = F_i(S_i) \quad / \quad \prod_{r=1}^{R} \prod_{j=1}^{n_{ir}} f_{ir}(j) \qquad (16)$$

This form of state dependent service rates is not allowed for type 1 service centers because the service rate for all tasks is assumed to be the same.

A third form of state dependent service rates involves the number of tasks at several service centers. This state dependent service rate allows the service rate of tasks within a subset of service centers to be an arbitrary but positive function of the total number of tasks in the subset of service

centers. Let $I = (i_1, i_2, \ldots, i_m)$ represent a subset of m service centers, and let N_I represent the total number of tasks in this subset of service centers. Let $f_I(N_I)$ represent the relative service rate of tasks in the subset I of service centers when there are N_I tasks present relative to the service rates when there is only one task present.

Then $\prod_{i \in I} F_i(S_i)$ is replaced by

$$\prod_{i \in I} F_i(S_i) \Big/ \prod_{j=1}^{N_I} f_I(j) \qquad (17)$$

Summary

Closed queueing network models have an inherent attractiveness as models of time-sharing computer systems. However, their use in performance optimization and performance projection of time-sharing systems has been limited because a number of unrealistic assumptions must be made to obtain a solution for most analytic models. The closed queueing network model presented in this chapter overcomes many of these unrealistic assumptions that have been traditionally associated with analytic models. For example, this model represents different types of tasks. It also represents service centers with different types of queueing disciplines and general service time distributions. In addition, a product form solution is available for this model. Therefore, this model is a

potential CPE tool which can be used in performance optimization and performance projection of the DECsystem-10.

VII. Calculation of Performance Measures

The closed queueing network model presented by Baskett, et. al. (Ref 25) is directly applicable to time-sharing systems like the DECsystem-10. In fact, the analysis which led to this model was motivated by the need for more realistic analytic models of time-sharing computer systems (Ref 25:259). Some features which make it especially attractive as a model of time-sharing systems include the following:

1. Different classes of tasks can be represented.
2. Classes of tasks may have different service time distributions and different transition probabilities.
3. Several different queueing disciplines are allowed.
4. Several service time distributions are allowed (those which have Laplace transforms).
5. Several types of state dependent service rates are allowed.
6. A product form solution exists for the equilibrium state probability distribution.

The four types of service centers are especially applicable to time-sharing systems. FCFS service centers are usually realistic representations of secondary storage devices. Processor sharing and LCFS service centers are good approximations of central processors. LCFS is an efficient preemptive scheduling method and round robin scheduling approaches processor sharing. Processor sharing service centers could thus be used to model the processors of the DECsystem-10 since the

queueing discipline of their main processor queue (PQ2) is round robin. Infinite server service centers can be used to realistically represent the active terminals of a time-sharing computer system.

In addition to being a realistic model of time-sharing computer systems, the model presented by Baskett, et. al. (Ref 25) is a potentially powerful computer performance evaluation tool. The feature which makes it potentially useful as a CPE tool is the product form of its solution for the equilibrium state probability distribution. From the equilibrium state probability distribution, several efficiency and effectiveness performance measures associated with time-sharing computer systems can be calculated. These performance measures include marginal queue length distributions, resource utilization, and mean response time for tasks. The product form of the solution for the equilibrium state probability distribution facilitates implementation of the model as a computer program.

Because this model is an attractive model of time-sharing systems and because of its potential as a computer performance evaluation tool, a program has been written to implement this model so that it can be used in performance evaluation of the DECsystem-10. Appendix A contains a source listing of this program, Appendix B contains a sample output of this program, and Appendix C contains instructions on how to use this program. The remainder of this chapter describes the algorithms which are used in this program to calculate the performance measures associated with time-sharing computer systems.

Approach

The program uses the aggregate system states of the model to calculate performance measures for the system being modeled. As stated in Chapter VI, an aggregate system state is defined as a vector $(S_1, S_2, \ldots, S_M)$ where each S_i is another vector $(n_{i1}, n_{i2}, \ldots, n_{iR})$. Each vector S_i represents the state of a service center and n_{ij} represents the number of class j tasks at service center i. As can be seen from this description of system states, the number of states (state space) increases rapidly as the number of tasks and the classes of tasks increase. For example, a network model with five service centers and two classes of tasks with 10 tasks in each class could have over 4×10^6 possible states.

Eq (8) presents the solution for the equilibrium state probability distribution of the model. To obtain numerical values for the equilibrium state probability distribution of a system being modeled requires that the normalization constant be calculated. Eq (12) presents a straightforward solution for the normalization constant. If Eq (12) is used to calculate the normalization constant and Eq (8) is used to calculate the equilibrium state probability distribution, the number of calculations required increases as the number of states in the network model grows. For even a very simple network model of a time-sharing system, the number of calculations required in such a straightforward approach may be prohibitive.

To overcome this problem, an efficient procedure is used in the program to calculate the normalization constant and the performance measures. This procedure was presented by Muntz and Wong (Ref 31) and was later included in Wong's dissertation (Ref 21). The efficient procedure which they present is an extension of an efficient procedure developed by Buzen (Ref 32) for calculating the normalization constant associated with the Jackson, et. al. model. One limitation of this procedure is that tasks are not allowed to change class. Therefore, the program as implemented does not allow tasks to change class. The present algorithms being used can be expanded to include class changes, but such a change would greatly increase the number of calculations required and the amount of main memory needed to execute the program. The program as implemented also does not allow the third type of state dependent service rates presented by Baskett, et. al. (Ref 25).

Normalization Constant

The efficient procedure for calculating the normalization constant which was presented by Muntz and Wong (Ref 31) is as follows:

1. Define a new vector S_i^* as the following:

$$S_i^* = (n_{i1}^*, n_{i2}^*, \ldots, n_{iR}^*) \tag{18}$$

where n_{ir}^*, r = 1,2,...,R, is the total number of class r tasks in service centers 1,2,...,i-1, and i. This means that

$$n_{ir}^* = \sum_{j=1}^{i} n_{jr} \tag{19}$$

2. Define

$$F_i^*(S_i^*) = \sum_{S_1+S_2+\ldots+S_i=S_i^*} F_1(S_1)F_2(S_2)\ldots F_i(S_i) \tag{20}$$

From Eq (20) the following can be shown:

$$F_1^*(\cdot) = F_1(\cdot) \tag{21}$$

and

$$C(\vec{N})^{-1} = F_M^*(\vec{N}) \tag{22}$$

3. Use the recusive formula relating $F_i^*(S_i^*)$ and $F_{i-1}^*(S_{i-1}^*)$. This relationship is as follows:

$$F_i^*(S_i^*) = \sum_{S_{i-1}^*+S_i=S_i^*} F_{i-1}^*(S_{i-1}^*)F_i(S_i) \tag{23}$$

4. For $i = 2,3,\ldots,M$, compute $F_i^*(S_i^*)$ from $F_{i-1}^*(S_{i-1}^*)$ and $F_i(S_i)$ using Eq (23). $C(\vec{N})^{-1} = F_M^*(\vec{N})$

Marginal Queue Length Distributions

Once the normalization constant is determined, the marginal queue length distributions of tasks at the various nodes in the network can be calculated. Let $P_i(S_i)$ be the marginal probability that service center i is in state S_i, $i = 1,2,\ldots,M$, and $P_i^*(S_i^*)$ be the marginal probability that the subset of service centers $1,2,\ldots,i-1$, and i is in state S_i^*, $i = 1,2,\ldots,M-1$. Then

$$P_M(S_M) = C(\vec{N})\, F_{M-1}^*(\vec{N} - S_M) F_M(S_M) \tag{24}$$

and

$$P_{M-1}^*(S_{M-1}^*) = C(\vec{N}) F_{M-1}^*(S_{M-1}^*) F_M(\vec{N} - S_{M-1}^*) \tag{25}$$

For $i=M-1, M-2,\ldots 2$, $P_i(S_i)$ and $P_{i-1}^*(S_{i-1}^*)$ are calculated recursively by the following:

$$P_i(S_i) = \sum_{k_1=n_{i1}}^{N_1} \sum_{k_2=n_{i2}}^{N_2} \cdots \sum_{k_R=n_{iR}}^{N_R} P_i^*(\vec{k}) \cdot \frac{F_{i-1}^*(\vec{k} - S_i) F_i(S_i)}{F_i^*(\vec{k})} \tag{26}$$

$$\text{and } P_{i-1}^*(S_{i-1}^*) = \sum_{k_1=n^*_{i-1,1}}^{N_1} \sum_{k_2=n^*_{i-1,2}}^{N_2} \cdots \sum_{k=n^*_{i-1,R}}^{N_R} P_i^*(\vec{k}) \cdot \frac{F_{i-1}^*(S_{i-1}^*) F_i(\vec{k} - S_{i-1}^*)}{F_i^*(\vec{k})} \tag{27}$$

where $\vec{k} = (k_1, k_2, \ldots, k_R)$.

For $i = 1$,

$$P_1(\cdot) = P_1^*(\cdot) \tag{28}$$

In Eq (26), $\frac{F_{i-1}^*(\vec{k} - S_i) F_i(S_i)}{F_i^*(\vec{k})}$ is the probability that service center i is in state S_i given that the subnetwork of service centers 1,2,...,i-1, and i is in state $\vec{k}$. In Eq (27), $\frac{F_{i-1}^*(S_{i-1}^*) F_i(\vec{k} - S_{i-1}^*)}{F_i^*(\vec{k})}$ is the probability that the subnetwork of service centers 1,2,...,i-2 and i-1 is in state S_i^* given that the subnetwork of service centers 1,2,...,i-1, and i is in state $\vec{k}$.

The marginal queue length distributions can be calculated directly from the $P_i(S_i)$'s. If $P_i(n_i)$ is the probability that the total number of tasks in service center i is n_i, then

$$P_i(n_i) = \sum_{\substack{\text{all states } S_i \\ \text{such that} \\ n_{i1}+n_{i2}+\ldots+n_{iR}=n_i}} P_i(S_i) \qquad (29)$$

and, if $P_{ir}(n_{ir})$ is the probability that the total number of tasks of class r in service center i is n_{ir}, then

$$P_{ir}(n_{ir}) = \sum_{\substack{\text{all states } S_i \text{ such that} \\ \text{the number of class r} \\ \text{tasks in service} \\ \text{center i is } n_{ir}}} P_i(S_i) \qquad (30)$$

Other Performance Measures

Several other performance measures can be calculated from the marginal queue length distributions. These are as follows:

1. The expected number of class r customers in service center i is defined as the following:

$$E(n_{ir}) = \sum_{j=1}^{N_r} j\, P_{ir}(j) \qquad (31)$$

and the expected number of total tasks in service center i is defined as the following:

$$E(n_i) = \sum_{r=1}^{R} E(n_{ir}) \qquad (32)$$

2. The utilization of service center i by the tasks of type r is given by the following equation if the service center is type 1, 2, or 4 and has only one server:

$$U_{ir} = \sum_{\text{All states } S_i} P_i(S_i) \frac{n_{ir}}{n_i} \tag{33}$$

where n_i is given by Eq (13). Total utilization of service center i is given by the following equation:

$$U_i = \sum_{r=1}^{R} U_{ir} \tag{34}$$

3. The mean departure rate of tasks of class r from service center i is given by the following equation:

$$\Lambda_{ir} = \sum_{\text{all states } S_i} P_i(S_i)\ u_{ir}(S_i) \tag{35}$$

where $u_{ir}(S_i)$ is the mean service rate of class r tasks at service center i when the service center is in state S_i. The mean departure rate of all tasks from service center i is given by the following equation:

$$\Lambda_i = \sum_{R=1}^{R} \Lambda_{ir} \qquad (36)$$

4. The response time can be calculated using Little's result (Ref 33) which states that for any queueing system, $\lambda T = N$, where,

   λ = mean arrival rate of tasks at the queueing system

   λ = mean departure rate of tasks from the queueing system

   T = mean time in the queueing system by tasks

   N = mean number of tasks in the queueing system

   Little's result can be used to find the average time spent in a queueing system by changing the equation to read the following:

$$T = N \ / \ \lambda \qquad (37)$$

Consider the queueing network model of the simple time-sharing model shown in Fig. 7. If the infinite server service center in this model is isolated from the rest of the service centers as shown in Fig. 10, the rest of the service centers can be treated as a separate queueing system. The arrival rate of tasks into this queueing system equates to the departure rate of tasks from the infinite server service center. The departure rate of tasks from the queueing system equates to the arrival rate of tasks at the infinite server service center. Applying Eq (37) the amount of time a task of type r

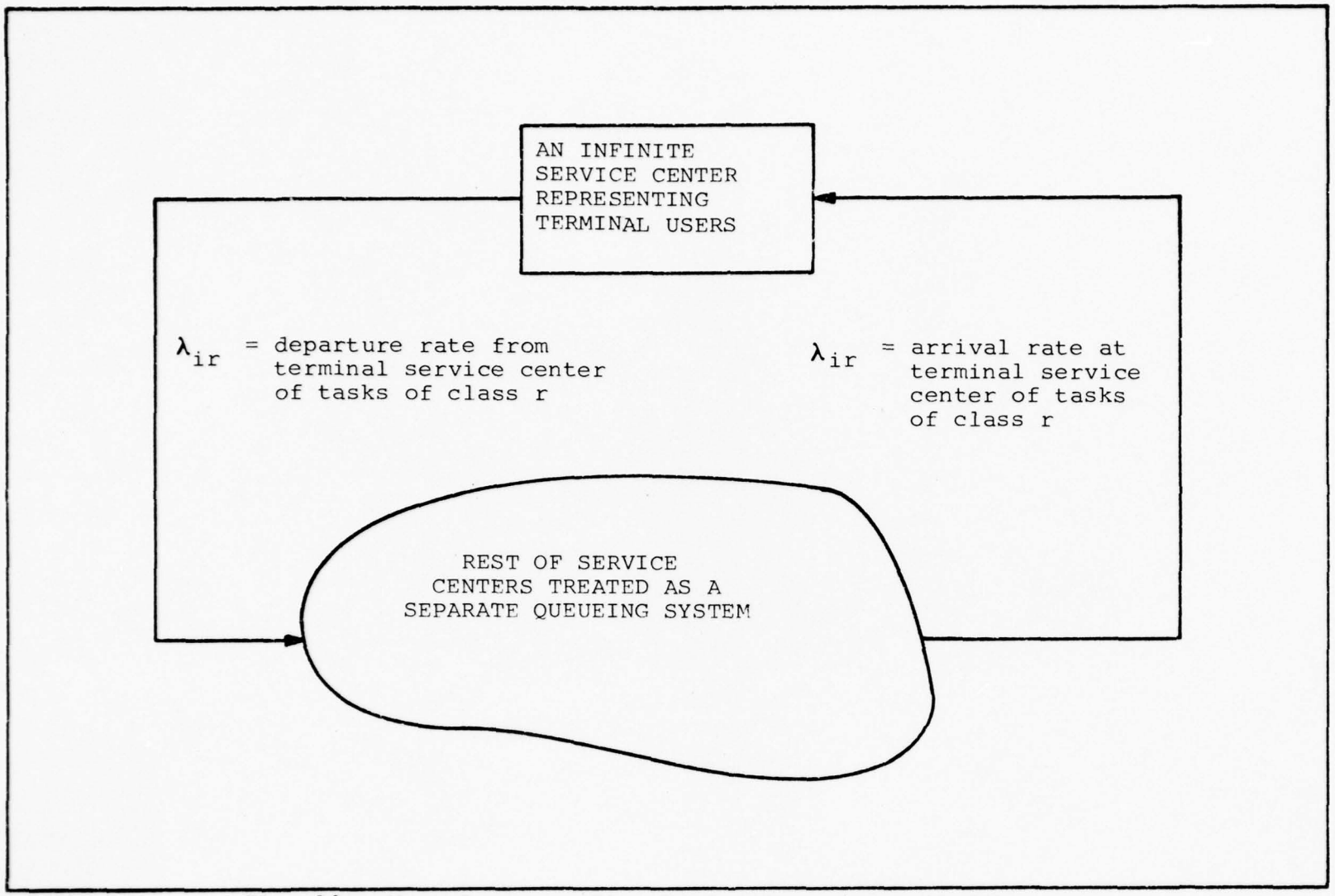


Fig. 10. Mean Response Time Derivation (From 21:59)

spends in the queueing system (response time) is given by the following:

$$T_{ir} = \frac{N_r - E(n_{ir})}{\Lambda_{ir}} \tag{38}$$

where, i is the infinite server service center, and T_{ir} is the response time for class r tasks. N_r is the total number of class r tasks in the model, and Λ_{ir} is the departure/arrival rate of class r tasks at the infinite server service center. T_{ir} actually represents the amount of time that a class r task spends in the queueing system between the time that it departs from the infinite server service center until it arrives back at the infinite server service center.

VIII. Validation Results

The closed queueing network model which is described in Chapters VI and VII was implemented as a FORTRAN program so that it could be used in performance evaluation of the DECsystem-10 at the Air Force Avionics Laboratory. Several experiments were conducted on the DECsystem-10 to determine the validity of values predicted by the model. The validation experiments and the results of these experiments are described in this chapter.

Basic Experiment

All of the validation experiments that were conducted at the Avionics Laboratory followed a similar procedure. The performance of the DECsystem-10 was measured during the execution of a synthetic workload (see Chapter III) which represented a number of time-sharing users. The program TRACK, which was described in Chapter II, was used to measure the performance of the system during these experiments. TRACK provided information which was used as input to the model and information which was used in validation of the model. For example, TRACK measured the overhead and the utilization of both processors. The CPU overhead measured by TRACK was incorporated into the service time of the users at the CPU node in the model. The CPU utilization measured by TRACK was compared with the CPU utilization predicted by the model. In addition to the utilization and overhead of the CPUs, TRACK measured the queue lengths associated with the disks and CPUs in the system.

In addition to TRACK, the program METER, which was also described in Chapter II, was used to trace the activities of a single synthetic user during each experiment. METER provided a time-stamped trace of the queue transfers associated with the user. From this trace, the mean response time, mean time spent at the CPU and mean time spent at the disk were calculated for the synthetic user to validate the corresponding results predicted by the model. Each experiment represented 20 synthetic users performing 60 time-sharing interactions and lasted for approximately 20 minutes.

The synthetic workload used in these experiments was not meant to be representative of the actual time-sharing workload of the DECsystem-10 at the Avionics Laboratory. Rather, the synthetic workload was meant to produce a measurable level of demands for system resources which are normally used by time-sharing users. Two classes of users were represented in the synthetic workload. The resource demands which characterize these two classes of users are shown in Table IV.

Experiment 1

The first experiment measured the performance of the DECsystem-10 during the execution of a workload consisting of 10 class one users and 10 class two users. During real-time processing, the secondary processor of the DECsystem-10 is primarily dedicated to real-time processes. Therefore, this experiment was executed with only the primary processor active to approximate the performance of the system as seen by time-sharing users during periods of time when real-time jobs are also executing.

Table IV

Synthetic User Characteristics

Class 1: Terminal wait = 300 time units

Processor time required = 43 time units

Number of I/Os = 10

Class 2: Terminal wait = 300 time units

Processor time required = 81 time units

Number of I/Os = 3

Fig. 11 shows the closed queueing network model for this experiment. Each resource is represented by a node in the network. The number in the upper left hand corner of each node specifies the node type, and the number in the lower right hand corner of each node represents the node identifier. Each u_i represents the service time of a class i user at the node. Each P_{ij} represents the transition probability that a class i user will proceed to node j when he leaves his current node.

The primary processor (CPUO) is represented by two nodes. One node represents the initial 3 jiffies (a jiffy is Digital Equipment Corporation terminology for the system clock cycle of 1/60 second) of processing time that a user receives when he leaves teletype I/O. The node is identified as the PQ1 node because the user is in the PQ1 run queue during this time. The second node represents the remaining processing time that a user requires when he is in the PQ2 round-robin run queue. Users only enter PQ2 if their processing requirements are not fulfilled during their PQ1 quantum time slice.

The service time of users at the CPU PQ2 node is increased to reflect the percentage of time that the processor cannot process users from the round-robin PQ2 run queue. This overhead is a sum of system overhead and the time that the processor spends servicing users from the PQ1 run queue. The system overhead is obtained from TRACK. However, to obtain the percentage of time that the processor spends servicing users from the PQ1 run queue, the model must be run assuming that no

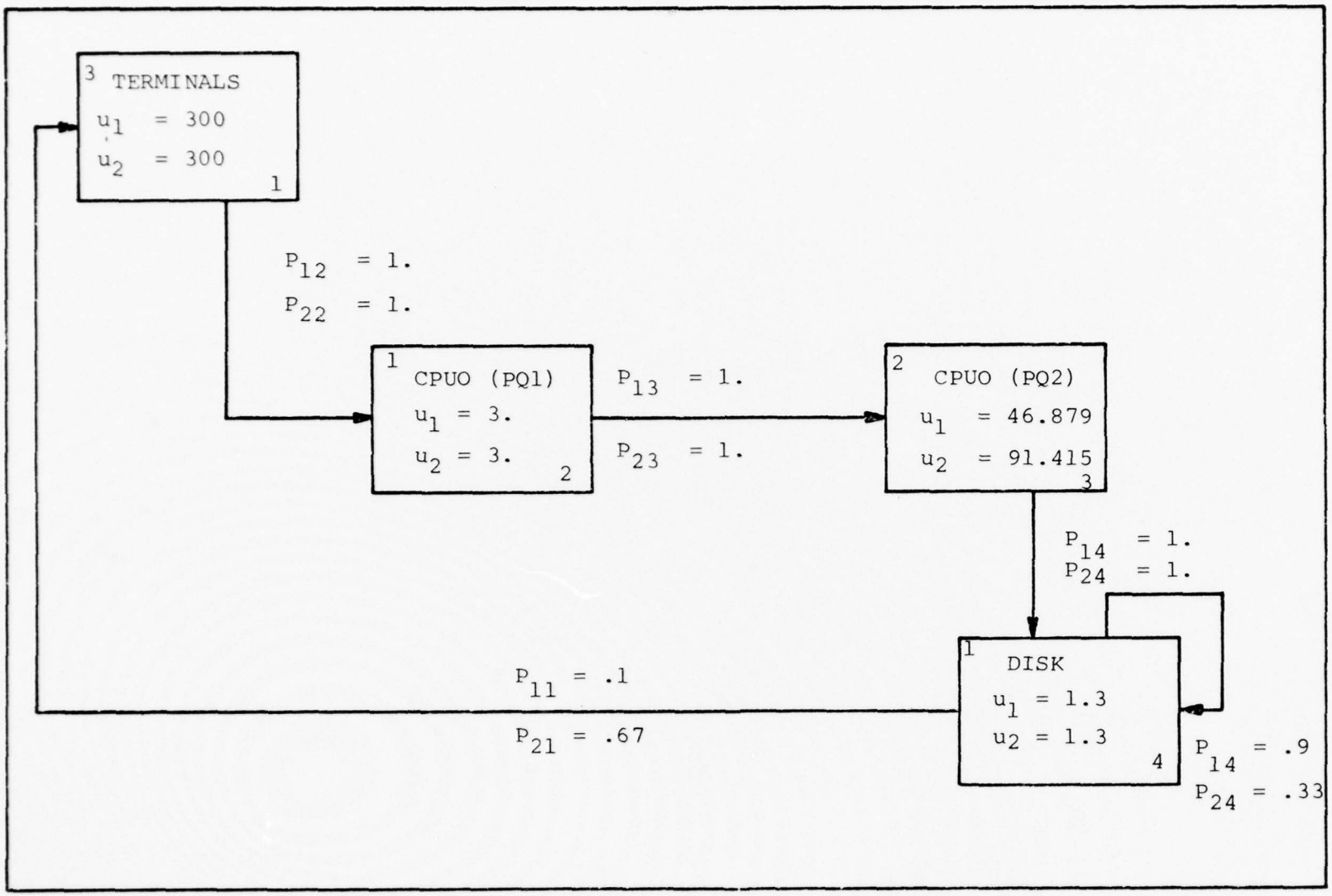


Fig. 11. Model of DECsystem-10 Used in Experiment 1

time is spent servicing users from the PQ1 run queue. The service time of users at the CPU PQ2 node is then increased based on the service rate of users at the CPU PQ1 node predicted by the model, and the model is rerun. One or two such iterations should produce assumed and predicted service rates for users at the CPU PQ1 node that are within specified tolerance levels. By including the PQ1 processing time as overhead at the CPU PQ2 node, the utilization of the CPU PQ2 node can be related to the utilization of the CPU.

The quantum processing time associated with the PQ1 run queue determines whether a job will complete processing while in the PQ1 run queue or will have to be placed in the round-robin PQ2 run queue for additional processing time. Therefore, this model of the central processor as two nodes allows the measurement of the effect changing the quantum processing time associated with the PQ1 run queue has on the response time of users. For example, increasing the quantum processing time associated with the PQ1 run queue decreases the probability of a job going to the round robin PQ2 run queue for additional processing. However, it also increases the overhead for jobs which are in the round robin PQ2 run queue. Thus, one of the cause and effect relationships within the DECsystem-10 can be analyzed.

The results of this validation experiment are shown in Table V. The errors encountered in the measured and predicted performance measures are less than 10%. Since the overhead of the software monitors was not included in the model, the response time errors are surprisingly low.

Table V

Experiment 1 Results

	Measured	Predicted	Error
Mean Response Time (class 1 user)	797.927 jiffies	729.562 jiffies	8.56%
Mean Response Time (class 2 user)	1280.354 jiffies	1378.476 jiffies	7.66%
Mean number of users in I/O wait queue	0.	.174	____
Mean number of users in run queues	15.3	15.074	1.47%
Mean number of users in teletype I/O queue	4.7	4.752	1.11%
CPU0 utilization	100%	100%	

Experiment 2

The second experiment was identical to the first experiment except that both processors were active. Fig. 12 shows the closed queueing network model which was used to predict performance measures in this experiment. In this model, the probability of a user being serviced by the primary or secondary processor when he is in the PQ2 run queue is assumed to be equal except for a skewing factor based on the different system overhead associated with each processor.

The results of this experiment are shown in Table VI. The errors in response time for the class 2 users and processor utilization are both higher than the errors for these performance measures in the first experiment. This increased error possibly relates to the assumptions which led to the construction of the model.

Experiment 3

The third experiment was conducted using the same data that was gathered in the previous experiment. This data was applied to a model with a different organization of the two processors than the organization used in Experiment 2. Fig. 13 shows the model used in this experiment. Rather than treat the two processors as separate nodes in the network, this model treats the two processors as a multiserver service center. The results of this experiment are shown in Table VII. The results show that both response time and CPU utilization are better approximated with the multiserver node than with the two single server node representation for the two processors.

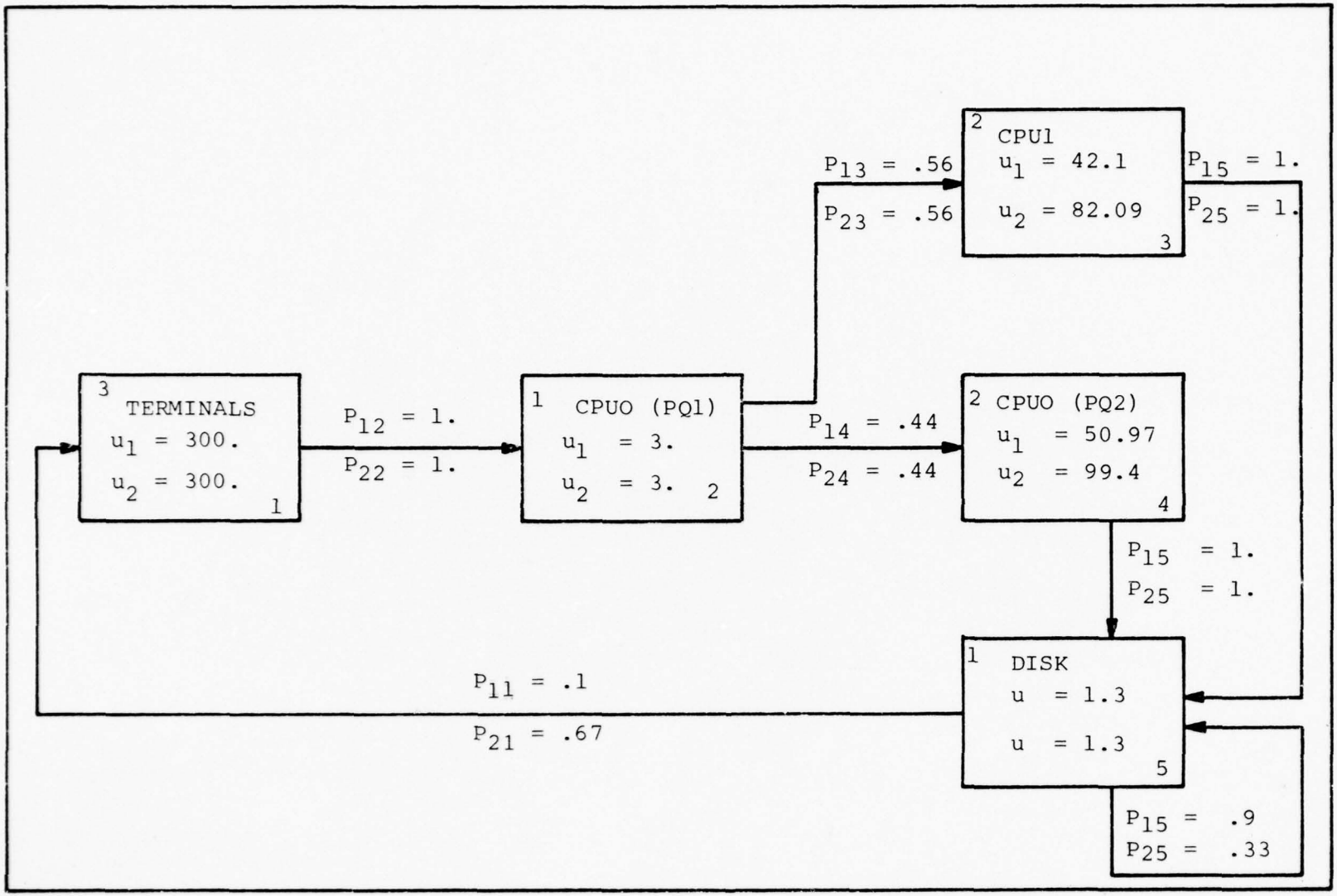


Fig. 12 Model of DECsystem-10 Used in Experiment 2

Table VI

Experiment 2 Results

	Measured	Predicted	Error
Mean Response Time (class 1 user)	281.867 jiffies	304.353 jiffies	7.97%
Mean Response Time (class 2 user)	456.84 jiffies	548.977 jiffies	20.16%
Mean number of users in I/O wait queue	.72	.35	51.38%
Mean number of users in run queues	15.39	11.16	27.48%
Mean number of users in teletype I/O queue	3.89	8.49	118.25%
CPU0 utilization	100%	91%	9.0%
CPU1 utilization	100%	91%	9.0%

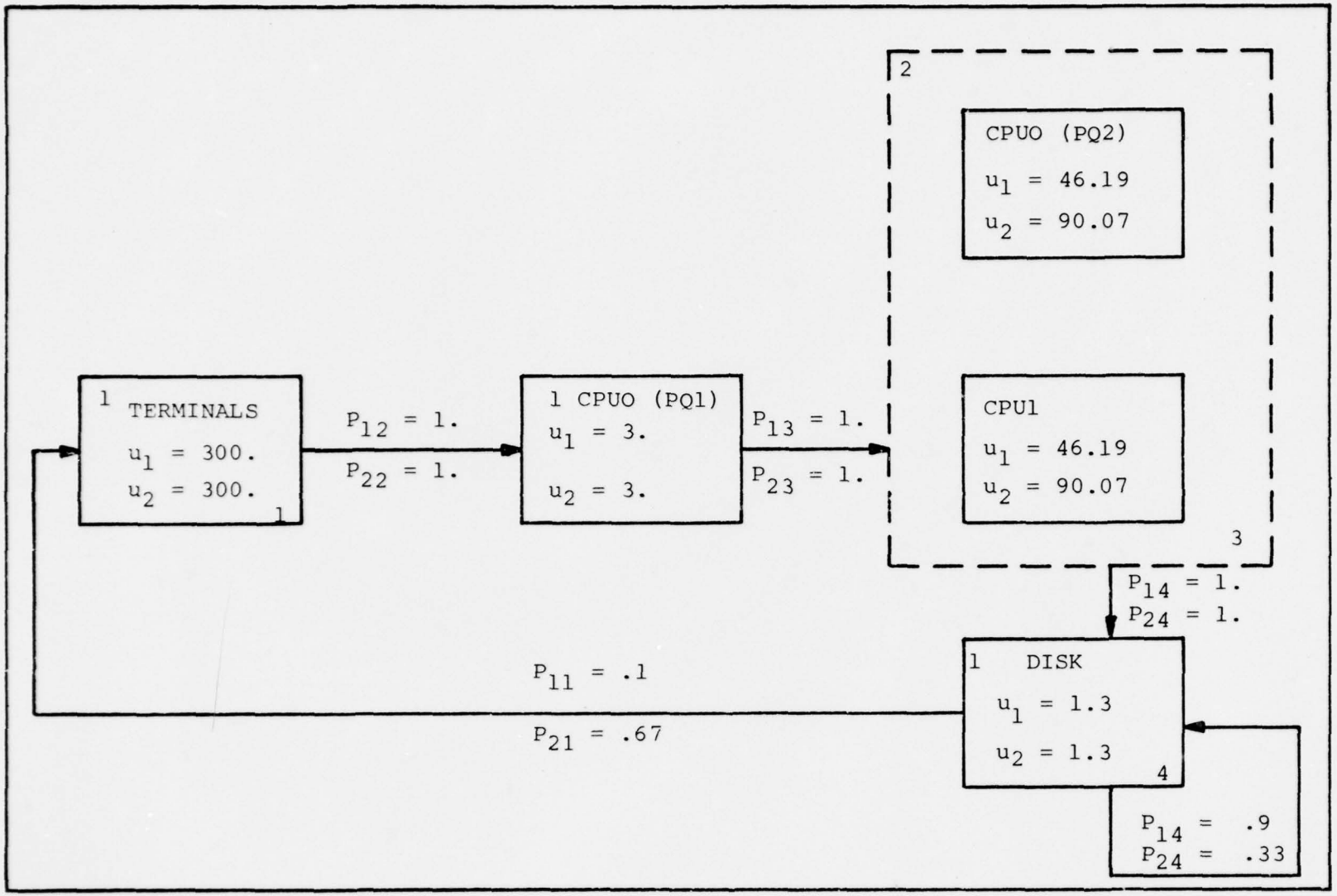


Fig. 13. Model of DECsystem-10 Used in Experiment 3

Table VII

Experiment 3 Results

	Measured	Predicted	Error
Mean Response Time (class 1 user)	281.867 jiffies	266.990 jiffies	5.27%
Mean Response Time (class 2 user)	456.84 jiffies	475.281 jiffies	4.03%
Mean number of users in I/O wait queue	.72	.38	47.22%
Mean number of users in run queues	15.39	10.46	32.03%
Mean number of users in teletype I/O queue	3.89	9.16	135.47%
CPU utilization	100%	98%	2.0%

Experiment 3 can be compared with Experiment 1 to analyze the effect that the activation of the secondary processor had on response time for the two classes of users. The measured and predicted average response times in Experiment 3 both show approximately a 280% improvement in average response time for the two classes of users over the measured values for average response time in Experiment 1. Although such a large improvement cannot be expected for most normal workloads, the model could be used to study the cause and effect relationship between real-time processing and time-sharing response time for various workloads. Of course, tests must first be performed with an active real-time process to determine if the time-sharing users are actually restricted to a single processor when real-time jobs are executing.

Experiment 4

Experiment 4 was an extension of Experiment 3. The core size of the jobs associated with the two classes of users were set at 100K for class 1 users and 80K for class 2 users to induce swapping. The measured results of this experiment are presented in Table VIII and Table IX. The utilization of the two processors presented in these tables indicates that approximately 60% of the time only one runable job was in main memory. This statement is supported by the average processing elapsed time for the two user classes. The average processing elapsed time for the two classes of users is close to the values previously set in Experiment 3 for the service

Table VIII

Experiment 4 Results For Class 1 Users

Mean Response Time	486.33 jiffies
Mean Swap In Time	370.08 jiffies
Mean Processor Time	46.64 jiffies
Mean I/O Time	59.65 jiffies
CPU0 Utilization	39%
CPU1 Utilization	92%

Table IX

Experiment 4 Results For Class 2 Users

Mean Response Time (one swap during teletype I/O)	480.22 jiffies
Mean Response Time (two swaps---one during teletype I/O and one during processing)	2329.41 jiffies
Mean Swap Time After Teletype I/O	361.91 jiffies
Mean Swap Time After In-Core-Protect-Time Exhausted	1860.23 jiffies
Mean Processor Time	93.8 jiffies
Mean I/O Time	19.05 jiffies

time of the users in the multiserver CPU node (see Fig. 13). Therefore, the data implies that there is little if any queueing of tasks at the processors in this experiment.

The jobs associated with the two classes of users were always swapped out of main memory during teletype I/O. After completion of teletype I/O, these jobs had to be swapped into main memory before they could be processed. The average swap in time for class 1 users was 370.08 jiffies and for class 2 users was 361.91 jiffies. These figures included the delay for core allocation as well as the delay for the physical I/O transfer of the jobs to main memory.

In this experiment, class 1 users were never swapped out of main memory during processing. However, class 2 users were swapped out of main memory during 50% of all interactions, indicating that their in-core-protect-time had been exhausted before they could finish processing. Class 2 users usually needed only a few jiffies of processing time before their next teletype I/O when they were swapped out. However, since class 2 users were in the PQ2 run queue when they were swapped out, users leaving teletype I/O and entering the PQ1 run queue had priority over the users in PQ2 to get swapped back into main memory. As a result, the average time that it took class 2 users to get swapped back into main memory to complete processing was 1860.23 jiffies. The average response time for class 2 users when they were swapped out of main memory while they were still in the PQ2 run queue was 2329.42 jiffies. When they were not swapped out their average response time

was only 480.22 jiffies, a decrease of 485%.

Modeling this type of activity with a closed queueing network model is difficult. Realistic memory allocation cannot be modeled because queueing models do not allow a task to have control of two resources at the same time. However, a possible solution to this problem is to insert a limited memory delay node and a disk swapping delay node into the model between the terminal node and the CPU nodes. By isolating the terminal and the limited memory delay nodes, the third type of state dependent service rate described in Chapter VI could be used to model the effect of limited memory. This could be done by restricting the flow of users out of these nodes into the rest of the network. Unfortunately, the time limitations of this thesis effort prevented the investigation of this possibility.

Conclusion

The experiments conducted indicate that the closed queueing network model presented in Chapters VI and VII is a viable model for performance evaluation of the DECsystem-10. The model allows a great deal of flexibility in modeling changes in the hardware configuration, workload, and the resource allocation algorithm parameters. The cause and effect relationship between the PQ1 quantum time slice and system response time for different classes of users can be studied. There is also the possibility that a state dependent solution can be derived for the representation of the concept of limited memory and its effect on system response time.

Future Efforts

The following are topics of studies which could be conducted as an extension of this thesis:

1. Additional experiments need to be executed to completely validate the model. Although the experiments conducted in this thesis effort indicate that the model is a valid CPE tool for performance evaluation of the DECsystem-10, additional experiments are needed for different workloads to increase confidence in the model.
2. The swapping algorithms of the DECsystem-10 need to be studied and tested to determine a state dependent service rate for the subset of nodes consisting of the terminal node and the limited memory delay node.
3. A workload characterization of the DECsystem-10 is needed to supply the input to the model to produce results which can be applied to the actual management and control of the DECsystem-10.

Bibliography

1. Dodson, Philip O. Control Data Corporations Cyber 70: Procedures for Performance Evaluation. Unpublished thesis. Wright-Patterson Air Force Base, Ohio: Air Force Institute of Technology, May 1974.
2. Bell, T.E., et. al. Computer Performance Analysis: Framework and Initial Phases for a Performance Improvement Effort. R-549-1-PR. Santa Monica, California: Rand Corporation, November 1972.
3. Datapro Research Corporation. "Digital Equipment DECsystem-10." Datapro: 70C-384-01a-70c-384-03b. Delran, New Jersey: Datapro Research Corporation, June 1976.
4. Digital Equipment Corporation. DECsystem-10 Monitor. Unpublished course handout. Maynard, Massachusetts: Digital Equipment Corporation, (no date available).
5. Digital Equipment Corporation. System Administrator's Guide to the 6.02 Scheduler. Maynard, Massachusetts: Digital Equipment Corporation, September 1975.
6. Digital Equipment Corporation. DECsystem-10 Operating System Commands Manual. Publication Number DEC-10-OSCMA-A-D. Maynard, Massachusetts: Digital Equipment Corporation, May 1974.
7. Digital Equipment Corporation. DECsystem-10 Monitor Calls. Publication Number DEC-10-OMCMA-B-D. Maynard Massachusetts: Digital Equipment Corporation, 1976.

8. Helleman, H. and T.F. Conroy. Computer System Performance. New York: McGraw-Hill Book Co., 1975.

9. Svobodova, L. Computer Performance Measurement and Evaluation Methods: Analysis and Applications. New York: American Elsevier Publishing Co., 1976.

10. Kimbleton, S.R. A Partial Structure for Performance Evaluation. Ann Arbor, Michigan: University of Michigan, May 1973. AD A010214.

11. Lucas, H.C. "Performance Evaluation and Monitoring." Computing Surveys, 3: 79-91 (September 1971).

12. Bell, T.E. Computer Performance Management Through Control Limits. TRW-55-76-01. Redondo Beach, California: TRW, January 1976.

13. Sekino, A. Performance Evaluation of Multiprogrammed Time-Shared Computer Systems. MAC TR-103. Cambridge, Massachusetts: Massachusetts Institute of Technology, September 1972. AD 749949.

14. Drummond, M.E. Evaluation and Measurement Techniques for Digital Computer Systems. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1973.

15. Bell, T.E. Computer Performance Analysis: Measurement Objectives and Tools. R-584-NASA/PR. Santa Monica, California: Rand Corporation, February 1971.

16. Nutt, J.G. Computer System Monitoring Techniques. CU-CS-013-73. Boulder, Colorado: University of Colorado, February 1973.

17. Fishman, G.S. Concepts and Methods in Discrete Event Digital Simulation. New York: John Wiley and Sons, Inc., 1973.

18. McLeod, J. "Simulation Is Wha-a-t?" Simulation, 1: (Fall 1963).

19. Coffman, E.G. and P.J. Denning, Operating Systems Theory. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1973.

20. Muntz, R.R. "Analytic Modeling of Interactive Systems." Proceedings of the IEEE, 63: 946-953 (June 1975).

21. Wong, J.W. Queueing Network Models for Computer Systems. UCLA-ENG-7579. Los Angeles, California; UCLA, October 1975.

22. Moore, C.G. Network Models for Large-Scale Time-Sharing Systems. Technical Report 71-1. Ann Arbor, Michigan: University of Michigan, April 1971. AD 727206

23. Jackson, J.R. "Jobshop-Like Queueing Systems." Management Science, 10:131-142 (October 1963).

24. Gordon, W.J. and G.F. Newell. "Closed Queueing Systems with Exponential Servers." Operations Research, 15:254-265 (March 1967).

25. Baskett, F., et. al. "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers." Journal of the ACM, 22:248-260 (April 1975).

26. Scherr, A. An Analysis of Time-Shared Computer Systems. Cambridge, Massachusetts: MIT Press, 1967.

27. Chien, P.P. "Queueing Network Model of Interactive Computing Systems." Proceedings of the IEEE, 63:954-957 (June 1975).

28. Kleinrock, L. Queueing Systems, Volume I: Theory. New York: John Wiley and Sons, Inc., 1975.

29. Cox, D.R. "A Use of Complex Probabilities in the Theory of Stochastic Processes." Proceedings of the Cambridge Philosophical Society, 51:313-319 (1955).

30. Chandy, K.M. "The Analysis and Solutions for General Queueing Networks." Proceedings Sixth Annual Princeton Conference on Information Sciences and Systems: 224-228. Princeton, New Jersey: Princeton University, March 1972.

31. Muntz, R.R. and J.W. Wong. "Efficient Computational Procedures for Closed Queueing Network Models." Proceedings of the 7th Hawaii International Conference on System Sciences: 33-36. Honolulu, Hawaii: University of Hawaii, January 1974.

32. Buzen, J.P. "Computational Algorithms for Closed Queueing Networks With Exponential Servers." Communications of the ACM, 16:527-531 (September 1973).

33. Little, J.D. "A Proof of the Queueing Formula $L = \lambda W$." Operations Research, 9:383-387 (May 1961).

Appendix A

Source Listing of Program

```
      PROGRAM PERFORM(INPUT,OUTPUT,TAPE3,TAPE5=INPUT,TAPE6=OUTPUT)
CCC
CCC
CCC         THE PROGRAM PERFOR4 IS A CLOSED QUEUEING NETWORK
CCC   MODEL WHICH CAN BE USED IN PERFORMANCE EVALUATION OF
CCC   TIME-SHARING COMPUTER SYSTEMS.  THE PROGRAM PERFORM IS
CCC   BASED ON A CLOSED QUEUEING NETWORK MODEL DEVELOPED BY
CCC   F. BASKETT, K. CHANDY, R. MUNTZ, AND F. PALACIOS.
CCC   THEIR MODEL WAS PRESENTED IN THE ARTICLE "OPEN, CLOSED,
CCC   AND MIXED NETWORK OF QUEUES WITH DIFFERENT CLASSES OF
CCC   CUSTOMERS", WHICH APPEARED IN THE JOURNAL OF THE ACM,
CCC   VLO. 22, NO. 2, PP. 248-260, APRIL, 1975.
CCC
CCC
CCC         THE COMPUTATIONAL ALGORITHMS WHICH ARE USED TO
CCC   IMPLEMENT THE MODEL OF BASKETT, ET. AL. ARE BASED ON THE
CCC   COMPUTATIONALLY EFFICIENT ALGORITHMS PRESENTED IN THE
CCC   DISSERATION OF J. WONG ENTITLED QUEUEING NETWORK MODELS
CCC   FOR COMPUTER SYSTEMS, UCLA-ENG-7579, UNIVERSITY OF
CCC   CALIFORNIA AT LOS ANGELES, OCTOBER, 1975.
CCC
CCC
CCC
      DIMENSION E(10,5), EXPVAL(10,5), DEPART(10,5), ENT(1000),
     1IFACTOR(41), INDEX3(31), ISKIP(5), ISTATE1(5), ISTATE2(5),
     2NODETYP(10), NORMCON(1000,2), NUSERS(5), PROB(10,10),
     3PROBCUM(1000), PROBMAR(1000), PROBPAR(10,5,41),
     4PROBTOT(10,41), SERVICE(10,5), UTIL(10,5), WAREA(10),
     5X(10), IDEP(10), DEP(10,5,40)
      DOUBLE PRECISION IFACTOR
      MAXNODE = 10
      MAXUSER = 40
      MAXUSE1 = MAXUSER + 1
      MAXTYPE = 5
      MAXSTAT = 1000
      ISIZE3 = 31
```

```
CCC
CCC
CCC
CCC   MAXNODE --- THE MAXIMUM NUMBER OF NODES WHICH CAN BE
CCC                REPRESENTED IN THE PROGRAM.
CCC
CCC   MAXUSER --- THE MAXIMUM NUMBER OF TOTAL USERS THAT CAN
CCC                BE REPRESENTED IN THE PROGRAM.
CCC
CCC   MAXUSER1 --- THE MAXIMUM NUMBER OF TOTAL USERS THAT CAN
CCC                 BE REPRESENTED IN THE PROGRAM + 1.
CCC
CCC   MAXTYPE --- THE MAXIMUM NUMBER OF TYPES OF USERS THAT
CCC                CAN BE REPRESENTED IN THE PROGRAM.
CCC
CCC   MAXSTAT --- THE MAXIMUM STATE SPACE FOR EACH NODE.
CCC
CCC   ISIZE3 --- THE MAXIMUM NUMBER OF RANDOM RECORDS (30)
CCC                WHICH CAN BE WRITTEN ON TAPE3 + 1. ISIZE3 IS
CCC                CALCULATED BY MAXNODE * 3 + 1.
CCC
CCC   E(I,J) --- THE MEAN ARRIVAL RATE OF TYPE J USERS AT
CCC               SERVICE CENTER I.  THE DIMENSIONS OF E ARE
CCC               (MAXNODE,MAXTYPE).
CCC
CCC   EXPVAL(I,J)  --- THE EXPECTED NUMBER OF TYPE J USERS
CCC                     AT NODE I AT EQUILIBRIUM.  THE DIMENSIONS
CCC                     OF EXPVAL ARE (MAXNODE,MAXTYPE).
CCC
CCC   DEPART(I,J)  --- THE DEPARTURE RATE OF TYPE J USERS FROM
CCC                     NODE I AT EQUILIBRIUM.  THE DIMENSIONS
CCC                     OF DEPART ARE (MAXNODE,MAXTYPE).
CCC
CCC   FNT(I) --- THE FUNCTION REPRESENTING THE RELATIVE
CCC               WEIGHTING OF STATE I IN THE CALCULATIONS OF
CCC               THE EQUILIBRIUM STATE PROBABILITY DISTRIBUTIONS.
```

```
CCC             THE DIMENSIONS OF FNT IS (MAXSTAT).
CCC
CCC   IFACTOR(I) --- VALUE OF I FACTORIAL.  DIMENSION OF
CCC                  IFACTOR IS (MAXUSER1).
CCC
CCC   INDEX3(I) --- THE LOCATION OF RECORD I ON THE RANDOM
CCC                 FILE TAPE3.  THE DIMENSION OF INDEX3 IS
CCC                 (ISIZE3).
CCC
CCC   ISKIP --- WORKING ARRAY USED TO SKIP OVER UNNEEDED
CCC             STATES IN THE STATE SPACE FOR A GIVEN NODE.
CCC             THE DIMENSION OF ISKIP IS (MAXTYPE).
CCC
CCC   ISTATE1 -- WORKING ARRAY USED TO SPECIFY A STATE OF A
CCC              GIVEN NODE.  THE DIMENSION OF ISTATE1 IS (MAXTYPE).
CCC
CCC
CCC   ISTATE2 --- WORKING ARRAY USED TO SPECIFY A STATE OF A
CCC               GIVEN NODE.  THE DIMENSION OF ISTATE2 IS
CCC               (MAXTYPE).
CCC
CCC   NODETYP(I) --- SPECIFIES THE TYPE OF NODE I.  THE
CCC                  DIMENSION OF NODEYYP IS (MAXNODE).
CCC
CCC   NORMCON --- WORKING ARRAY USED TO CALCULATE THE NORMAL-
CCC               IZATION CONSTANT.  THE DIMENSIONS OF NORMCON
CCC               ARE (MAXSTAT,2).
CCC
CCC   NUSERS(I) --- NUMBER OF TYPE I USERS REPRESENTED IN THE
CCC                 MODEL.  THE DIMENSION OF NUSERS IS (MAXTYPE).
CCC
CCC   PROB(I,J) --- THE PROBABILITY THAT A TASK OF A GIVEN TYPE
CCC                 WILL PROCEED TO NODE I WHEN IT DEPARTS
CCC                 NODE J.  THE DIMENSIONS OF PROB ARE
CCC                 (MAXNODE,MAXNODE).
CCC
```

```
CCC   PROBCUM(I) --- THE PROBABILITY THAT A SUBNETWORK OF
CCC                  NODES IN THE NETWORK MODEL IS IN A GIVEN
CCC                  STATE.  THE STATES IN THE STATE SPACE ARE
CCC                  INDEXED BY THE SUBSCRIPT I.  THE DIMENSION
CCC                  OF PROBCUM IS (MAXSTAT).
CCC
CCC   PROBMAR(I) --- THE PROBABILITY THAT A SINGLE NODE IN
CCC                  THE NETWORK MODEL IS IN A GIVEN STATE.
CCC                  THE STATES IN THE STATE SPACE ARE INDEXED
CCC                  BY THE SUBSCRIPT I.  THE DIMENSION OF
CCC                  PROBMAR IS (MAXSTAT).
CCC
CCC   PROBPAR(I,J,K) --- THE PROBABILITY THAT THERE ARE K
CCC                      USERS OF TYPE J AT NODE I.  THE
CCC                      DIMENSIONS OF PROBPAR ARE (MAXNODE,
CCC                      MAXTYPE,MAXUSE1).
CCC
CCC   PROBTOT(I,J) --- THE PROBABILITY THAT THERE ARE J USERS
CCC                    AT NODE I.  THE DIMENSIONS OF PROBTOT
CCC                    ARE (MAXNODE,MAXTYPE).
CCC
CCC   SERVICE(I,J) --- THE SERVICE TIME OF A TYPE J USER AT
CCC                    NODE I.  THE DIMENSIONS OF SERVICE
CCC                    ARE (MAXNODE,MAXTYPE).
CCC
CCC   UTIL(I,J) --- THE UTILIZATION OF NODE I BY USERS OF TYPE
CCC                 J.  THE DIMENSIONS OF UTIL ARE (MAXNODE,
CCC                 MAXTYPE).
CCC
CCC   WAREA --- WORKING ARRAY USED TO CALCULATE THE MEAN ARRIVAL
CCC             RATE OF USERS AT THE NODES IN THE NETWORK
CCC             THE DIMENSION OF WAREA IS (MAXNODE).
CCC
CCC   X --- WORKING ARRAY USED TO CALCULATE THE MEAN ARRIVAL
CCC         RATE OF USERS AT THE NODES IN THE NETWORK.
CCC         DIMENSION OF X IS (MAXNODE).
```

```
CCC
CCC    IDEP(I) --- INDICATES STATE DEPENDENT SERVICE
CCC                RATES AT NODE I.
CCC                 1 --- NO STATE DEPENDENT SERVICE RATES.
CCC                 2 --- SERVICE RATES DEPENDENT ON THE TOTAL
CCC                       NUMBER OF USERS AT THE NODE.
CCC                 3 --- SERVICE RATES DEPENDENT ON THE NUMBER
CCC                       OF USERS OF A TYPE AT THE NODE.
CCC                 THE DIMENSION OF IDEP IS (MAXNODE).
CCC
CCC    DEP(I,J,K) --- THE RELATIVE SERVICE RATE OF TYPE J USERS
CCC                   AT NODE I WHEN THERE ARE K USERS PRESENT
CCC                   AT NODE I.  THE DIMENSIONS OF DEP ARE
CCC                   (MAXNODE,MAXTYPE,MAXUSER).
CCC
CCC
CCC
      WRITE(6,1)
    1 FORMAT(1H1,9X,"THIS IS A CLOSED QUEUEING NETWORK MODEL ",
     1"FOR TIME-SHARING COMPUTER SYSTEMS.")
CCC
CCC    OPEN THE MASS STORAGE FILE.
CCC
      CALL OPENMS(3,INDEX3,ISIZE3,0)
CCC
CCC    CALL INPUT TO INPUT THE DATA NECESSARY TO EXECUTE THE
CCC    PROGRAM.
CCC
      CALL INDATA(PROB,SERVICE,E,NODETYP,NUSERS,MAXNODE,
     1MAXUSER,MAXTYPE,MAXSTAT,NODES,IUSERS,
     1ITYPES,X,INDEX3,ISIZE3,WAREA,ISTATES,IDEP,DEP),RETURNS(10)
CCC
CCC    CALL FUNCT TO CALCULATE THE FUNCTIONS FNT FOR ALL NODES.
CCC
      CALL FUNCT(E,SERVICE,IFACTOR,ISTATE1,FNT,NUSERS,NODETYP,
     1NODES,IUSERS,ITYPES,MAXNODE,MAXUSER,MAXUSE1,MAXTYPE,
```

```
     2MAXSTAT,INDEX3,ISIZE3,ISTATES,IDEP,DEP),RETURNS(10)
CCC
CCC    CALL NORMAL TO CALCULATE THE NORMALIZATION CONSTANT.
CCC
       CALL NORMAL(ENT,NORMCON,NUSERS,ISTATE1,ISTATE2,ISKIP,
     1MAXNODE,MAXSTAT,NODES,ITYPES,MAXTYPE,INDEX3,ISIZE3,ISTATES)
CCC
CCC    CALL MARGIN TO CALCULATE THE MARGINAL PROBABILITY
CCC    DISTRIBUTION FOR THE STATE OF THE NODES.
CCC
CCC
       CALL MARGIN(ENT,NORMCON,PROBMAR,PROBCUM,NUSERS,ISTATE1,ISTATE2,
     1ISKIP,MAXNODE,MAXSTAT,MAXTYPE,NODES,ITYPES,INDEX3,ISIZE3,ISTATES)
CCC
CCC    CALL EXPECT TO CALCULATE VARIOUS PERFORMANCE MEASURES.
CCC
       CALL EXPECT(PROBMAR,PROBTOT,PROBPAR,NUSERS,ISTATE1,
     1NODETYP,EXPVAL,DEPART,UTIL,SERVICE,E,MAXNODE,
     1MAXUSE1,MAXTYPE,NODES,ITYPES,IUSERS,MAXSTAT,INDEX3,ISIZE3,ISTATES,
     2IDEP,DEP,MAXUSER)
       GO TO 20
   10  WRITE(6,15)
   15  FORMAT(//10X,"ERRORS HAVE OCCURRED IN THE INPUT DESCRIPTION.",
     1//2X,"CHECK YOUR INPUT DATA FOR CORRECTNESS.")
   20  CONTINUE
CCC
CCC    CLOSE THE MASS STORAGE FILE.
CCC
       CALL CLOSMS(3)
       STOP
       END
```

```
      SUBROUTINE INDATA(PROB,SERVICE,E,NODETYP,NUSERS,MAXNODE,
     1MAXUSER,MAXTYPE,MAXSTAT,NODES,IUSERS,ITYPES,X,INDEX3,ISIZE3,
     2WAREA,ISTATES,IDEP,DEP),RETURNS(ERROR)
      DIMENSION PROB(MAXNODE,MAXNODE), SERVICE(MAXNODE,MAXTYPE),
     1E(MAXNODE,MAXTYPE), NODETYP(MAXNODE), NUSERS(MAXTYPE),
     2X(MAXNODE), WAREA(MAXNODE), INDEX3(ISIZE3), IDEP(MAXNODE),
     3DEP(MAXNODE,MAXTYPE,MAXUSER)
CCC
CCC   SUBROUTINE INPUT READS THE DATA NECESSARY TO RUN THE
CCC   PROGRAM.  THE CHARACTERISTICS OF THE TIME-SHARING COMPUTER
CCC   SYSTEM BEING MODELED ARE INPUT IN THIS ROUTINE.
CCC
      WRITE(6,1)
    1 FORMAT(////10X,"*****THE INPUT TO THE MODEL FOLLOWS*****")
CCC
CCC   READ THE NUMBER OF NODES IN THE NETWORK AND THE NUMBER
CCC   OF TYPES OF TASKS.
CCC
      READ(5,10) NODES, ITYPES
   10 FORMAT(2I10)
      IF(EOF(5)) 1000,15
   15 CONTINUE
      IF((NODES .GT. MAXNODE) .OR. (ITYPES .GT. MAXTYPE)) GO TO 1100
      WRITE(6,20) NODES, ITYPES
   20 FORMAT(//10X,"THE NUMBER OF NODES REPRESENTED IN THE MODEL = ",
     1I3//10X,"THE NUMBER OF TASK TYPES REPRESENTED IN THE ",
     2"MODEL = ",I3)
      READ(5,25) (NUSERS(I),I=1,ITYPES)
   25 FORMAT(8I10)
      IF(EOF(5)) 1000,30
   30 CONTINUE
      IUSERS = 0
      ISTATES = 1
      WRITE(6,35)
   35 FORMAT(////10X,"THE DISTRIBUTION OF TASKS IN THE SYSTEM ",
     1"BY TASK TYPE "//17X,"TASK TYPE",10X,"NUMBER OF TASKS")
```

```
      DO 45 I = 1, ITYPES
      WRITE(6,40) I, NUSERS(I)
   40 FORMAT(/20X,I3,20X,I3)
      IUSERS = IUSERS + NUSERS(I)
      ISTATES = ISTATES * (NUSERS(I) + 1)
   45 CONTINUE
      WRITE(6,46) IUSERS
   46 FORMAT(//18X,"TOTAL",20X,I3)
      IF((IUSERS .GT. MAXUSER) .OR. (ISTATES .GT. MAXSTAT)) GO TO 1200
CCC
CCC   READ THE NODE CHARACTERISTICS.
CCC
      WRITE(6,48) ITYPES
   48 FORMAT(////10X,"NODE CHARACTERISTICS"//10X,"NODE",5X,"NODE",5X,
     1"SERVICE"/10X,"NAME",5X,"TYPE",7X,"RATE",6X,"NODE SERVICE RATES ",
     2"FOR TASK TYPES 1 THROUGH",I3)
      DO 65 I = 1, NODES
      READ(5,50) NODETYP(I), IDEP(I), (SERVICE(I,J),J=1,ITYPES)
   50 FORMAT(I10,I10,10F10.5)
      IF(EOF(5)) 1000, 55
   55 CONTINUE
      IF(IDEP(I) .EQ. 2) READ(5,56) (DEP(I,1,J),J=1,MAXUSER)
   56 FORMAT(4(/8F10.5))
      IF(IDEP(I) .NE. 3) GO TO 59
      DO 58 J = 1, ITYPES
      READ(5,56) (DEP(I,J,K),K=1,MAXUSER)
   58 CONTINUE
   59 CONTINUE
      WRITE(6,60) I, NODETYP(I), IDEP(I), (SERVICE(I,J),J=1,ITYPES)
   60 FORMAT(/10X,I4,5X,I4,7X,I4,6X,8F10.5,10(/41X,8F10.5))
   65 CONTINUE
CCC
CCC   READ THE TRANSITION PROBABILITIES BETWEEN NODES.
CCC
      WRITE(6,67)
   67 FORMAT(////10X,"TRANSITION PROBABILITIES DESCRIBING ",
```

```
     1"MOVEMENT OF TASKS AMONG THE MODEL'S NODES")
      DO 110 K = 1, ITYPES
      WRITE(6,68) K, NODES
   68 FORMAT(///10X,"TRANSITION PROBABILITIES FOR TASK TYPE ",
     1I3//10X,"DEPARTURE",7X,"PROBABILITY OF TASK MOVEMENT FROM"/12X,
     2"NODE",9X,"DEPARTURE NODE TO NODES 1 THROUGH",I3)
      DO 75 I = 1, NODES
      READ(5,70) (PROB(J,I),J=1,NODES)
   70 FORMAT(8F10.5)
      WRITE(6,74) I, (PROB(J,I),J=1,NODES)
   74 FORMAT(/12X,I4,10X,10F10.5,10(/24X,10F10.5))
   75 CONTINUE
      NUMPROB = MAXNODE * MAXNODE
      CALL WRITMS(3,PROB(1,1),NUMPROB,1)
      DO 80 I = 1, NODES
      PROB(I,I) = PROB(I,I) - 1.
      PROB(1,I) = 0.
      X(I) = 0.
   80 CONTINUE
      PROB(1,1) = 1.
      X(1) = 1.
CCC
CCC   THE RELATIVE TRANSITION RATE BETWEEN NODES IS CALCULATED
CCC   SOLVING THE SET OF LINEAR EQUATION SPECIFIED BY
CCC   PROB AND X.  THE SUBROUTINE LEQTIF SOLVES THIS SET OF
CCC   LINEAR EQUATIONS.  LEQTIF IS A SYSTEM SUPPORTED SUBROUTINE
CCC   AT WRIGHT-PATTERSON AFB, OHIO.
CCC
      CALL LEQT1F(PROB,1,NODES,MAXNODE,X,0,WAREA,IER)
      CALL READMS(3,PROB(1,1),NUMPROB,1)
      DO 100 I = 1, NODES
      GAMMA = 0.
      DO 90 J = 1, NODES
      GAMMA = GAMMA + (PROB(I,J) * X(J))
   90 CONTINUE
      IF(ABS(GAMMA - X(I)) .LT. .1) GO TO 100
```

```
      WRITE(6,95) I, (X(L),L=1,NODES)
   95 FORMAT(10X,"THE RELATIVE SERVICE RATE BETWEEN NODES CALCULATED ",
     1"FROM THE COEFFICIENT ARRAY PROB IS IN ERROR FOR NODE ",
     2I3//5X,"THE RELATIVE SERVICE RATES ARE.",/10X,
     310(/10(F10.5,2X)))
      GO TO 115
  100 CONTINUE
      DO 105 I = 1, NODES
      E(I,K) = X(I)
  105 CONTINUE
  110 CONTINUE
  115 CONTINUE
      RETURN
 1000 WRITE(6,1005)
 1005 FORMAT(//10X,"THERE IS NO QUEUING NETWORK DATA ON THE INPUT FILE")
      RETURN ERROR
 1100 WRITE(6,1105) MAXNODE, MAXTYPE, NODES, ITYPES
 1105 FORMAT(//10X,"THE MAXIMUM NUMBER OF NODES = ",
     1I3," AND THE MAXIMUM NUMBER OF TASKS TYPES = ",I3/2X,
     1"YOUR INPUTS ARE ",I3," AND ",I3," RESPECTIVELY.")
      RETURN ERROR
 1200 WRITE(6,1205) MAXUSER, MAXSTAT, IUSERS, ISTATES
 1205 FORMAT(//10X,"THE MAXIMUM NUMBER OF TASKS = ",
     1I3, " AND THE MAXIMUM NUMBER OF STATES = ",I3/2X,
     2"YOUR INPUTS ARE ",I3," AND ",I3," RESPECTIVELY.")
      RETURN ERROR
      END
```

```
      SUBROUTINE FUNCT(E,SERVICE,IFACTOR,ISTATE,FNT,NUSERS,NODETYP,
     1NODES,IUSERS,ITYPES,MAXNODE,MAXUSER,MAXUSE1,
     2MAXTYPE,MAXSTAT,INDEX3,ISIZE3,ISTATES,IDEP,DEP),RETURNS(ERROR)
      DIMENSION E(MAXNODE,MAXTYPE), SERVICE(MAXNODE,MAXTYPE),
     1IFACTOR(MAXUSE1), ISTATE(MAXTYPE), FNT(MAXSTAT),
     2NUSERS(MAXTYPE), NODETYP(MAXNODE), INDEX3(ISIZE3),
     3IDEP(MAXNODE), DEP(MAXNODE,MAXTYPE,MAXUSER)
      DOUBLE PRECISION IFACTOR
CCC
CCC   THIS SUBROUTINE CALCULATES THE VALUES FOR THE FUNCTIONS
CCC   WHICH ARE USED TO CALCULATE THE PROBABILITY THAT THE
CCC   COMPUTER SYSTEM BEING MODELED IS IN A GIVEN STATE AT
CCC   AT EQUILIBRIUM.
CCC
      IFACTOR(1) = 1
      DO 10 I = 1, MAXUSER
      IFACTOR(I+1) = IFACTOR(I) * I
   10 CONTINUE
      DO 60 I = 1, NODES
      INDEX = 1
      DO 20 J = 1, ITYPES
      ISTATE(J) = 0
   20 CONTINUE
   25 CONTINUE
      NUMBER = 1
      DO 30 J = 1, ITYPES
      NUMBER = NUMBER + ISTATE(J)
   30 CONTINUE
      J = 2
   35 CONTINUE
      FVALUE1 = 1.
      FVALUE2 = 1.
      DO 38 K = 1, ITYPES
      L = ISTATE(K)
      FVALUE1 = FVALUE1 * ((E(I,K) * SERVICE(I,K))**L)
      FVALUE2 = FVALUE2 * IFACTOR(L+1)
```

```
38 CONTINUE
   IF(NODETYP(I) .EQ. 3) FVALJE2 = 1. / FVALUE2
   IF(NODETYP(I) .NE. 3) FVALUE2 = IFACTOR(NUMBER) / FVALUE2
   FNT(INDEX) = FVALUE1 * FVALUE2
   IF(IDEP(I) .NE. 2) GO TO 41
   NUM = NUMBER - 1
   IF(NUM .EQ. 0) GO TO 41
   FFF = 1.
   DO 40 K = 1, NUM
   FFF = FFF * DEP(I,1,K)
40 CONTINUE
   FNT(INDEX) = FNT(INDEX) / FFF
41 CONTINUE
   IF(IDEP(I) .NE. 3) GO TO 44
   FFF = 1.
   DO 43 K = 1, ITYPES
   IF(ISTATE(K) .EQ. 0) GO TO 43
   NUM = ISTATE(K)
   DO 42 KK = 1, NUM
   FFF = FFF * DEP(I,K,KK)
42 CONTINUE
43 CONTINUE
   FNT(INDEX) = FNT(INDEX) / FFF
44 CONTINUE
   INDEX = INDEX + 1
   NUMBER = NUMBER + 1
   ISTATE(1) = ISTATE(1) + 1
   IF(ISTATE(1) .LE. NUSERS(1)) GO TO 35
45 CONTINUE
   ISTATE(J) = ISTATE(J) + 1
   IF(ISTATE(J) .GT. NUSERS(J)) GO TO 55
   K = J - 1
   DO 50 L = 1, K
   ISTATE(L) = 0
50 CONTINUE
   GO TO 25
```

```
   55 CONTINUE
      J = J + 1
      IF(J .LE. ITYPES) GO TO 45
CCC
CCC   THE FUNCTIONAL VALUES CALCULATED FOR ALL POSSIBLE STATES
CCC   OF EACH NODE ARE SAVED SO THAT THEY CAN BE USED TO
CCC   CALCULATE THE NORMALIZATION CONSTANT.
CCC
      CALL WRITMS(3,FNT(1),ISTATES,I)
   60 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE NORMAL(FNT,NORMCON,NUSERS,ISTATE1,
     1ISTATE2,ISKIP,MAXNODE,MAXSTAT,NODES,ITYPES,
     2MAXTYPE,INDEX3,ISIZE3,ISTATES)
      DIMENSION FNT(MAXSTAT), NORMCON(MAXSTAT,2),
     1NUSERS(MAXTYPE), ISTATE1(MAXTYPE), ISTATE2(MAXTYPE),
     2ISKIP(MAXTYPE), INDEX3(ISIZE3)
      INTEGER FINDEX
      REAL NORMCON
CCC
CCC   THIS SUBROUTINE CALCULATES THE NORMALIZATION CONSTANT
CCC   FOR THE MODEL.  THE NORMALIZATION CONSTANT ASSURES THAT
CCC   THE PROBABILITY OF THE SYSTEM BEING IN ALL STATES SUMS
CCC   TO UNITY.
CCC
      CALL READMS(3,NORMCON(1,1),ISTATES,1)
      CALL WRITMS(3,NORMCON(1,1),ISTATES,MAXNODE+1)
      NORM1 = 1
      NORM2 = 2
      DO 110 I = 2, NODES
      CALL READMS(3,FNT(1),ISTATES,I)
      INDEX = 1
      DO 15 J = 1, ITYPES
      ISTATE1(J) = 0
   15 CONTINUE
   20 CONTINUE
      K = 2
      DO 25 J = 1, ITYPES
      ISTATE2(J) = 0
   25 CONTINUE
      KK = 1
      FINDEX = 1
      LINDEX = INDEX
      ITEMP = 1
      DO 30 J = 1, ITYPES
      ISKIP(J) = ITEMP*(NUSERS(J) - ISTATE1(J))
      ITEMP = ITEMP * (NUSERS(J) + 1)
```

```
30 CONTINUE
   ISKIP(1) = ISKIP(1) + 1
   FVALUE = 0.
40 CONTINUE
   FVALUE = FVALUE + NORMCON(FINDEX,NORM1) * FNT(LINDEX)
   IF(FINDEX .EQ. INDEX) GO TO 70
   ISTATE2(KK) = ISTATE2(KK) + 1
   IF(ISTATE2(KK) .GT. ISTATE1(KK)) GO TO 50
   FINDEX = FINDEX + 1
   LINDEX = LINDEX - 1
   GO TO 40
50 CONTINUE
   FINDEX = FINDEX + ISKIP(KK)
   LINDEX = LINDEX - ISKIP(KK)
   KK = KK + 1
   ISTATE2(KK) = ISTATE2(KK) + 1
   IF(ISTATE2(KK) .GT. ISTATE1(KK)) GO TO 50
   L = KK - 1
   DO 60 J = 1, L
   ISTATE2(J) = 0
60 CONTINUE
   KK = 1
   GO TO 40
70 CONTINUE
   NORMCON(INDEX,NORM2) = FVALUE
   INDEX = INDEX + 1
   ISTATE1(1) = ISTATE1(1) + 1
   IF(ISTATE1(1) .LE. NUSERS(1)) GO TO 20
80 CONTINUE
   ISTATE1(K) = ISTATE1(K) + 1
   IF(ISTATE1(K) .GT. NUSERS(K)) GO TO 100
   L = K - 1
   DO 90 J = 1, L
   ISTATE1(J) = 0
90 CONTINUE
   GO TO 20
```

```
100 CONTINUE
    K = K + 1
    IF(K .LE. ITYPES) GO TO 80
    CALL WRITMS(3,NORMCON(1,NORM2),ISTATES,MAXNODE+I)
    ITEMP = NORM1
    NORM1 = NORM2
    NORM2 = ITEMP
110 CONTINUE
    RETURN
    END
```

```
      SUBROUTINE MARGIN(FNT,CONNORM,PROBMAR,PROBCUM,
     1NUSERS,ISTATE1,ISTATE2,ISKIP,MAXNODE,MAXSTAT,
     2MAXTYPE,NODES,ITYPES,INDEX3,ISIZE3,ISTATES)
      DIMENSION FNT(MAXSTAT), CONNORM(MAXSTAT,2),
     1PROBMAR(MAXSTAT), PROBCUM(MAXSTAT),
     2NUSERS(MAXTYPE), ISTATE1(MAXTYPE), ISTATE2(MAXTYPE),
     3ISKIP(MAXTYPE), INDEX3(ISIZE3)
      INTEGER FINDEX
CCC
CCC   THIS SUBROUTINE CALCULATES THE PROBABILITY THAT A GIVEN
CCC   NODE IS IN A GIVEN STATE.  THIS PROBABILITY IS CALCULATED
CCC   FOR ALL POSSIBLE STATES FOR EACH NODE.
CCC
      NORM1 = 1
      NORM2 = 2
      CALL READMS(3,CONNORM(1,NORM1),ISTATES,MAXNODE+NODES)
      CONSTAN = 1. / CONNORM(ISTATES,NORM1)
      FINDEX = 1
      LINDEX = ISTATES
      PPP = 0.
      CALL READMS(3,FNT(1),ISTATES,NODES)
      CALL READMS(3,CONNORM(1,NORM2),ISTATES,MAXNODE+NODES-1)
   20 CONTINUE
      PROBMAR(FINDEX) = CONSTAN * CONNORM(LINDEX,NORM2) *
     1FNT(FINDEX)
      PROBCUM(FINDEX) = CONSTAN * CONNORM(FINDEX,NORM2) *
     1FNT(LINDEX)
      PPP = PPP + PROBMAR(FINDEX)
      FINDEX = FINDEX + 1
      LINDEX = LINDEX - 1
      IF(FINDEX .LE. ISTATES) GO TO 20
      IF(NODES .LE. 2) GO TO 150
      NCYCLES = NODES - 2
      CALL WRITMS(3,PROBMAR(1),ISTATES,MAXNODE*2+NODES)
      DO 140 I = 1, NCYCLES
      NSUB = NODES - I
```

```
      CALL READMS(3,FNT(1),ISTATES,NSUB)
      ITEMP = NORM1
      NORM1 = NORM2
      NORM2 = ITEMP
      CALL READMS(3,CONNORM(1,NORM2),ISTATES,MAXNODE+NSUB-1)
      INDEX = 1
      DO 30 J = 1, ITYPES
      ISTATE1(J) = 0
   30 CONTINUE
      K = 2
      PPP = 0.
   40 CONTINUE
      DO 50 J = 1, ITYPES
      ISTATE2(J) = ISTATE1(J)
   50 CONTINUE
      FINDEX = 1
      LINDEX = INDEX
      KK = 1
      ITEMP = 1
      DO 60 J = 1, ITYPES
      ISKIP(J) = ITEMP * ISTATE1(J)
      ITEMP = ITEMP * (NUSERS(J) + 1)
   60 CONTINUE
      ISKIP(1) = ISKIP(1) + 1
      FVALUE1 = 0.
      FVALUE2 = 0.
   70 CONTINUE
      FVALUE1 = FVALUE1 + PROBCUM(LINDEX) *
     1((CONNORM(FINDEX,NORM2) * FNT(INDEX)) /
     2CONNORM(LINDEX,NORM1))
      FVALUE2 = FVALUE2 + PROBCUM(LINDEX) *
     1((CONNORM(INDEX,NORM2) * FNT(FINDEX)) /
     2CONNORM(LINDEX,NORM1))
      IF(LINDEX .EQ. ISTATES) GO TO 100
      ISTATE2(KK) = ISTATE2(KK) + 1
      IF(ISTATE2(KK) .GT. NUSERS(KK)) GO TO 80
```

```
      FINDEX = FINDEX + 1
      LINDEX = LINDEX + 1
      GO TO 70
   80 CONTINUE
      FINDEX = FINDEX + ISKIP(KK)
      LINDEX = LINDEX + ISKIP(KK)
      KK = KK + 1
      ISTATE2(KK) = ISTATE2(KK) + 1
      IF(ISTATE2(KK) .GT. NUSERS(KK)) GO TO 80
      L = KK - 1
      DO 90 J = 1, L
      ISTATE2(J) = ISTATE1(J)
   90 CONTINUE
      KK = 1
      GO TO 70
  100 CONTINUE
      PROBMAR(INDEX) = FVALUE1
      PPP = PPP + FVALUE1
      PROBCUM(INDEX) = FVALUE2
      INDEX = INDEX + 1
      ISTATE1(1) = ISTATE1(1) + 1
      IF(ISTATE1(1) .LE. NUSERS(1)) GO TO 40
  110 CONTINUE
      ISTATE1(K) = ISTATE1(K) + 1
      IF(ISTATE1(K) .GT. NUSERS(K)) GO TO 130
      L = K - 1
      DO 120 J = 1, L
      ISTATE1(J) = 0
  120 CONTINUE
      K = 2
      GO TO 40
  130 CONTINUE
      K = K + 1
      IF(K .LE. ITYPES) GO TO 110
      CALL WRITMS(3,PROBMAR(1),ISTATES,MAXNODE*2+NSUB)
  140 CONTINUE
```

```
150 CONTINUE
    PPP = 0.
    DO 160 I = 1, ISTATES
    PPP = PPP + PROBCUM(I)
160 CONTINUE
    CALL WRITMS(3,PROBCUM(1),ISTATES,MAXNODE*2+1)
    RETURN
    END
```

```
      SUBROUTINE EXPECT(PROBMAR,PROBTOT,PROBPAR,NUSERS,ISTATE,
     1NODETYP,EXPVAL,DEPART,UTIL,SERVICE,E,
     1MAXNODE,MAXUSE1,MAXTYPE,NODES,ITYPES,IUSERS,MAXSTAT,INDEX3,
     2ISIZE3,ISTATES,IDEP,DEP,MAXUSER)
      DIMENSION PROBMAR(MAXSTAT), PROBTOT(MAXNODE,MAXUSE1),
     1PROBPAR(MAXNODE,MAXTYPE,MAXUSE1), NUSERS(MAXTYPE),
     2ISTATE(MAXTYPE), NODETYP(MAXNODE), EXPVAL(MAXNODE,MAXTYPE),
     3DEPART(MAXNODE,MAXTYPE), UTIL(MAXNODE,MAXTYPE),
     4SERVICE(MAXNODE,MAXTYPE), INDEX3(ISIZE3), E(MAXNODE,MAXTYPE),
     5IDEP(MAXNODE), DEP(MAXNODE,MAXTYPE,MAXUSER)
CCC
CCC   THIS SUBROUTINE CALCULATES VARIOUS PERFORMANCE MEASURES
CCC   FOR THE TIME-SHARING SYSTEM BEING MODELED.  THESE
CCC   MEASURES INCLUDE THE MARGINAL QUEUE LENGTH DISTRIBUTION
CCC   FOR EACH TYPE OF USER AT EACH NODE, THE UTILIZATION OF
CCC   THE NODES BY EACH TYPE OF USER, AND THE RESPONSE TIME
CCC   FOR EACH TYPE OF USER.
CCC
      WRITE(6,1)
    1 FORMAT(////10X,"*****THE OUTPUT OF THE MODEL FOLLOWS*****")
      DO 70 I = 1, NODES
      CALL READMS(3,PROBMAR(1),ISTATES,MAXNODE*2+I)
      K = 1
      INDEX = 0
      NTOTAL = 0
      DO 20 J = 1, ITYPES
      ISTATE(J) = 1
   20 CONTINUE
   40 CONTINUE
      NTOTAL = NTOTAL + 1
      INDEX = INDEX + 1
      PROB = PROBMAR(INDEX)
      PROBTOT(I,NTOTAL) = PROBTOT(I,NTOTAL) + PROB
      DO 50 J = 1, ITYPES
      JJ = ISTATE(J)
      PROBPAR(I,J,JJ) = PROBPAR(I,J,JJ) + PROB
```

```
      IF(JJ .EQ. 1) GO TO 50
      PARTIAL = PROB * (JJ - 1)
      IF(NODETYP(I) .EQ. 3) GO TO 45
      PARTIAL = PARTIAL / (NTOTAL - 1)
      UTIL(I,J) = UTIL(I,J) + PARTIAL
   45 CONTINUE
      PARTIAL = PARTIAL * (1. / SERVICE(I,J))
      DEPART(I,J) = DEPART(I,J) + PARTIAL
   50 CONTINUE
      IF(INDEX .EQ. ISTATES) GO TO 70
   60 CONTINUE
      ISTATE(K) = ISTATE(K) + 1
      IF(ISTATE(K) .GT. NUSERS(K) + 1) GO TO 65
      K = 1
      GO TO 40
   65 CONTINUE
      ISTATE(K) = 1
      NTOTAL = NTOTAL - NUSERS(K)
      K = K + 1
      IF(K .LE. ITYPES) GO TO 60
   70 CONTINUE
      IUSERS1 = IUSERS + 1
      WRITE(6,74) NODES
   74 FORMAT(////10X,"THE MARGINAL QUEUE LENGTH PROBABILITIES ",
     1"AT EQUILIBRIUM"///10X,"QUEUE LENGTH",5X,"QUEUE LENGTH ",
     2"PROBABILITY FOR NODES 1 THROUGH ",I3)
      DO 75 I = 1, NODES
      EXPVAL(I,1) = 0.
   75 CONTINUE
      DO 110 J = 1, IUSERS1
      JJ = J - 1
      DO 90 I = 1, NODES
      EXPVAL(I,1) = EXPVAL(I,1) + JJ * PROBTOT(I,J)
   90 CONTINUE
      WRITE(6,80) JJ, (PROBTOT(I,J),I=1,NODES)
   80 FORMAT(/14X,I3,7X,10F10.5,10(/24X,10F10.5))
```

```
110 CONTINUE
    WRITE(6,111)
111 FORMAT(////10X,"THE EXPECTED NUMBER OF TASKS AT EACH NODE"///10X,
   1"NODE",10X,"EXPECTED VALUE")
    DO 113 I = 1, NODES
    WRITE(6,112) I, EXPVAL(I,1)
112 FORMAT(/10X,I4,10X,F10.5)
113 CONTINUE
    WRITE(6,114)
114 FORMAT(/////10X,"THE MARGINAL QUEUE LENGTH PROBABILITIES ",
   1"AT EQUILIBRIUM CLASSIFIED BY TASK TYPE")
    DO 160 J = 1, ITYPES
    WRITE(6,115) J, NODES
115 FORMAT(////10X,"THE MARGINAL QUEUE LENGTH PROBABILITIES ",
   1"AT EQUILIBRIUM FOR TASK TYPE",I3///10X,"QUEUE LENGTH",5X,
   2"QUEUE LENGTH PROBABILITIES FOR NODES 1 THROUGH ",I3)
    IUSERS1 = NUSERS(J) + 1
    DO 116 I = 1, NODES
    EXPVAL(I,J) = 0.
116 CONTINUE
    DO 150 K = 1, IUSERS1
    KK = K - 1
    DO 120 I = 1, NODES
    EXPVAL(I,J) = EXPVAL(I,J) + KK * PROBPAR(I,J,K)
120 CONTINUE
    WRITE(6,130) KK, (PROBPAR(I,J,K),I=1,NODES)
130 FORMAT(/14X,I3,7X,10F10.5,10(/24X,10F10.5))
150 CONTINUE
160 CONTINUE
    WRITE(6,162) ITYPES
162 FORMAT(////10X,"THE EXPECTED NUMBER OF TASKS AT EACH NODE ",
   1"CLASSIFIED BY TASK TYPE"///10X,"NODE",10X,"EXPECTED NUMBER ",
   2"OF TASK TYPES 1 THROUGH ",I3,"FOUND AT NODE")
    DO 166 I = 1, NODES
    WRITE(6,164) I, (EXPVAL(I,J),J=1,ITYPES)
164 FORMAT(/10X,I4,10X,10F10.5,10(/24X,10F10.5))
```

```
166 CONTINUE
    WRITE(6,170) ITYPES
170 FORMAT(////10X,"NODE UTILIZATION BY EACH TASK TYPE"///10X,
   1"NODE",10X,"UTILIZATION OF NODE BY TASK TYPE 1 THROUGH",I3)
    DO 180 I = 1, NODES
    IF(NODETYP(I) .EQ. 3) GO TO 180
    WRITE(6,175) I, (UTIL(I,J),J=1,ITYPES)
175 FORMAT(/10X,I4,10X,10F10.5,10(/24X,10F10.5))
180 CONTINUE
    DO 190 I = 1, NODES
    DO 185 J = 2, ITYPES
    UTIL(I,1) = UTIL(I,1) + UTIL(I,J)
185 CONTINUE
190 CONTINUE
    WRITE(6,195)
195 FORMAT(////10X,"TOTAL NODE UTILIZATION"///10X,
   1"NODE",10X,"UTILIZATION")
    DO 200 I = 1, NODES
    IF(NODETYP(I) .EQ. 3) GO TO 200
    WRITE(6,198) I, UTIL(I,1)
198 FORMAT(/10X,I4,10X,F10.5)
200 CONTINUE
    WRITE(6,210) ITYPES
210 FORMAT(////10X,"THE MEAN TIME EACH TASK TYPE SPENDS ",
   1"AT THE VARIOUS NODES FOR EACH INTERACTION"///10X,"NODE",10X,
   2"MEAN TIME FOR TASK TYPE 1 THROUGH",I3)
    DO 230 I = 1, NODES
    IF(IDEP(I) .EQ. 1) GO TO 205
    WRITE(6,204)
204 FORMAT(/10X,"THE MODEL DOES NOT CALCULATE THIS TIME FOR "/10X,
   1"NODES WITH STATE DEPENDENT SERVICE RATES.")
    GO TO 230
205 CONTINUE
    DO 220 J = 1, ITYPES
    UTIL(I,J) = 0.
    IF((DEPART(I,J) .EQ. 0.) .OR. (E(I,J) .EQ. 0.)) GO TO 220
```

```
      UTIL(I,J) = EXPVAL(I,J) / DEPART(I,J) * E(I,J)
  220 CONTINUE
      WRITE(6,225) I, (UTIL(I,J),J=1,ITYPES)
  225 FORMAT(/10X,I4,10X,10F10.5,10(/24X,10F10.5))
  230 CONTINUE
      WRITE(6,240)
  240 FORMAT(/////10X,"THE MEAN RESPONSE TIME FOR THE VARIOUS ",
     1"TASK TYPES")
      DO 280 I = 1, NODES
      IF(NODETYP(I) .NE. 3) GO TO 280
      WRITE(6,250) I
  250 FORMAT(////10X,"THE MEAN RESPONSE TIME FOR TASKS ",
     1"REPRESENTED AT NODE ",I3//10X,"TASK TYPE",10X,
     2"RESPONSE TIME")
      DO 270 J = 1, ITYPES
      TIME = 0.
      IF(DEPART(I,J) .EQ. 0.) GO TO 255
      TIME = (NUSERS(J) - EXPVAL(I,J)) / DEPART(I,J)
  255 CONTINUE
      WRITE(6,260) J, TIME
  260 FORMAT(/13X,I3,14X,F10.5)
  270 CONTINUE
  280 CONTINUE
      RETURN
      END
```

Appendix B

Sample Program Output

Appendix B

Sample Program Output

This appendix contains a sample run for the closed queueing network model presented in Appendix A. The input data for this test case is listed below. The output of the program begins on the following page.

```
         1111111111222222222233333333334444444444555555555556
1234567890123456789012345678901234567890123456789012345678901234567890
```

```
    5        2
   10       10
    3        1      300.      300.
    1        1        3.        3.
    2        1    45.79     89.31
    2        1    42.10     82.09
    1        1      1.3       1.3
   0.       1.       0.        0.        0.
   0.       0.    .4789     .5211        0.
   0.       0.       0.        0.        1.
   0.       0.       0.        0.        1.
   .1       0.       0.        0.        .9
   0.       1.       0.        0.        0.
   0.       0.    .4789     .5211        0.
   0.       0.       0.        0.        1.
   0.       0.       0.        0.        1.
 .3334      0.       0.        0.     .6666
```

THIS IS A CLOSED QUEUEING NETWORK MODEL FOR TIME-SHARING COMPUTER SYSTEMS.

*****THE INPUT TO THE MODEL FOLLOWS*****

THE NUMBER OF NODES REPRESENTED IN THE MODEL = 5

THE NUMBER OF TASK TYPES REPRESENTED IN THE MODEL = 2

THE DISTRIBUTION OF TASKS IN THE SYSTEM BY TASK TYPE

TASK TYPE	NUMBER OF TASKS
1	10
2	10
TOTAL	20

NODE CHARACTERISTICS

NODE NAME	NODE TYPE	SERVICE RATE	NODE SERVICE RATES FOR TASK TYPES 1 THROUGH 2	
1	3	1	300.00000	300.00000
2	1	1	3.00000	3.00000
3	2	1	45.79000	89.31000
4	2	1	42.10000	82.09000
5	1	1	1.30000	1.30000

TRANSITION PROBABILITIES DESCRIBING MOVEMENT OF TASKS AMONG THE MODEL'S NODES

TRANSITION PROBABILITIES FOR TASK TYPE 1

DEPARTURE NODE	PROBABILITY OF TASK MOVEMENT FROM DEPARTURE NODE TO NODES 1 THROUGH 5				
1	0.00000	1.00000	0.00000	0.00000	0.00000
2	0.00000	0.00000	.47890	.52110	0.00000
3	0.00000	0.00000	0.00000	0.00000	1.00000
4	0.00000	0.00000	0.00000	0.00000	1.00000
5	.10000	0.00000	0.00000	0.00000	.90000

TRANSITION PROBABILITIES FOR TASK TYPE 2

DEPARTURE NODE	PROBABILITY OF TASK MOVEMENT FROM DEPARTURE NODE TO NODES 1 THROUGH 5				
1	0.00000	1.00000	0.00000	0.00000	0.00000
2	0.00000	0.00000	.47890	.52110	0.00000
3	0.00000	0.00000	0.00000	0.00000	1.00000
4	0.00000	0.00000	0.00000	0.00000	1.00000
5	.33340	0.00000	0.00000	0.00000	.66660

*****THE OUTPUT OF THE MODEL FOLLOWS*****

THE MARGINAL QUEUE LENGTH PROBABILITIES AT EQUILIBRIUM

QUEUE LENGTH	QUEUE LENGTH PROBABILITY FOR NODES 1 THROUGH 5				
0	.00009	.91127	.09287	.09263	.72841
1	.00085	.08057	.09257	.09248	.20081
2	.00411	.00709	.09228	.09213	.05314
3	.01318	.00061	.09155	.09145	.01345
4	.03139	.00005	.09024	.09019	.00325
5	.05927	.00000	.08801	.08800	.00074
6	.09233	.00000	.08445	.08449	.00015
7	.12201	.00000	.07915	.07922	.00003
8	.13948	.00000	.07180	.07189	.00001
9	.14002	.00000	.06238	.06249	.00000
10	.12483	.00000	.05130	.05141	.00000
11	.09968	.00000	.03941	.03951	.00000
12	.07175	.00000	.02788	.02796	.00000

13	.04675	.00000	.01786	.01792	.00000
14	.02763	.00000	.01016	.01020	.00000
15	.01481	.00000	.00502	.00504	.00000
16	.00716	.00000	.00208	.00209	.00000
17	.00308	.00000	.00070	.00070	.00000
18	.00115	.00000	.00018	.00018	.00000
19	.00035	.00000	.00003	.00003	.00000
20	.00007	.00000	.00000	.00000	.00000

THE EXPECTED NUMBER OF TASKS AT EACH NODE

NODE	EXPECTED VALUE
1	8.87232
2	.09722
3	5.32898
4	5.33547
5	.36542

THE MARGINAL QUEUE LENGTH PROBABILITIES AT EQUILIBRIUM CLASSIFIED BY TASK TYPE

THE MARGINAL QUEUE LENGTH PROBABILITIES AT EQUILIBRIUM FOR TASK TYPE 1

QUEUE LENGTH	QUEUE LENGTH PROBABILITIES FOR NODES 1 THROUGH 5				
0	.00193	.94642	.21208	.21162	.76499
1	.01429	.05092	.20319	.20297	.18369
2	.05068	.00254	.18418	.18417	.04103
3	.11406	.00012	.15331	.15344	.00842
4	.18148	.00000	.11360	.11379	.00156
5	.21495	.00000	.07256	.07273	.00026
6	.19381	.00000	.03855	.03866	.00004
7	.13302	.00000	.01627	.01633	.00000
8	.06757	.00000	.00510	.00512	.00000
9	.02353	.00000	.00105	.00106	.00000
10	.00448	.00000	.00011	.00011	.00000

THE MARGINAL QUEJE LENGTH PROBABILITIES AT EQUILIBRIUM FOR TASK TYPE 2

QUEUE LENGTH	QUEUE LENGTH PROBABILITIES FOR NODES 1 THROUGH 5				
0	.01767	.96059	.15166	.15133	.93742
1	.07645	.03784	.15065	.15043	.05879
2	.16067	.00142	.14795	.14786	.00357
3	.21772	.00005	.14176	.14178	.00021
4	.21272	.00000	.12963	.12973	.00001
5	.15852	.00000	.10959	.10974	.00000
6	.09273	.00000	.08212	.08227	.00000
7	.04300	.00000	.05162	.05174	.00000
8	.01562	.00000	.02520	.02527	.00000
9	.00422	.00000	.00840	.00843	.00000
10	.00070	.00000	.00142	.00143	.00000

THE EXPECTED NUMBER OF TASKS AT EACH NODE CLASSIFIED BY TASK TYPE

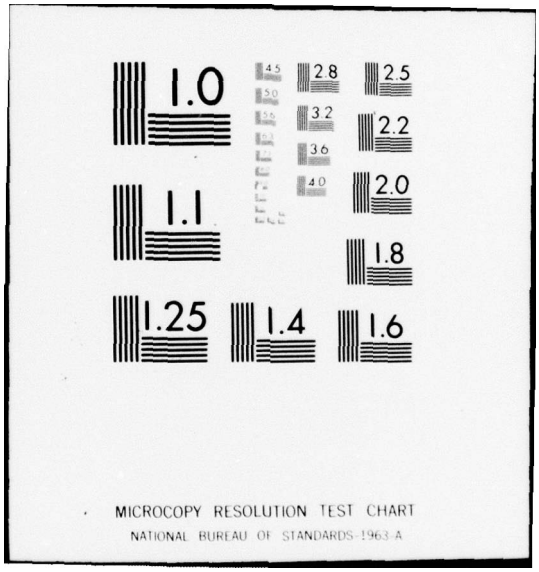
NODE	EXPECTED NUMBER OF TASK TYPES 1 THROUGH 2FOUND AT NODE	
1	5.15130	3.72152
2	.05637	.04084
3	2.24522	3.08376
4	2.24328	3.08718
5	.29882	.06650

AD-A039 719 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
QUEUEING NETWORK MODEL FOR PERFORMANCE EVALUATION OF THE DECSYS--ETC(U)
MAR 77 L E MCKENZIE

UNCLASSIFIED AFIT/GCS/EE/77-1 NL

3 OF 3
ADA039719

END
DATE FILMED
6-77


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

NODE UTILIZATION BY EACH TASK TYPE

NODE	UTILIZATION OF NODE BY TASK TYPE 1 THROUGH 2	
2	.05151	.03722
3	.37654	.53059
4	.37670	.53057
5	.22322	.04837

TOTAL NODE UTILIZATION

NODE	UTILIZATION
2	.08873
3	.90713
4	.90737
5	.27159

THE MEAN TIME EACH TASK TYPE SPENDS AT THE VARIOUS NODES FOR EACH INTERACTION

NODE	MEAN TIME FOR TASK TYPE 1 THROUGH 2	
1	300.00000	300.00000
2	3.28309	3.29241
3	130.75624	248.58238
4	130.93474	248.85836
5	17.40283	5.36849

THE MEAN RESPONSE TIME FOR THE VARIOUS TASK TYPES

THE MEAN RESPONSE TIME FOR TASKS REPRESENTED AT NODE 1

TASK TYPE	RESPONSE TIME
1	282.37590
2	506.10164

Appendix C

Users' Guide

Appendix C

Users' Guide

The program presented in Appendix A is a closed queueing network model for time-sharing computer systems. A description of the system being modeled is required as input into this program. The following is a description of the data and the format of the data that comprises the program input:

1. NODES --- the number of nodes in the model.

   ITYPES --- the number of user types in the model.

   INPUT CODE ---
```
READ(5,10) NODES, ITYPES
10 FORMAT(2I10)
```

2. NODETYP(1) --- the type of node I.
   1 --- FCFS node.
   2 --- Processor sharing node.
   3 --- Infinite server node.
   4 --- LCFS node.

   IDEP(I) --- state dependent service rate indicator for node I.
   1 --- The service rate for users at node I is not dependent on the state of the node.
   2 --- The service rate for users at node I is dependent on the total number of users at the node. This type of state dependent service rate is applicable to node types 1, 2, and 4.
   3 --- The service rate for users at node I is dependent on the number of users of each type (1 - ITYPES) at the node. This type of state dependent service rate is applicable to node types 2 and 4 only.

   SERVICE(I,J) --- The mean service time for a type J user at node I. J = 1-ITYPES.

```
INPUT CODE ---    READ(5,50) NODETYPE(I),
                  DEP(I), SERVICE (I,J),
                  J = 1, ITYPES
               50 FORMAT(2I10,10F10.5)
```

DEP(I,1,K) if IDEP(I) = 2 --- DEP(I,1,K) is the relative service rate of users at node I when there are K users present as compared to the rate when one user is present at the node.

```
INPUT CODE ---    READ(5,56) (DEP(I,1,J),
                  J = 1, MAXUSER)
               56 FORMAT(4)/8F10.5))
```

DEP(I,J,K)if IDEP(I) = 3 --- DEP(I,J,K) is the relative service rate of users at node I when there are K users of type J present as compared to the service rate when one user is present at the node.

```
INPUT CODE ---    DO 58 J = 1, ITYPES
                  READ(5,56) (DEP(I,J,K),
                  K = 1, MAXUSER)
               56 FORMAT(4(/8F10.5))
               58 CONTINUE
```

3. For each type of user in the system (1 - ITYPES):

PROB(J,I) --- The probability that a user of a given type will arrive at node I after departing node J. I = 1 - NODES. J = 1 - NODES.

```
INPUT CODE ---  DO 75 I = 1, NODES
                READ(5,70) (PROB(J,I),
                J = 1, NODES)
             70 FORMAT(8F10.5)
             75 CONTINUE
```

VITA

Leslie E. McKenzie, Jr. was born 21 June 1949 in Dillon, South Carolina. He graduated from Dillon High School, Dillon, South Carolina in 1967 and attended Clemson University from which he received a Bachelor of Science degree in mathematics in May 1971. Upon graduation, he received a commission in the USAF through the ROTC program. He was a computer systems analyst at SAC Hq, Offutt AFB, Nebraska until entering the School of Engineering, Air Force Institute of Technology, in September, 1975.

Permanent address: Route 3, Box 115
Dillon, South Carolina 29536

This thesis was typed by Bonnie L. Templeton

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER GCS/EE/77-1	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)* QUEUEING NETWORK MODEL FOR PERFORMANCE EVALUATION OF THE DECSYSTEM-10		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Leslie E. McKenzie, Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 2003-03-21
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Systems Group (AFAL/AAF-2) Air Force Avionics Laboratory Wright-Patterson AFB, Ohio 45431		12. REPORT DATE March, 1977 13. NUMBER OF PAGES 188
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)* Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT *(of this Report)*

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17

JERRAL F. GUESS, Captain, USAF
Director of Information

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

Computer performance evaluation
Analytic modeling
Closed queueing network models
Time-sharing computer systems

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

A closed queueing network model of the DECsystem-10 at the Air Force Avionics Laboratory (AFAL), Wright-Patterson AFB, Ohio was developed. This model was developed to provide personnel at the Avionics Laboratory with a computer performance evaluation (CPE) tool which they could use to improve the performance of their DECsystem-10.

The hardware configuration and operating system features

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

of the DECsystem-10 are discussed in this report. Features which affect performance are emphasized. Then, a review of computer peformance evaluation and the CPE tools and techniques currently available is presented. Of these tools and techniques, simulation and analytic modeling are investigated as the two major approaches to computer system modeling.

A closed queueing network model which can be used to evaluate performance of the DECsystem-10 is described. This model can represent different types of time-sharing users and system resources with general service time distributions. The model can also represent several different queueing disciplines for system resources.

Validation experiments indicated that the closed queueing network model described in this report is a viable CPE tool for the DECsystem-10. A possible improvement to the model would be the inclusion of the effect that swapping has on performance measures in time-sharing computer systems.

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE *(and Subtitle)*		5. TYPE OF REPORT & PERIOD COVERED
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR*(s)*		8. CONTRACT OR GRANT NUMBER*(s)*
9. PERFORMING ORGANIZATION NAME AND ADDRESS		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)*		15. SECURITY CLASS. *(of this report)*
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT *(of this Report)*		
17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*		
20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*		

DD FORM 1 JAN 73 **1473** EDITION OF 1 NOV 65 IS OBSOLETE