THE UNIVERSITY OF MICHIGAN

SYSTEMS ENGINEERING LABORATORY

Department of Electrical Engineering
College of Engineering

SEL Technical Report No. 5

MARKOV MODELING AND ANALYSIS OF A MULTIPLE ACCESS COMPUTER SYSTEM

by

J. L. Smith
J. H. Jackson

March, 1966

under contract with:

ROME AIR DEVELOPMENT CENTER
Research and Technology Division
Air Force Systems Command
Contract No. AF 30(602)-3953
Griffiss Air Force Base, New York

ACKNOWLEDGEMENT

The authors are indebted to Mr. V. L. Wallace for his valuable suggestions on modeling and the use of RQA.

TABLE OF CONTENTS

TABLE OF CONTENTS (Cont.)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES (Cont.)

ABSTRACT

Information flow and processor usage in a multiple access computer system is described by several Markovian queueing models. The performance of the system under various loading conditions is analyzed using a computer program designed to evaluate the equilibrium probability distributions of large scale Markov processes.

INTRODUCTION

A numerical technique for determining the stationary distribution of large-scale Markov processes has recently been developed at this laboratory.[1] The technique is used in this report to analyze Markov models of a typical on-line multiple-access command and control computer system.[2] The models incorporate the important queueing phenomena in the information transfers among the high speed memory, the disc file and the user communication consoles, and in the multiprogrammed operation of the central computer. Performance parameters for the system are defined and used to present the results of the analysis.

A model of the disc channel component of the system is also analyzed to determine how the channel would perform under different loading conditions with single or multiple block transfer capabilities.

The operation of the computer system is described in Section 1 and the aims of the analysis in Section 2. Section 3 is a brief survey of the modeling techniques and approximations used. Sections 4, 5, 6 and 7 describe the actual models and results. Section 8 gives conclusions and suggests some further work in analysis of this type of computer system.

1. System Description

This section describes the computer system in sufficient detail for understanding of the modeling process. For further detail the reader should consult Refs. 2 and 3.

1. 1. The On-Line Computer System

The major hardware sections of the on-line computer system are shown in Fig. 1. A central computer, associated with a large core storage module, executes all user task programs in the time-shared mode. The hardware provides storage protection and I/O instruction traps so that the central computer can be multi-programmed. Bulk storage for programs and data files is provided by a large capacity disc memory unit. A disc file control unit (DFCU), containing a core-storage buffer and the necessary control logic, controls information transfers between the computer and the disc memory unit. The query-response communication consoles (Q-RCC) have several facilities – type keyboard, program selection keys, CRT message display, message storage – for the operation to use in communicating on-line with his program or the control program. The console I/O buffer (CIOB) contains a core-storage buffer and control logic for communication between the consoles and the computer. The magnetic tape and paper tape units provide alternative media for storing and for loading new programs and data into the system. The I/O channels operate on an interrupt basis independently of the central processor, with the main core memory access cycles being shared.

The storage protection features of the computer are such that the

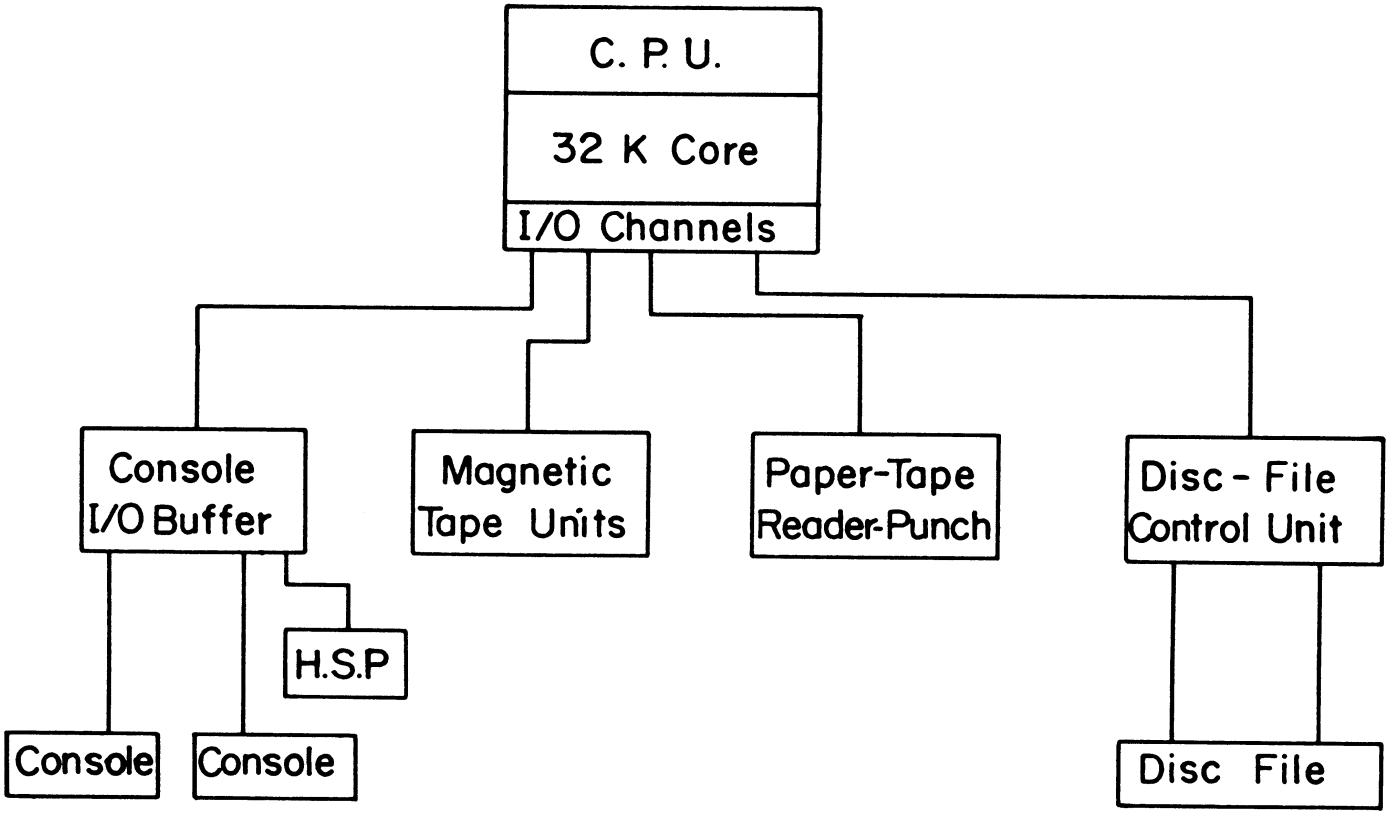

Fig. 1. An on-line multiple access computer system (the single thread system).

core store may be completely used by one program or else divided into two, four, or eight subsections (interlaces). Different programs can reside in separate interlaces, some of which are in general used by the executive control program (ECP). A user program executing within an interlace can transfer control to another interlace only through modification of registers by the ECP. Each interlace is subdivided into eight blocks, and transfer of control outside of each block requires register modifications. All I/O operations are executed under control of the ECP.

Under one executive control program modeled here (ECP-1), the core is always operated in the eight-interlace mode. ECP-1 will be allocated the lower 1024 locations of memory (constituting a small fraction of each interlace), along with a variable number of interlaces, and it will have the ability to vacate all except the 1024 low-core locations. However, for normal operation it will occupy several interlaces, with its less frequently executed sections being overlaid from the disc file.

Console operators will be concerned with special tasks which require computer assistance for information storage and retrieval, report generation, and particular computations. As users of the system they will request the execution of sequences of programs by operating select buttons. Each sequence will consist of standard task programs arranged in order and designed to provide automatic control, as well as the processing requirements, for the operators' tasks of planning, analyzing, and reporting.

Task programs and data files will be kept in the disc memory unit.

Upon a request for a particular sequence, the programs will be called, in the desired order, to assigned interlace positions. During execution the task programs will generate a series of query-response communications with the console operator, and the responses will provide branching information for the sequences. Other necessary information and utility programs will be called from the disc memory unit.

1. 2. The Disc-File Channel

A schematic diagram of the disc-file channel (DFCU) is shown in Fig. 2. It is designed to control information transfers among three computers and three disc-memory units, and also disc-file searches. Information is transferred in 512-word blocks, in two stages. After the first stage the block is contained in a 512-word sector of the DFCU core store. The second stage is to transfer the block either to the disc-memory unit or to the computer core. The DFCU has seven buffer sectors available and is capable of simultaneously controlling three data transfers between buffer sectors and either computers or disc-memory units. The limitation is that only two transfers involving any one disc-memory unit or any one computer can be in operation at one time.

The computer system described in Section 1. 1. has only one computer and one disc-memory unit. Consequently this configuration would make possible a higher information-transfer rate per computer than when three computers are competing for the channel facilities. However, the flexibility in the ECP needed for executing more than one block transfer simultaneously would have to be implemented. ECP-1,

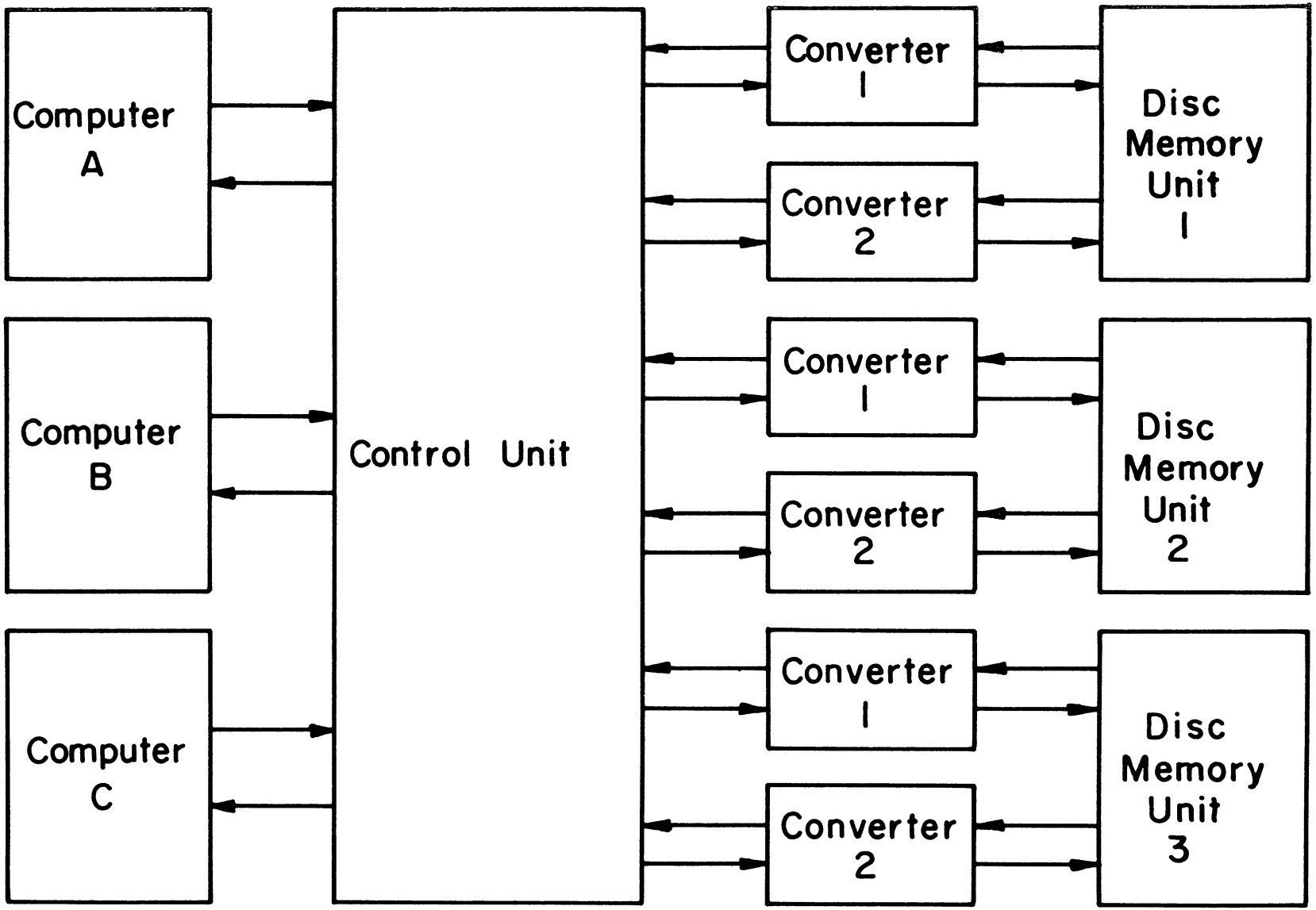

Fig. 2. Disc-file channel.

referred to in Section 1. 1, will execute only one block transfer at a time.

Information transfers are initiated from a computer by transmitting an external function to the DFCU. The transfer operation is then carried out independent of the computer except when core memory access cycles must be shared between the computer and the I/O channel during one stage of the transfer. Once the DFCU has acted on the external function, the transfer time is determined by the positioning and latency time of the disc arm plus the time taken for the two stages of the block transfer.

A disc-file search can also be requested. This involves positioning followed by a maximum of eight disc revolutions during which the search key is sought in all tracks of a given position. Since only the address of the block containing the key is returned to the computer, the transfer time is negligible.

The DFCU will attempt to execute three transfer requests concurrently when required. Since each disc-memory unit has only two positioning devices with their associated logic units only two operations involving any one disc-memory unit can be executed concurrently. Therefore, as requests are treated on a first-come/first-served basis, conflicts could reduce the number of concurrent operations to two.

2. Aims of Analysis

2. 1. General Aims

For the proposed hardware and software configurations of the given system we wish to determine:

(1) System capacity

(a) Response time to operator

(b) Program throughput

(c) Information transfer capacity

(2) Bottlenecks in information flow and processing

(3) Advantages to be obtained from possible hardware and software alternatives to the present system

(4) Scheduling and priority algorithms which will optimize system performance criteria

2. 2. Analysis of the On-Line Computer System

The on-line computer system, hereafter called the single thread system, with its associated execution control program, ECP-1, is a pilot version of larger multi-computer systems. Programs run on this system will be essentially the same in the larger systems, as will be the mode of accessing programs and data from the disc memory units and the mode of communication among the consoles, user programs, and control programs. Therefore results from this analysis will be important as a guide to the performance of larger systems.

In multiple access operation a necessary feature is that those programs which require only a moderate amount of processor* time do not experience long waiting times so that the operator encounters no serious delay

*Here processor is used in general sense to mean CPU, I/O channel, etc.

in his work outside the computer. Therefore an important result to be obtained from the model solutions is a measure of the number of consoles (operators) that the system can satisfactorily serve in parallel. Since at this stage is difficult to define "serious delay" results must be presented on a useful comparative scale.

To obtain any performance measure it is essential to have considerable knowledge of the characteristics of the ECP and the task programs. Important characteristics concern CPU execution times, disc-channel usage, console communications, and operator response times when the program is dependent on the operator's supplying data. Because of the specific and repetitive nature of the tasks to be performed by operators, it seems that if modes of operation are fixed the programs will have reliable and therefore measurable statistics. In the absence of such measurements, the present analysis is carried out for various ranges, thought to be representative, of the important characteristics.

At the same time as the system's performance is measured, the use of major system components is measured as well. This is necessary in the case of the CPU and the disc channel, so that the effects of their processing capacities can be determined.

Because of the number of indexing and validity checking tables to be accessed during the loading of a task program, a design alternative has been suggested in which these tables would be stored in a device providing faster access (e.g., a fixed-head-per-track disc). Such an

alternative would considerably reduce the time of loading and of the loader's occupying the disc channel. To provide a measure of the gain the costlier equipment would give, we calculate herein the performance parameters for the loading times estimated for both cases.

Algorithms for scheduling the use of system processors are an important part of the ECP, for they may significantly affect the efficiency and capacity of the system. Since heavy use of the disc channel is expected, two possible disc-channel-scheduling algorithms are implemented in the models and the system's performance under each is compared.

One of the main problems to be resolved in any multi-programming system is the most desirable mode of sharing high-speed storage between user and executive programs. By appropriately modifying the amount of disc I/O traffic generated by a program according to the amount of high-speed storage space it receives, some comparisons are made.

2. 3. Analysis of the Disc Channel

As mentioned in Section 1, the disc channel is designed to serve up to three computers and three disc-memory units. When only one computer is using the channel, as in the single-thread system, it is possible for two data transfers for that computer to be in execution concurrently. Two input transfers or two output transfers between the core buffer of the DFCU and the computer cannot be carried out simultaneously, but the core-to-core transfer here is so much faster than the mechanical access used with the disc that in effect the channel has dual-transfer

capacity.

However, the I/O channel control of the computer is designed to handle only one input transfer and one output transfer at any time; so some additional software control is necessary to handle two transfers in parallel. The improvement in information transfer capacity for this increase in software complexity is measured by comparing the solutions obtained by application of two different models.

In the larger systems proposed, the disc channel will service two or three computers. It is also possible in these systems for the DFCU to execute two data transfers concurrently for one computer, provided that the additional software control is in effect. However, in the multi-computer systems the limitation that the DFCU can concurrently control only three transfer operations means that only one computer can have two transfers in execution at any time. Hence transfer rate per computer will not improve as much as in the single-computer system. Models for multi-computer use of the disc channel are analyzed to make this comparison.

3. Multi-Dimensional Continuous-Time Markov Queueing Models*

We now briefly present some of the techniques used in producing Markov queueing models of computer systems.

3.1. Continuous-Time Markov Distributions

A given program will use the various processors of the system for random intervals of time. These service intervals can be described by a probability law in the form of a continuous time distribution function $F_X(t) = P(X \leq t)$, which gives the probability that the random interval X is not greater than t. Some useful distributions which give the Markov property are now presented; all of them are derived from the basic distribution.

(a) The Negative Exponential Distribution

$$P(X \leq t) = 1 - e^{-\mu t}, \qquad t \geq 0 \tag{3.1}$$

---

* For a discussion of the use of such models and techniques for their analysis the reader is referred to Ref. 4. There it is shown that the limiting state probabilities, p_i (the probability of the system existing in state i under equilibrium conditions), can be found by solving a set of linear operations

$$\sum_{j=0}^{M} u_{ij} p_j = 0 \qquad i=0,1,\ldots,M$$

$$\sum_{j=0}^{M} p_j = 1$$

Unless the model has the Markov property (either because of the probability distributions involved or the through expansion of the state space) the linear equations cannot be formed.

The mean $\overline{X} = 1/\mu$. Such a distribution function, and its associated density function, are plotted in Fig. 3. For use in explaining the derived distributions we diagram schematically, in Fig. 4, a device with service time distributed according to Eq. 3.1.

(b) The Erlang Distribution

This distribution is related to the negative exponential distribution in the following manner. Consider the total service time required by a program from a particular processor to be composed of n phases. The phases must be executed in order, and one phase must be completed before the next begins. If the service time in each phase is distributed according to a negative exponential, the distribution of the complete service time gives a model which has the Markov property.* This model describes the general n-th order Erlang distribution; it is schematically represented in Fig. 5.

---

* The use of the Erlang distribution and other distributions derived from the exponential is discussed by Morse (Ref. 4), who makes the following important observation: "....only exponential facilities give rise to simple linear equations for detailed balance of transitions between states, independent of time. For other types of distributions we normally have to solve much more complicated sets of equations. Yet many operational situations correspond to service time distributions which are appreciably different from exponential .... a fair number of these can be simulated by a suitable choosen set of exponential facilities, with appropriate rules for transition. This is not to say that the service facility in question necessarily has the actual structure corresponding to the model which simulates its statistical behavior, all that is necessary for our analysis is that the model does simulate this behavior."

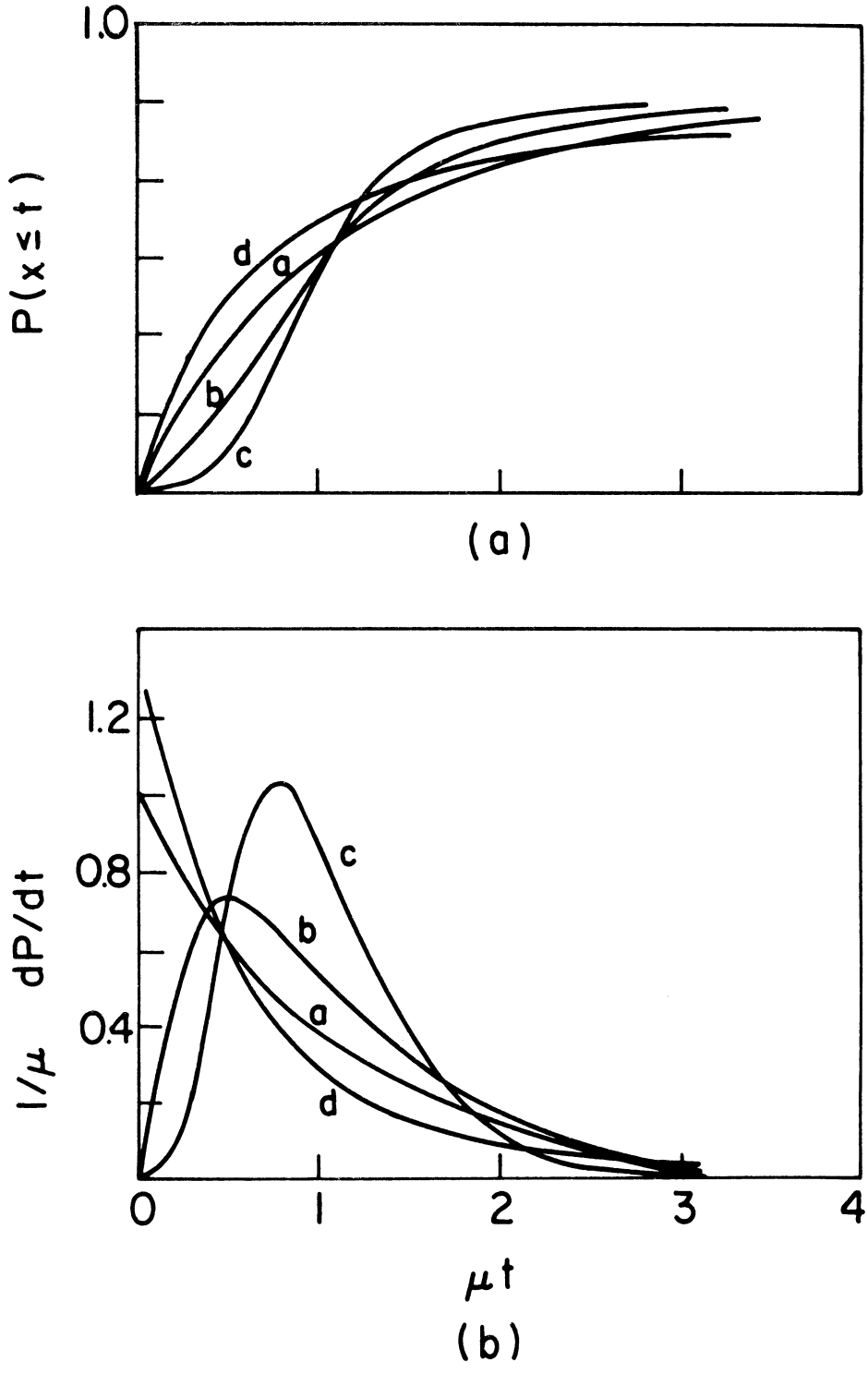

Fig. 3. Distribution functions $P(X \leq \mu t)$ and density functions $1/\mu$ dP/dt derived from the negative exponential.
The mean of each distribution is $1/\mu$.

(a) Negative exponential.
(b) Special second order Erlang.
(c) Special fifth order Erlang.
(d) Hyperexponential, $\mu_1 = \mu/2$, $\mu_2 = 2\mu$.

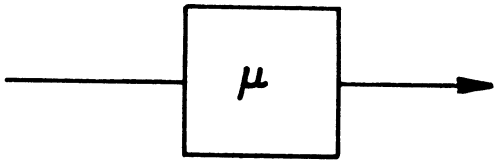

Fig. 4. A negative exponential service channel with mean service time $1/\mu$.

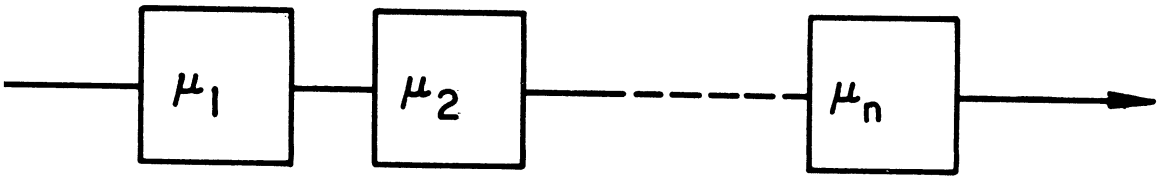

Fig. 5. An n-th order Erlang service channel.

If we constrain the parameters in Fig. 5 so that

$$\mu = \mu_1 = \mu_2 = ... = \mu_n \tag{3.2}$$

we obtain the special n-th order Erlang distribution, for which

$$P(X \leq t) = 1 - e^{-n\mu t} \sum_{k=0}^{n-1} \frac{(n\mu t)^k}{k!} \tag{3.3}$$

The mean $\overline{X} = 1/\mu$.

(c) The Hyperexponential Distribution

In this distribution, the random service time for a particular processor is determined by some one of n possible negative exponential distribution functions. There is a discrete probability function which independently determines, for each use of the processor, which distribution function is in effect. An n-th order hyperexponential service channel is represented schematically in Fig. 6. The quantities α_i define the discrete probability function so that

$$\sum_{i=1}^{n} \alpha_i = 1 \tag{3.4}$$

$$P(X \leq t) = 1 - \sum_{i=1}^{N} \alpha_i e^{-\mu_i t} \tag{3.5}$$

The mean $\overline{X} = \sum_{i=1}^{n} \alpha_i/\mu_i$. Again Morse's remarks, as quoted in the footnote, are relevant.

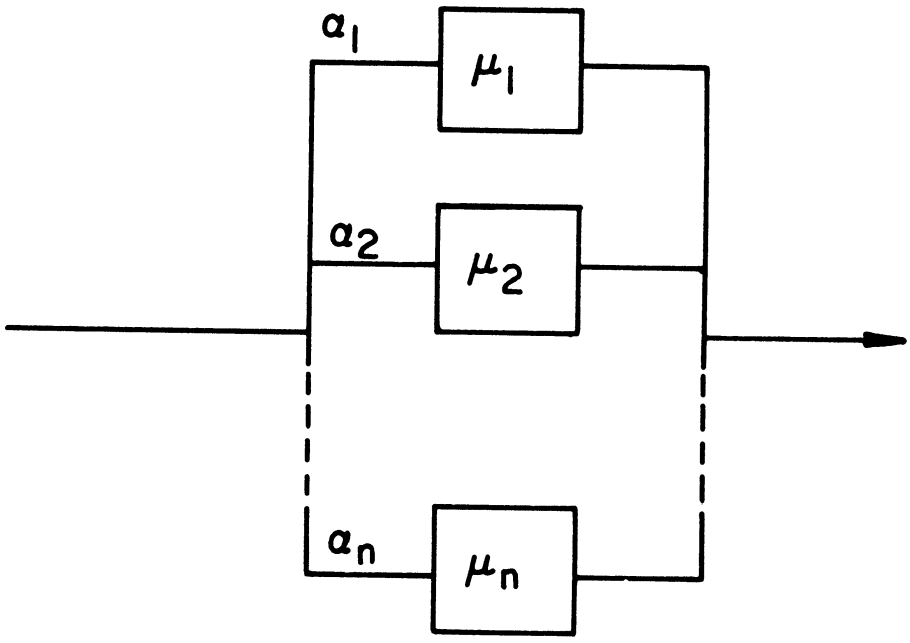

Fig. 6. An n-th order hyperexponential channel.

3. 2. Probability Branching

Figures 5 and 6 show how a set of service devices with negative exponential characteristics can be made to simulate a service with a completely different distribution. The probabilities associated with each of the service distributions determine the resultant distribution; however, none of the individual service distributions need correspond to that of any physical device.

The same technique may be used in modeling a system in which the service of only one of two or more different processors may be requested. The difference here is that we are now concerned with distributions which represent the service times of actual processors, and the use or probabilities to determine which processor is selected requires the assumption that each request for use of a processor is independent of all preceding requests.

For example, consider a program which has been in execution in an on-line environment and has stopped because it requires some intermediate set of data. The required data might be on a disc file or might need to be supplied by a console operator in response to a query generated by the program. Either way, the time required to supply the data will be random: the time required for use of the disc channel, or human response time. The time taken for either kind of data retrieval can be well modeled by some exponentially derived distributions, although one would expect the means of the distributions in the two cases to differ considerably. If the two types of request occur independently with fixed frequency in all programs, the use of the CPU, disc channel, and operator channel (including the operator)

could be very simply modeled as shown schematically in Fig. 7.

Here we have assumed that a single operator is using the system, so that no queues for use of any of the processors can form. The program executes in the CPU for a random length of time, exponentially distributed with mean $1/\mu_1$. Then, with probability p, it requests data from the disc file, so that the retrieval time is exponentially distributed with mean $1/\mu_2$; or with probability 1 - p, it requests information from the operator, whose response time is exponentially distributed with mean $1/\mu_3$. It is assumed that the program halts until the requested information is available, then executes for another random interval of time, and so on.

This very simplified example (in which loading of programs is completely ignored) illustrates the use of probability branching in a model to take into account random requests for processors in a system. The resultant model is a three state Markov process from which can be determined the steady-state probability that each processor of the system is in use. If state 1 is defined to be that the program is in execution, state 2 that the program is awaiting disc-file retrieval, and state 3 that the program is awaiting the console operator's response, the stationary probabilities p_i are given by Eq. 3.6.

$$\begin{bmatrix} -\mu_1 & \mu_2 & \mu_3 \\ p\mu_1 & -\mu_2 & 0 \\ (1-p)\mu_1 & 0 & -\mu_3 \end{bmatrix} \cdot \begin{bmatrix} p_1 \\ p_2 \\ p_3 \end{bmatrix} = \underline{0} \qquad (3.6)$$

The type of equations shown in Eq. 3.6 are efficiently solved by RQA[1] when

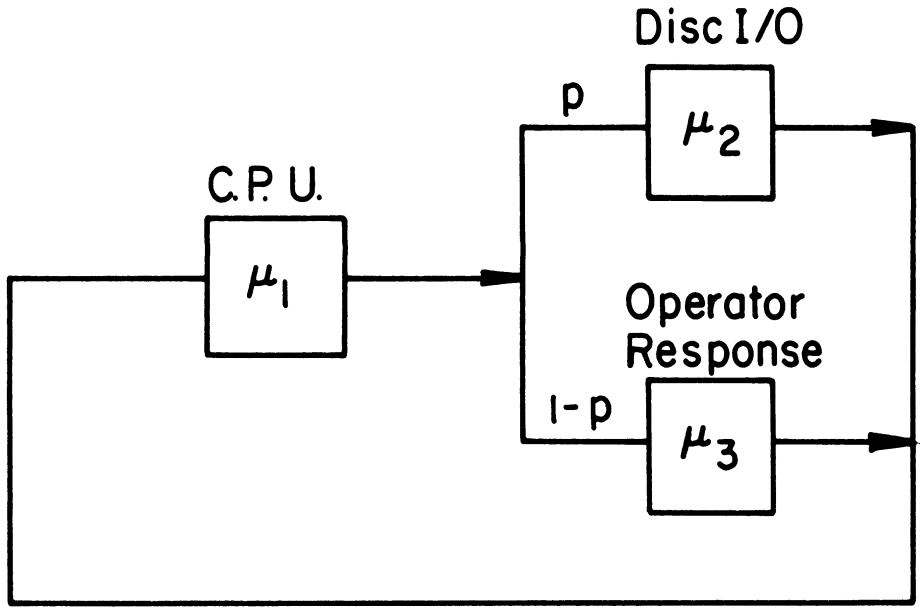

Fig. 7. A model of a simple on-line computer operation.

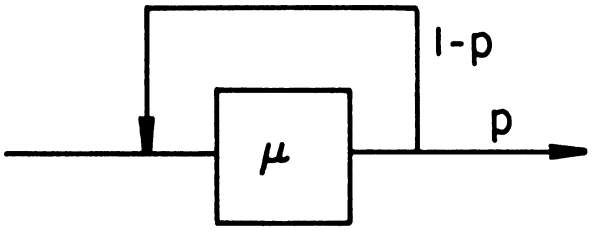

Fig. 8. Probability branching in a negative exponential service channel. The result is equivalent to a negative exponential service channel with mean service time $1/p\,\mu$.

there is a large number of states.

A special application of probability branching is illustrated in Fig. 8. With probability 1 - p, on completion of service the request being processed returns for another complete processing operation. If the service time for each processing operation has a negative exponential distribution with mean $1/\mu$, it can be shown that the total processing time taken by each request has a negative exponential distribution with mean $1/p\mu_1$. This modeling technique is useful in the following type of example. Suppose that a particular processing operation consists of a random number of identical phases—for example, loading of a random number of independently located blocks from a disc file. Suppose further that the completion of each phase generates a decision point; for example, the arrival of a request for a disc-file operation that takes precedence over block-loading will cause the block-loading to be interrupted, but only after the block being processed is completely loaded. The decision strategy involved here can be incorporated by using a model such as that diagrammed in Fig. 8. If the loading time in the example had been modeled by a single exponential distribution, there would have been no convenient method of incorporating regular decision points.

Inherent in the inclusion of the probablility branching is a discrete probability distribution on the number of times the processor is used in a single request. It is immediately evident that the mean of this distribution is 1/p. In fact, the distribution is geometric and has a variance of $(1 - p)/p^2$.

4. Models of the Single-Thread System

The basic queueing model used for analysis of the single-thread system is diagrammed in Fig. 9. The random service times associated with each operation are modeled as in Section 3. Any point in the model at which a queue for use of a service device may form is represented by a circle containing a number equal to the maximum allowable queue length. The direction of request flow is indicated. It is necessary to describe some service rules before the model is complete, and these, together with variations of this model, are given below.

4. 1. Service Rules

In all the models analyzed we consider that a fixed number N of operators are continually requesting operational sequences to be executed. One sequence is in execution for each operator at any time, and this sequence is completed before the next is begun. It is assumed that the particular interlace mode being used is sufficient to hold one task program of the current sequence requested by each operator. These programs queue for use of the CPU, and it is assumed that they execute for a random period of time (distributed according to a negative exponential with mean $1/\mu_1$) before they stop for one of the following three reasons:

(a) With probability p the task program has called for a data block, subroutine, or some other information block from the disc file and requires that this block be loaded before execution can continue. The time for accessing and transferring ring an information block

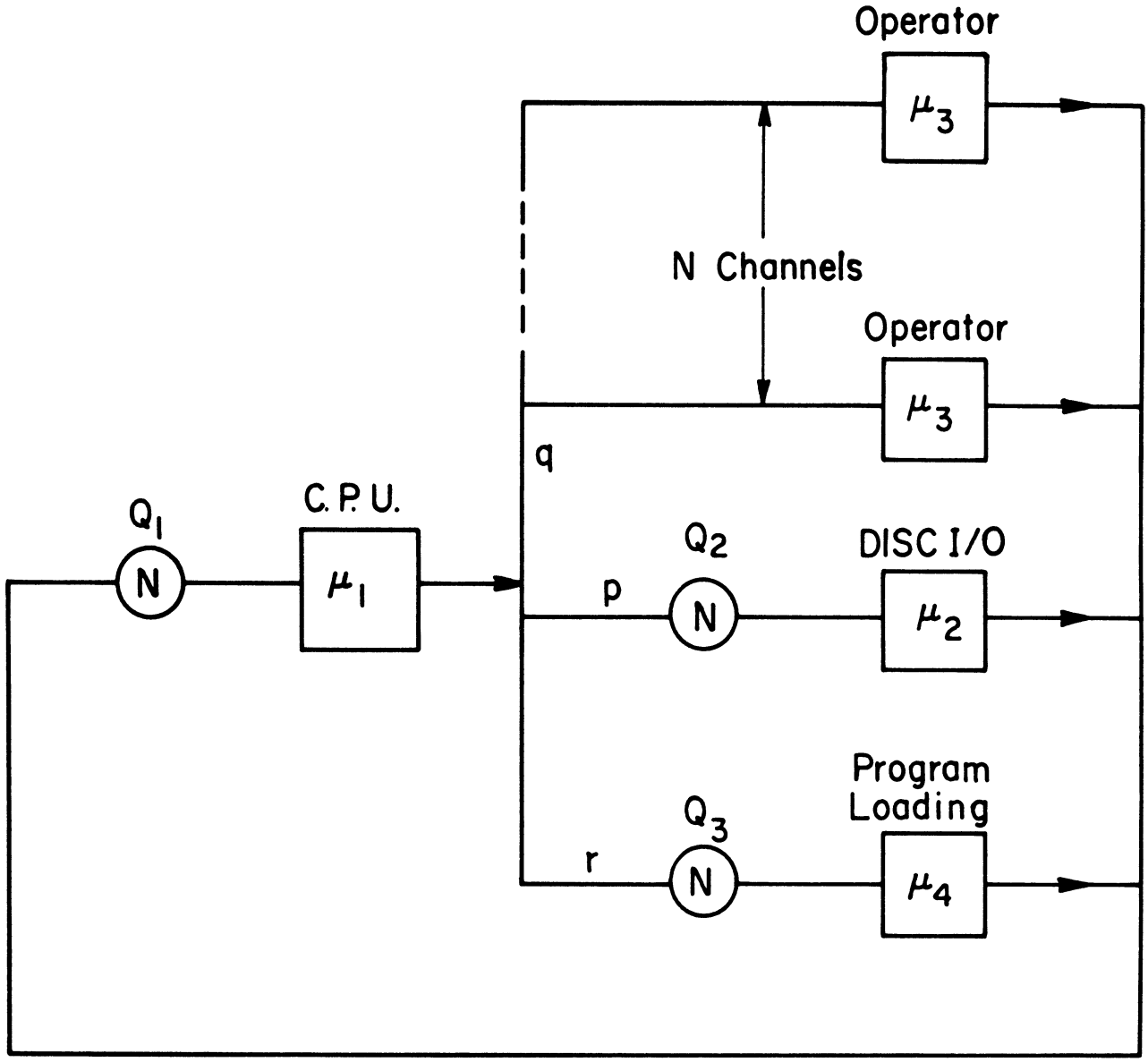

Fig. 9. The basic model of the single-thread system.

from disc file to computer core is modeled by a negative exponential distribution whose mean is designated by $1/\mu_2$; for the disc-memory unit of this system it is about 150 msec.

It is possible for a queue of such transfer requests from some or all of the N programs in the system to form. This queue is designated Q_2 in Fig. 9. After the requested block is transferred, the task program is again eligible to execute. If the CPU is idle, the program commences execution immediately; otherwise it enters queue Q_1 to await execution.

(b) With probability q the task program has transmitted a message to the console operator, and he must respond before execution can continue. Normally he then interprets the message, determines his response, and then responds by use of either the type keyboard or a select button. The time involved for these operations is random; it is modeled by a negative exponential distribution with mean $1/\mu_3$.

Besides the operator's response time, there is involved the time for transferring information from the computer to the console and back to the computer in the query-response cycle. This transfer uses a single I/O channel and the console I/O buffer, which serves all consoles. Hence there could be some queueing in this channel, but because the human response time involved in the total operation is so much longer, this queue and transfer time is ignored. Thus,

in effect it is assumed that query-response cycles with all N operators can be carried out in parallel.

When the operator's response is received, the program is eligible to enter the queue Q_1 to execute.

(c) With probability r ($r = 1 - p - q$) the task program has completed execution and a call has been made to load the next task program of the operational sequence.

Loading requires accessing of index tables, validity-checking tables, and program and data blocks from the disc file. Many short periods of execution of the program-loader section of the executive program will be interspersed among the accessing of blocks from the disc file. The relative times involved are such that the loading time will be completely determined by the disc-acessing times; in the basic model this time is assumed to have a negative exponential distribution with mean $1/\mu_4$.

Requests for loading of task programs might come from all operational sequences within a short period; this would give a queue, Q_3, of loading requests. Each task program must be completely loaded before the next is begun. When loaded, the task program enters queue Q_1 for execution.

So far we have mentioned only the use of the processors by task programs. But since the executive program is a major user of the CPU and the other processors its requirements will affect the processing of task programs, and must be allowed for. For simplicity in modeling, almost all executive-program processor time will be modeled as an integral part of the processor

time taken by the task programs currently in execution. We will now consider the use of each processor in detail.

Task-program use of the CPU has already been described as a series of execution phases interrupted by I/O operations. The execution phases will be assumed to be composed of

(a) Task-program CPU time

(b) Executive-program CPU time necessary to carry out the requests of the current phase of the task program and general executive control functions for the system

The total CPU time taken by both task program and executive program is described by a negative exponential distribution, with a mean of $1/\mu_1$, for each phase of execution. As the probability branching in the model provides for a mean number, $1/r$, of execution phases per task program, we assume, in effect, that the total CPU time per task program has a mean of $1/r\mu_1$ with negative exponential distribution (see Sec. 3.2). A constant portion of this, dependent on the values of p, q, r, and μ_1 is assumed to be used in executive-program functions.

In all models the task program produces a random number of independent requests for block transfers using the disc channel. No distinction is made between input and output transfers, and it is assumed that the disc channel can handle only one block transfer at a time (as under ECP-1). The number of such requests per task program is geometrically distributed, with mean p/r and variance p/r^2. In general, any needs for swapping blocks

of the executive program between computer and disc file are assumed to be included in the quantities attributed to the task programs. If all the executive program and its tables were kept in core, then there would be no such executive disc traffic; however, if storage capacity is limited, the executive program may contribute significantly to the disc I/O traffic. Any I/O traffic to the consoles generated independently by the executive program is also modeled as part of the random number of query responses generated by the user's task program.

The one executive function which has been modeled separately is the loading of a task program. The use of the CPU for this function is not distinguished; rather, it is lumped with task program CPU time. However, the function of the disc channel in loading is important in any model. The number of blocks accessed during the loading of a task program is a random variable, dependent on the program size and the number of tables to be accessed from the disc file. However, the standard interlace size assigned to programs, and the necessary validity-checking and index tables which have to be accessed, do suggest that a distribution with less variance than the negative exponential, as well as a low probability of very short loading times, would be required to describe the loading time. This leads to some alternatives to the basic model.

Since the operations of program-loading and disc I/O requested by task programs cannot proceed concurrently, the executive control program must incorporate a scheduling algorithm. Hence the sequence of block transfers

which constitues a loading operation might be interrupted by a single block transfer requested by some task program. However, in the basic model this was not permitted; entries in Q_3 were given priority over those in Q_2. A different priority scheme is outlined in the description of the alternative models.

4. 2. Variations of the Basic Model

We now identify the three models used in this analysis. Model 1 is the basic one described above and represented in Fig. 9. In use of the disc channel, the program-loading operation is given priority so that, once loading has begun no other disc I/O operation is carried out for any other program. If a loading request arrives when a block is already being transferred for a task program, this transfer is completed before the program loader assumes control of the disc I/O channel. All requests for program loading in Q_3 are then served before any more entries in Q_2 are served.

A schematic diagram of model 2 is shown in Fig. 10. Here a major change is the representation of loading time by a second-order Erlang distribution. The mean of each phase of the distribution is $\frac{1}{A\,\mu_2}$ (for $A < 1$); thus the mean loading time has been directly related to the mean block-transfer time from the disc file $1/\mu_2$. The priority rules for use of the disc channel are the same as in model 1.

Model 3 is also represented by the schematic diagram of Fig. 10. In this model, however, we have attempted a simulation of the case in which preemptive priority is given to the use of the disc channel for block transfers requested by task programs (entries in Q_2). If a task program requests a block transfer while a loading operation (from Q_3) is in progress, the current block transfer in the Q_3 operation is completed, and then the use of the disc

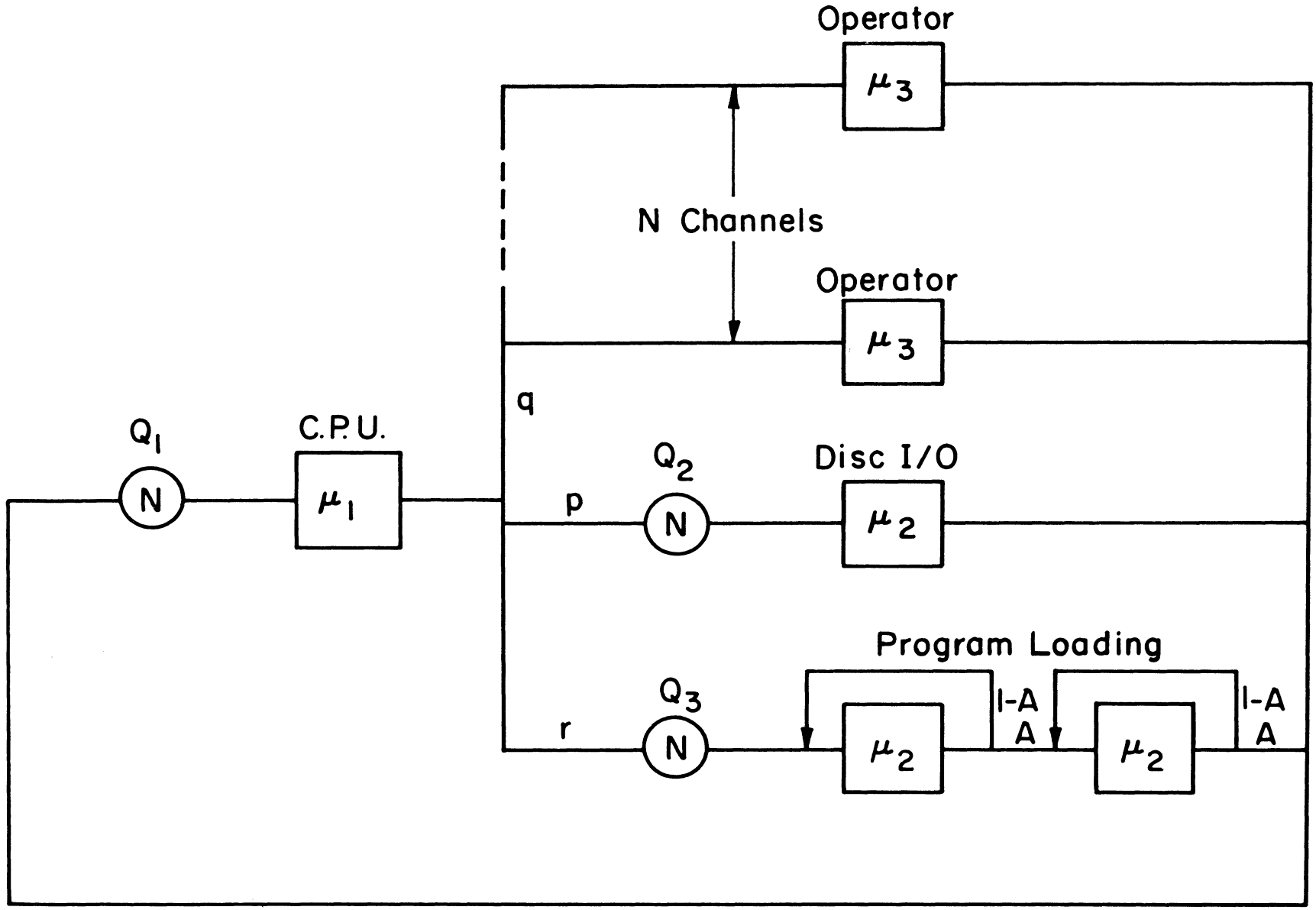

Fig. 10. Models 2 and 3, models of the single-thread system.

channel is given to the Q_2 operation. Any additional requests for Q_2 operations are served before the loader is allowed to use the disc channel again. To implement this strategy in the model, we have used the probability branching in each phase of the Erlang channel. Each time an event in the model occurs that corresponds to a completion in the exponential channel (mean service time $1/\mu_2$) of the loader, a check is made to see if any entries are present in Q_2. If so, all these entries are serviced, preempting the loading operation. When the loading process is allowed to resume, the service enters either the first or the second phase of the Erlang channel, according to the branching probability A. This, in an approximate fashion, corresponds to block transfer in program-loading with preemption by task program I/O.

In no model has any attempt been made to represent the different use of the disc file and data channel in the search operation. That use must here be considered part of the normal disc I/O operations. It is assumed that an I/O request causes a halt in the requesting program. All program and operator characteristics are assumed to have identical distributions for each operator.

5. Results from the Analysis of the Single-Thread System

5. 1. Normalizing

All service times used in obtaining solutions have been normalized to the mean disc access and transfer time for a single block. Thus, in all cases $1/\mu_2 = 1$. The hardware of the single-thread system gives this mean block-transfer time to be 150 msec. The absolute values of the mean service times assumed for all other processor uses can be readily calculated from this figure.

5. 2. Performance Measures

Since one of the prime purposes of this type of computer system is to provide a satisfactory computation aid and automatic control facility for the data-processing steps required by operators using Q-RCC consoles, the system's response to operator inputs is an important measure of performance. Task programs which make up the operational sequences requested by operators will make varied use of the CPU, core storage, disc, and console channels. Programs requiring a large amount of processor time would cause the operator to experience delays, even if he were the sole user of the system. On the other hand, programs requiring moderate amounts of processor time and frequently interacting with the operator may give what the operator will judge as instantaneous response, especially if he is the sole user of the system.

In the light of these comments, one useful measure of the system's capacity is the reduction in service that an operator and his program experience as more consoles are put into use. Two parameters evaluated from the stationary distributions on the states of the models will give such a measure. The first is the operator busy fraction which is that fraction of the time the operator is using the system that he is actually engaged in making responses to his task program. Obviously this parameter will depend greatly on the relative mean values of the task-program processor times and the individual operator's mental and physical response times. However, it probably represents the average operator's subjective evaluation of the

system. The second parameter is the task program response* the average fraction of the time a task program requires to use system processors that it does actually use them. Defining,

B = operator busy fraction

M_1 = the mean time a task program uses system processors in its entire execution

M_2 = the mean time an operator spends in response to his task program

We have task program response $= \frac{M_1}{(1 - B) M_2}$

If there is only one task program in the system, it can use the CPU or I/O channels whenever it requires, so that the task program response is 1. The presence of more than one task program makes this parameter less than 1, and it is a measure of the overall queueing delays experienced by task programs. It depends on operator's response time only insofar as programs are not eligible to use system processors while waiting for operator response.**

Two more parameters give measures of processor usage: the probability that the CPU is in use and the probability that the disc channel is in use. Finally, the parameter program throughput is the mean number of task programs completed per unit time. Program throughput can be derived

---

* This parameter is analogous to Scherr's[5] response parameter.

** This restriction of the models will be true in most on-line operation, except when I/O channel use is brought about by swapping programs.

as a function N (the number of consoles in use by plotting either of the two preceding parameters versus N.

$$\frac{\text{Program throughput } (N = n)}{\text{Program throughput } (N = 1)} = \frac{\text{Probability that CPU is in use } (N = n)}{\text{Probability that CPU is in use } (N = 1)}$$

5. 3. Results

Plots of the performance parameters defined in the preceding section are given in Figs. 11 to 15. Important results from all three models are given for a normalized range of task-program and operator response characteristics. These are defined as follows, in terms of the model parameters shown in Figs. 9 and 10.

$1/\mu_1 r$ = mean CPU execution time per task program (plus ECP overhead)

$1/\mu_2$ = mean block-transfer time between disc file and computer

$1/\mu_3$ = mean operator response time

$2/A\mu_2$ = mean task-program-loading time (models 2 and 3)

$1/\mu_4$ = mean task-program-loading time (model 1)

q/r = mean number of console query-response cycles per task program

p/r = mean number of block transfers between disc file and computer per task program (including ECP overhead)

N = number of consoles in use

5. 3. 1. Processor Usage. In the next three sections all results referred to will be from model 3. The same inferences could be obtained from the other models.

Figure 11 shows how performance is limited as task programs make increasing use of the disc channel. For these curves the mean task program execution time is short ($1/\mu_1 r = 1$), and hence the CPU is idle most of the time. When each task program makes only light use of the disc channel ($p/r = 5$), we see that the operator busy fraction stays high for at least three users. Nevertheless, the task programs experience significant queueing delays in the system (note task-program response) since the increase in the number of users causes the disc channel use to rise noticeably. When disc-channel usage per task program increases to $p/r = 20$, the disc channel becomes saturated. With five users it is in use 95% of the time, and the operator busy fraction drops from 0. 73 to 0. 46. The task-program response curve indicates the queueing delays in the disc channel.

The program throughput, plotted in Fig. 11 (b), curve 1, shows that, for five users, programs are processed at 4. 7 times the rate when there is only one. The reason is that the dominant factor in program completion rate is still the operator's response time, and operators can respond in parallel. However, curve 3 shows that the program throughput is only 3. 7 for five users; here queueing in the disc channel is causing significant delays (shown by a task-program response of 0. 31), and operator's response time is no longer the dominant factor in the completion rate.

For the program characteristics used in Fig. 11, the CPU was

never in use more than 5% of the time. In Fig. 13 we have chosen a contrasting set of characteristics, in which the CPU capacity is the performance limit. Task program execution time is now relatively long: $1/\mu_1 r = 75$. Operator's response time does not solely determine the completion rate, even when there is only a single user of the system. Queueing delays for use of the CPU become significant as soon as there is more than one user. Performance is fairly insensitive to the range of disc-channel usage chosen. Owing to the saturation of the CPU, the program throughput increases very little for more than three users, and for five the operator busy fraction has dropped by 50%.

Note that curve 3 gives slightly higher program throughput than curve 1. The higher frequency of block transfers for task programs in curve 3 increases the number of periods of CPU execution per program but reduces the average duration of each period. The net result is to reduce queueing delays and thus improve the program throughput.

5. 3. 2. Memory Sharing. In the final operating configuration of the system, allocation of the core store to the executive control program and to user programs will have to be carefully balanced. Reducing the allocation to the ECP increases its contribution to the disc I/O traffic; by considering this increase to be reflected in the number of disc I/O transfers per task program in our model, we can reach some conclusions.

Reduction of ECP core storage will allow the use of core store to be extended in two ways: additional task programs can be held in core concurrently, and the number of interlaces per task program can be increased.

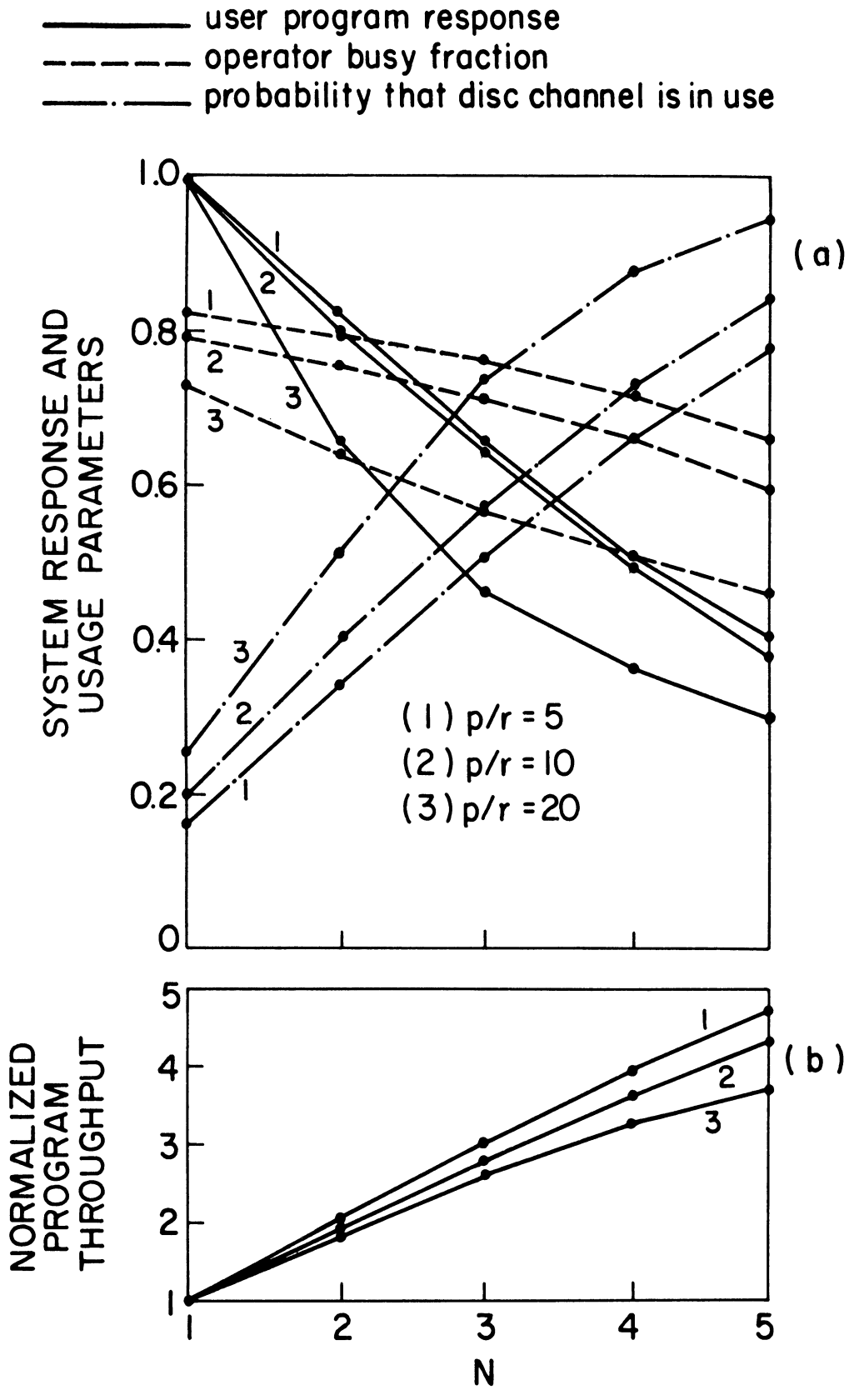

Fig. 11. System performance parameters from model 3 for $1/\mu_1 r = 1$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $2/A\mu_2 = 15$, $q/r = 5$.

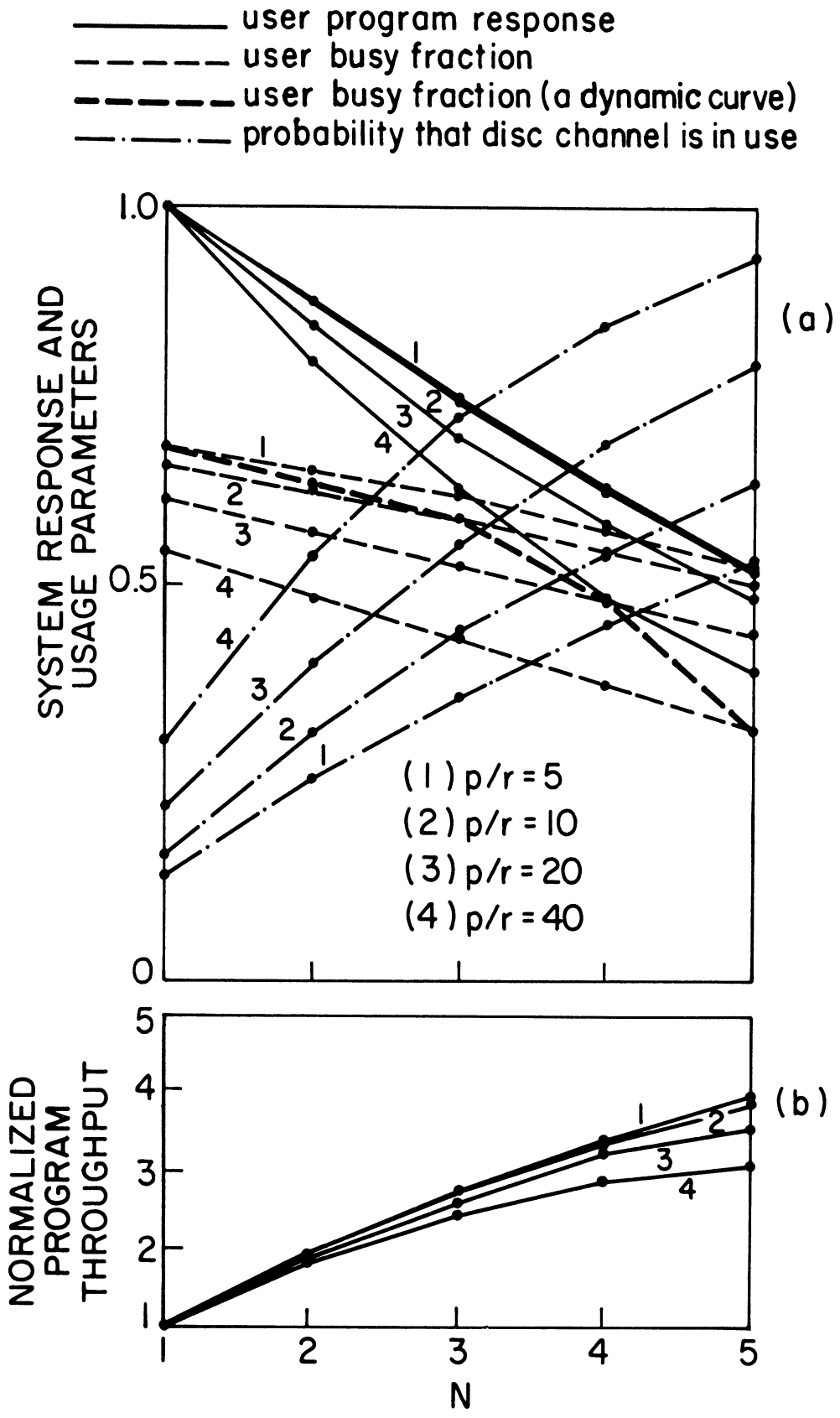

Fig. 12. System performance parameters from model 3 for $1/\mu_1 r = 25$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $2/A\mu_2 = 15$, $q/r = 5$.

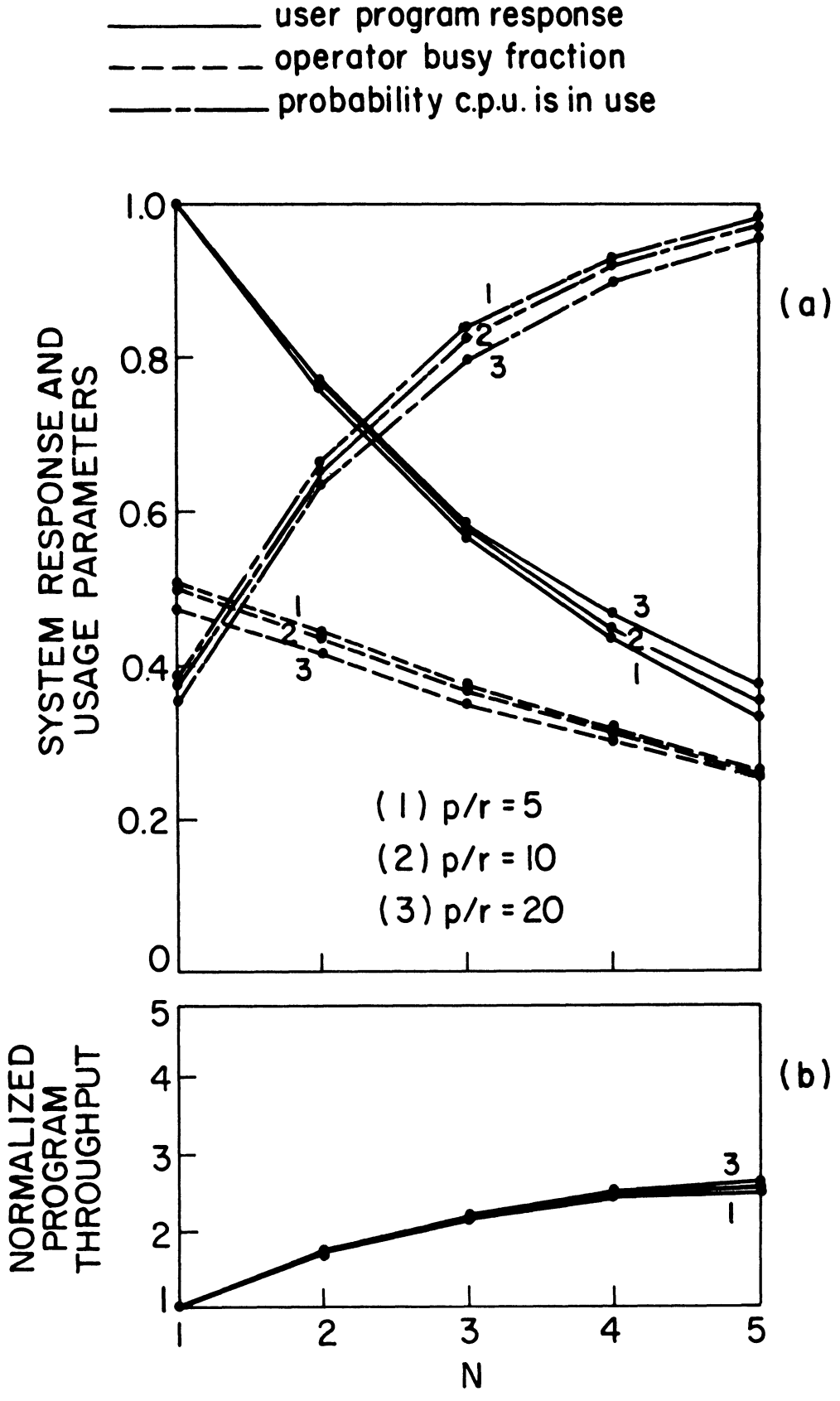

Fig. 13. System performance parameters from model 3 for $1/\mu_1 r = 75$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $2/A\mu_2 = 15$, $q/r = 5$.

In constructing our models we have assumed that the number of concurrent users of the system is small enough to allow a task program for each user to be held in core at the same time. For two or three users this will often be true; granted, for more users task programs will have to be swapped between the core store and the disc file, but our present interest is in performance for a small number of users.

Without detailed analysis of the ECP modules and complete statistics on user programs, it is difficult to give anything but order-of-magnitude estimates of how much disc-channel traffic will be generated by restricting ECP core storage. In its entirety, ECP-1 will require more than four interlaces, or half the core store. If allocated two interlaces, ECP-1 may require more than ten block tranfers between core and disc for each task program executed. (For example, some ECP blocks would have to be loaded for each task-program-loading operation and each scheduling operation, thus requiring overlay and so on). If the allocation were even less, blocks necessary for the control of I/O operations would be restricted from core storage; hence the total disc I/O traffic would depend on the number of I/O operations (with consoles and disc) requested by each task program. The result would be an increase in disc I/O traffic so sharp that the useful system output would probably decrease.

If a single user is operating with the task program and response statistics of Fig. 12, curve 1, his operator busy fraction is 0.69. If the ECP core allocation is now reduced so that a second user's task program

can also be accommodated in core, then by changing the task program I/O statistics to account for the increased disc I/O traffic generated by the ECP overlays, we can determine the resultant performance from the appropriate curve at N = 2. Curve 2 would correspond to an increase of five blocks of disc I/O per task program, and curves 3 and 4 to increases of 15 and 35, respectively. In this manner a dynamic operating characteristic could be determined for the range of N. An example of a dynamic curve for user busy fraction corresponding to the I/O statistics of Table 1 is given in Fig. 12.

Table 1

N	1	3	4	5
p/r	5	10	20	40

If the core allocation to the ECP is reduced to allow larger task programs to be held in core, one is in effect, trading an increase in ECP disc I/O traffic for a reduction in task-program disc I/O traffic. Given sufficient program statistics, comparisons similar to those above can be made.

In general, if the available core storage is insufficient to hold a task program for each user, the scheduling algorithm must be incorporated into the model. However, the algorithm for ECP-1 is simple enough that interpretation from the results for the present models is possible. The algorithm is as follows. No task program is swapped from core, and all sequences progress towards completion at the same rate; that is, one task program for each sequence is completed before another is attempted for

any sequence. Therefore, if there are N users, and only n task programs can be accommodated in core simultaneously ($N \geq n$), each user receives the performance determined from the appropriate curve at the point $N = n$, but only for a mean fraction n/N of the total time he is at the console. For the rest of the time he receives no use of the system.

The effect of an alternative scheduling algorithm in which job programs can be swapped during a wait for an operator's response can be estimated as follows. If two users are operating with the task-program and response statistics of Fig. 12, curve 1, the operator busy fraction is 0.66. If the number of users increases to three, and one task program is always maintained on the disc file, there is some probability that on reception of each operator's response his task program will have to be swapped from the disc file. If we assume this probability to be 1/3, we have a mean of 5/3 task program-swapping operations induced during each task program execution. If the number of blocks involved in each swap is 32 (two interlaces swapped), the extra disc I/O traffic is 50 blocks per task program. Since more than the average number of contigous blocks would be transferred in swapping operations, we could choose curve 4 as indicative of the disc I/O traffic. The operator busy fraction is now 0.44, and the program throughput has increased by 0.4.

5. 3. 3. System Capacity. Sections 5. 3. 1. and 5. 3. 2. treated the variations in system processor use, and therefore in the performance obtained by each operator, with different program statistics and numbers

of users. It is clear that the operator's response time to each query is a key factor in determining how many consoles the system can service without serious loss of performance.

The mean value of operator's response time used in Figs. 11, 12, and 13, when transformed to an absolute time, is about 3 seconds. This means that we expect most operator's responses to consist of single key operations (and this has been suggested by the system planners). However, if many responses were more involved we would have to assume a larger mean response time. To illustrate the resultant effect, the curves of Fig. 14 have been plotted for $1/\mu_3 = 50$, corresponding to an absolute mean response time of 7.5 seconds. For similar program statistics we can notice considerable improvement in the performance parameters defined. However, it would not be advisable to extrapolate these curves past N = 5 without taking into account the necessary swapping and ECP overhead involved.

5. 3. 4. Design Alternatives. In Fig. 15 the performance parameters are plotted* as a function of the number of users for two sets of statistics. Three curves for each parameter are given, to correspond to three design features implemented in the models. Curves (1) apply to reduced loading time for task programs (to be implemented by using a fixed-head-per-track

---

* Note that in Fig. 15 parameter values have been computed only for N = 1, 3, and 5.

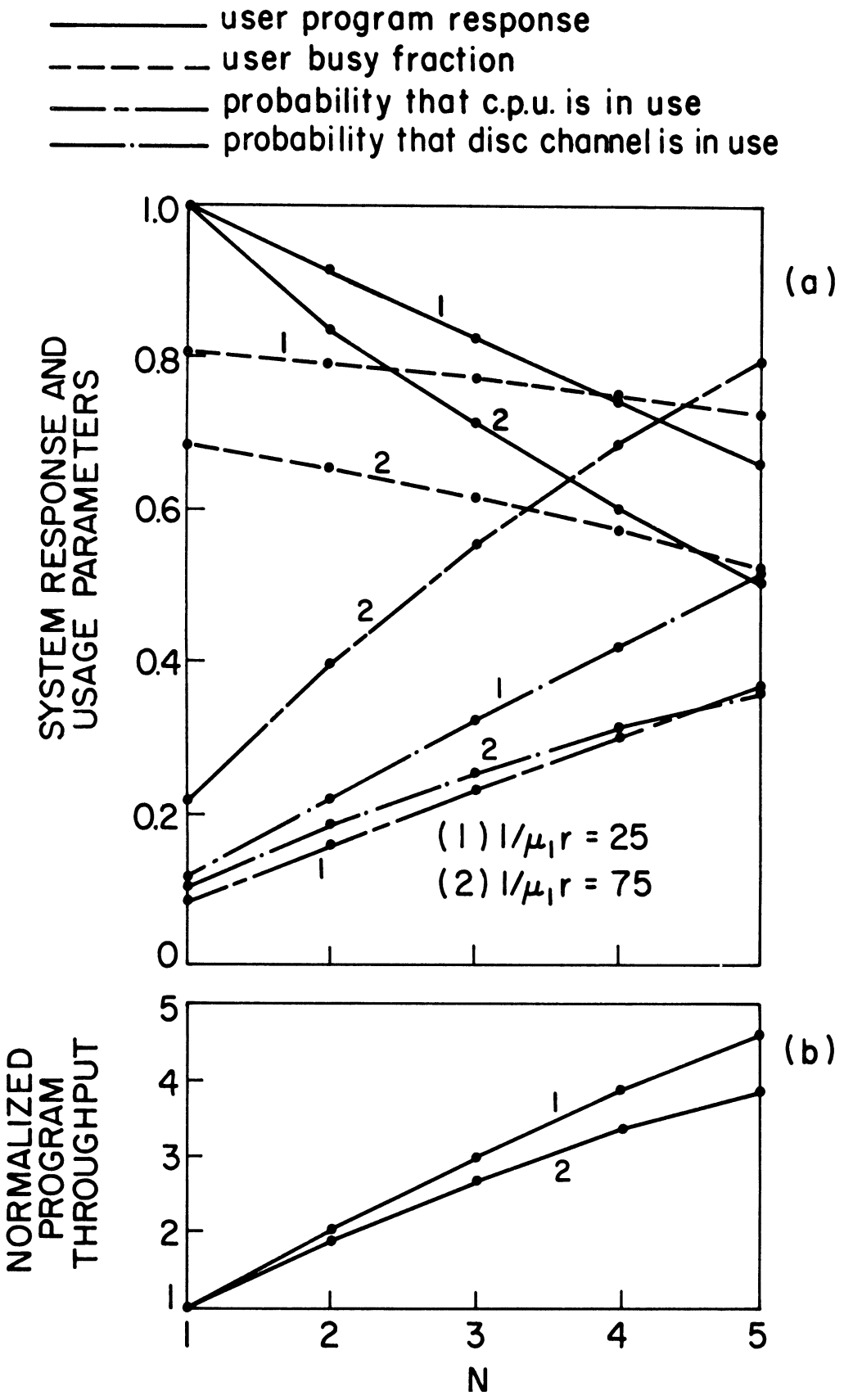

Fig. 14. System performance parameters from model 3 for $1/\mu_2 = 1$, $1/\mu_3 = 50$, $2/A\mu_2 = 15$, $q/r = 5$, $p/r = 20$.

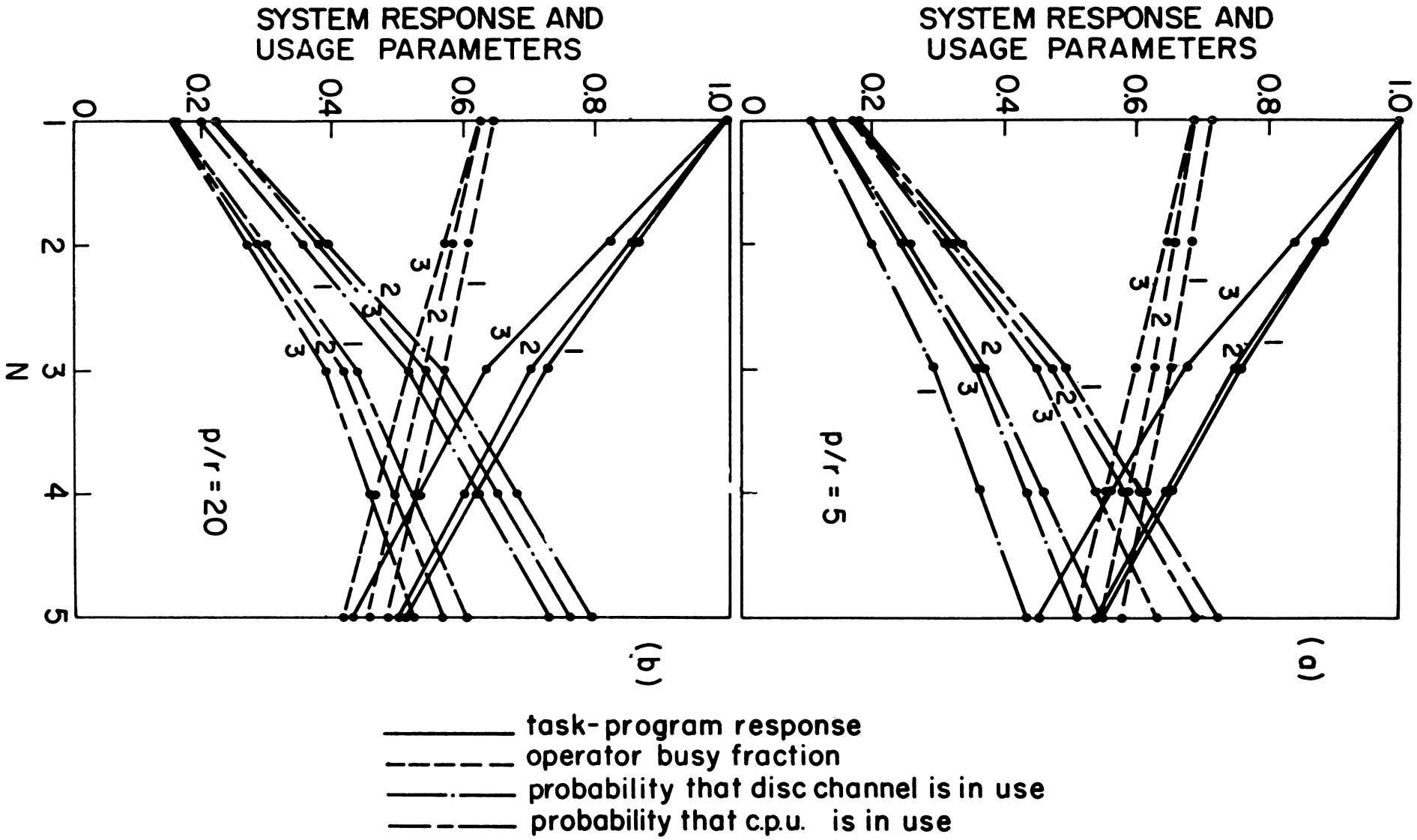

Fig. 15. System performance parameters from models 2 and 3 for $1/\mu_1 r = 25$, $1/\mu_2 = 1$, $1/\mu_3 = 20$, $2/A\mu_2 = 10, 15$, $q/r = 5$.

section of the disc file). Curves (2) apply to loading the same task programs and associated tables from the main disc file. Since both these sets of curves were obtained from model 3, they correspond to the strategy whereby task-program-disc I/O requests are given preemptive priority over new task-program-loading requests for use of the disc channel. Curves (3), obtained from model 2, apply when program-loading requests have priority over task-program-disc I/O requests for use of the disc channel.

The differences in the performance parameters illustrated in Fig. 15 were observed for a wide range of statistics. For more than one user the order of preference for the alternatives based on the performance parameters was always 1, 2, 3. Whenever considerable use of the disc file is made one would expect similar differences in the performance parameters under the three alternatives.

5. 3. 5. Sensitivity of Results to Loading-Time Distribution. Table 2 gives performance parameters for a range of statistics used in models 1 and 2. The scheduling and priority strategies are identical in these two models, but the loading process has been modeled by an exponential distribution in model 1 and by a second-order Erlang distribution in model 2. The results are tabulated for equal mean values of these distributions, and it is seen that the performance parameter values differ by no more than about 1 percent. One could therefore conclude that the critical characteristic of the loading-time distribution is its mean value, insofar as the distribution affects the mean values of performance parameters.

$1/\mu_3 = 20$, $q/r = 5$, $1/\mu_1 = 25$ $1/\mu_4 = 2/A\mu_2 = 15$

$\frac{p}{r}$	N	Prob. CPU is in Use		Prob. Operator is Busy		Prob. Disc Channel is in Use			
						Program-Loading		Task Program I/O	
		1	2	1	2	1	2	1	2
5	1	.172	.172	.689	.689	.103	.103	.035	.035
5	3	.447	.450	.597	.600	.264	.268	.089	.090
5	5	.626	.633	.502	.507	.370	.377	.125	.127
10	1	.167	.167	.667	.667	.100	.100	.067	.067
10	3	.423	.425	.565	.568	.251	.254	.169	.170
10	5	.584	.589	.468	.472	.346	.351	.234	.235
20	1	.156	.156	.625	.625	.094	.094	.125	.125
20	3	.387	.388	.515	.517	.228	.231	.310	.311
20	5	.522	.524	.416	.418	.308	.311	.418	.419

Table 2

6. Models of the Disc File Channel

In this analysis we make the following assumptions on the use of the disc-file channel and the disc-memory units:

(1) Each computer using the disc-file channel is permanently assigned two buffer sectors in the core storage of the disc-file-control unit. Thus the DFCU can accept two data-transfer requests from each computer.

(2) The I/O control modules include the software needed to carry out two input block transfers or two output block transfers concurrently, when the facilities are available.

(3) The time required to transfer a block through the one I/O channel between the computer and the DFCU is so short, in comparison with the total transfer time, that the whole channel may be considered as two parallel channels.

(4) Simultaneous requests for the transfer of blocks involving the same disc of any disc-memory unit are infrequent, as are simultaneous requests for the transfer of more than two blocks from the same disc-memory unit.

(5) The time to transfer a single block in either direction between the computer and the disc-memory unit can be modeled with a negative exponential distribution.

Four models were used to make comparative performance estimates of several disc-channel configurations. These are shown schematically in

Figs. 16 to 19 and will be referred to as models D1(1), D1(2), D2, and D3.

Model D1(1) represents the operation of the disc channel in the single-thread system with ECP-1. Only one computer is using the channel, and only one block transfer may be in operation at any time. We assume that the ECP and task programs generate disc I/O requests according to a Poisson distribution with rate λ, and that each block transfer takes a mean time $1/\mu$ (with a negative exponential distribution). Further, we assume that the maximum number of I/O requests which can be queued for use of the disc channel is reached when all programs are delayed by disc I/O. We denote by n the number of I/O requests waiting or being processed. The arrival process from a computer is discontinued once it has n_{max} requests queued or in process.

In model D1(2) we are again concerned with only a single-computer system, but the dual-transfer capacity of the disc channel is in use. Hence when need arises two block transfers can be in execution at the same time.

Models D3 and D4 represent the use of the disc channel in multi-computer systems. Model D3 has two arrival queues, corresponding to the requests generated by each of two computers. It is assumed that each computer can have at most two transfers in execution at any time, and that the disc channel can have no more than three transfers in execution at any time. Similarly, in Model D3 we consider three computers generating I/O requests. Again, no more than three transfers may be in execution at any time, and no more than two of these may be for any one computer. The

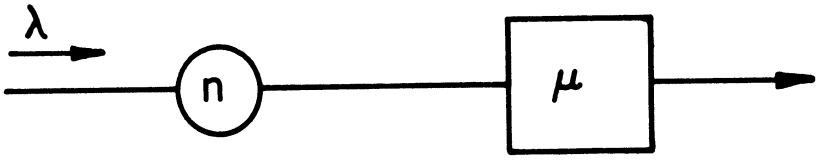

Fig. 16. Queueing model D1(1) of the disc channel for single computer use under ECP-1 strategy.

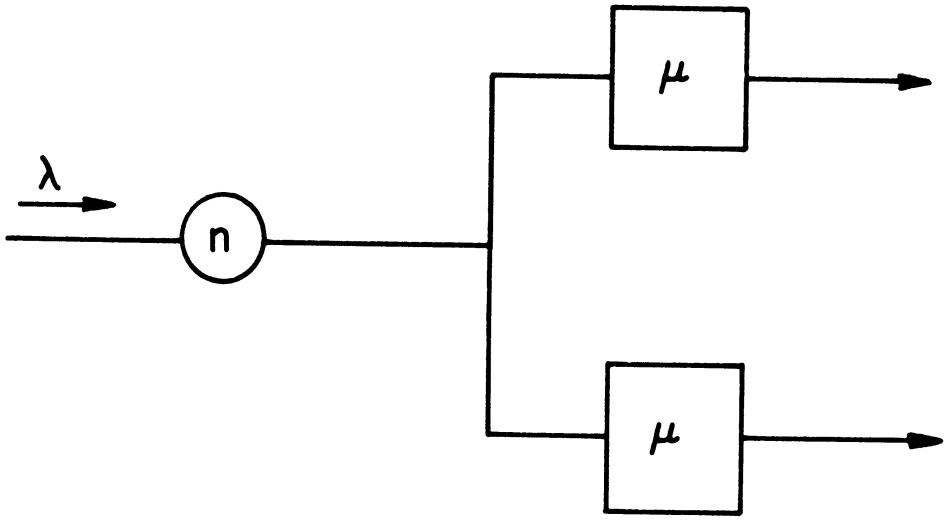

Fig. 17. Queueing model D1(2) of the disc channel for single computer use under the dual-transfer strategy.

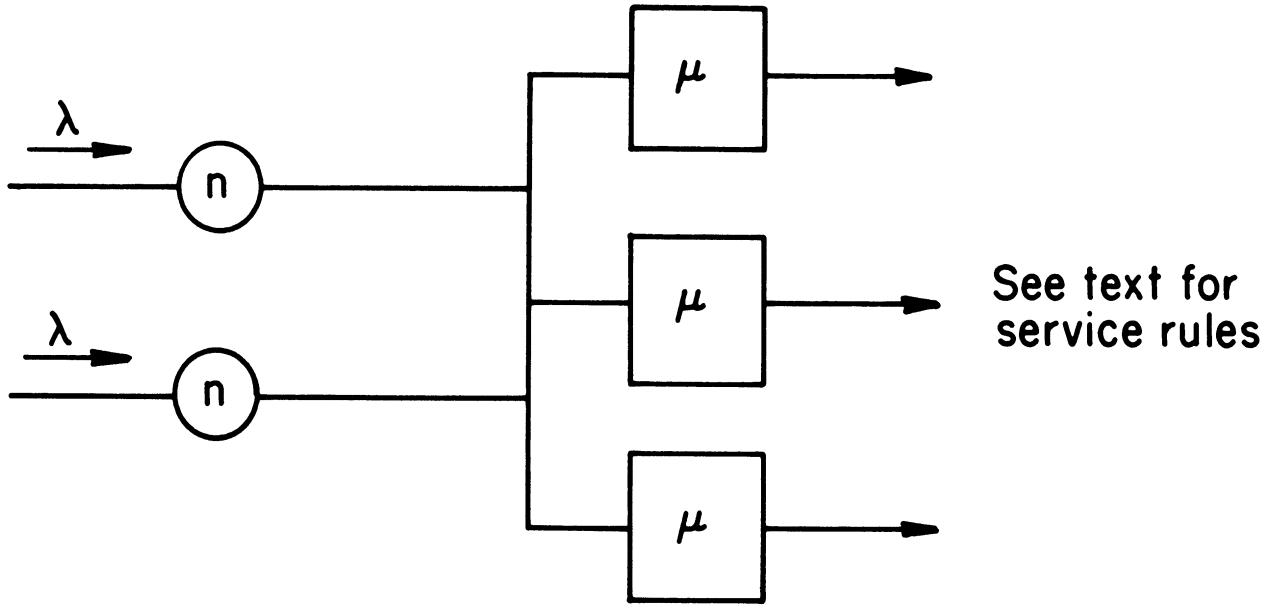

Fig. 18. Queueing model D2 of the disc channel for two-computer use under the dual-transfer strategy.

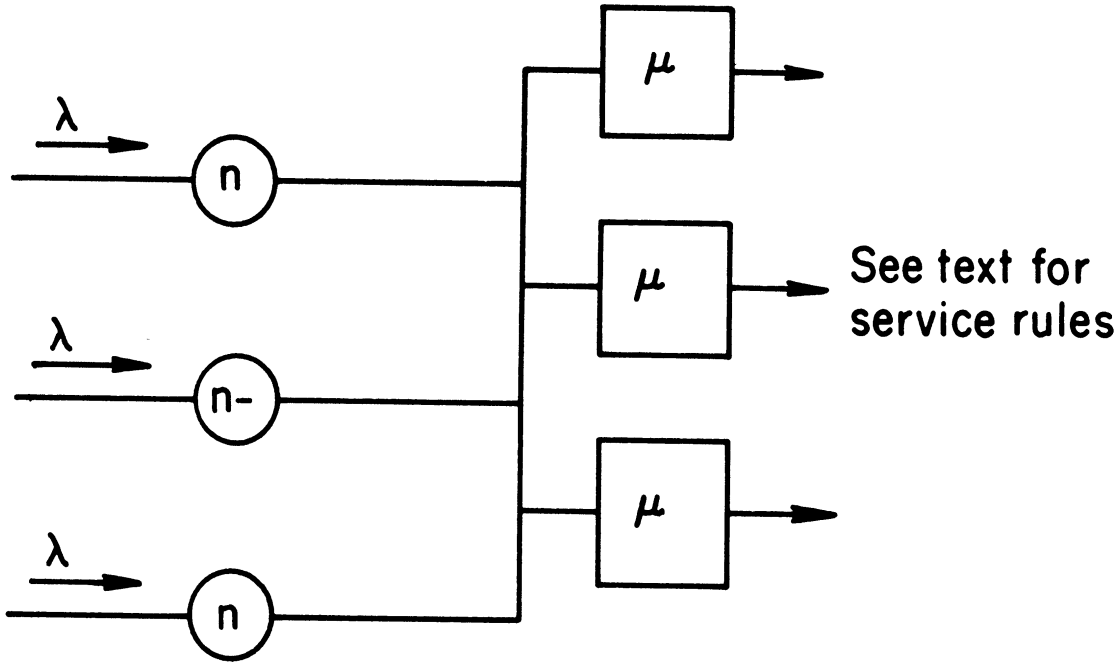

Fig. 19. Queueing model D3 of the disc channel for three-computer use under the dual-transfer strategy.

arrival and service distributions and the restriction on queue length are the same as for Model D1(1).

7. Results from the Disc-Channel Models

The performance of the disc channel in the various computer systems which it serves can be measured by its capacity to transfer blocks between the disc-memory units and the computers and by queueing delays experienced by programs using the disc channel. We define two performance parameters to measure these characteristics. The first parameter is throughput, the normalized rate at which blocks are transferred. The normalization is with respect to the mean transfer time $1/\mu$ for a single block located in semi-random fashion (see Ref. 6, Appendix B). The second parameter E(n), is the expected number of transfer requests waiting or in process per computer.

If the disc channel is not to be a bottleneck in the operation of the system, one would expect E(n) to be little larger than the number of simultaneous transfers allowed for each computer. That is, the number of I/O requests queued should be small so that only short delays beyond the transfer time will be induced. The maximum throughput, with no queueing delay, could be obtained if an I/O request were to occur just at the completion of each block transfer. The throughput per computer for models D1(1), D1(2), D2, and D3 would then be 1, 2, 1.5, and 1 respectively. However, the arrival process for I/O requests is random and mostly independent of the transfer process. Therefore one cannot expect the maximum throughput of the channel to be obtained in practice, and the actual

throughput will depend on the mean rate of arrivals.

In Figs. 20(i) to 20(iii), throughput per computer and E(n) have been plotted against the normalized arrival rate ρ, with constant n_{max} (the maximum number of I/O requests per computer).

$$\rho = \frac{\text{mean arrival rate for I/O requests}}{\text{mean transfer rate for I/O requests}} = \frac{\lambda}{\mu}$$

For each model, throughput is asymptotic to the maximum values quoted, as $\rho \to \infty$. However, significant queues for most of the operating time have formed well before the asymptote is neared. We now compare the performance of the four configurations modeled.

Model D1(1) gives a standard for comparison. This is the performance to be expected in the single-thread system with ECP-1. Only one transfer may be in operation at any time. With $\rho = 1$, we expect half the programs to be requiring use of the channel at any time ($E(n) = n_{max}/2$); and for a typical number of programs ($n_{max} = 4$) the channel is operating 80% of the time.

In model D1(2) we have implemented the dual-transfer capacity of the disc channel and assigned it completely to one computer. For $\rho = 1$, the curves show only small queueing delays and the use of the channel is about 50% of the total capacity. For twice that request rate ($\rho = 2$), queueing delays and total channel usage are the same as in the single-transfer case with $\rho = 1$. Correspondingly, throughput is almost doubled.

In the multi-computer system the results of model D1(1) would apply on a per-computer basis if each computer were restricted to a single transfer operation at all times (ECP-1 type strategies). The results of models D2

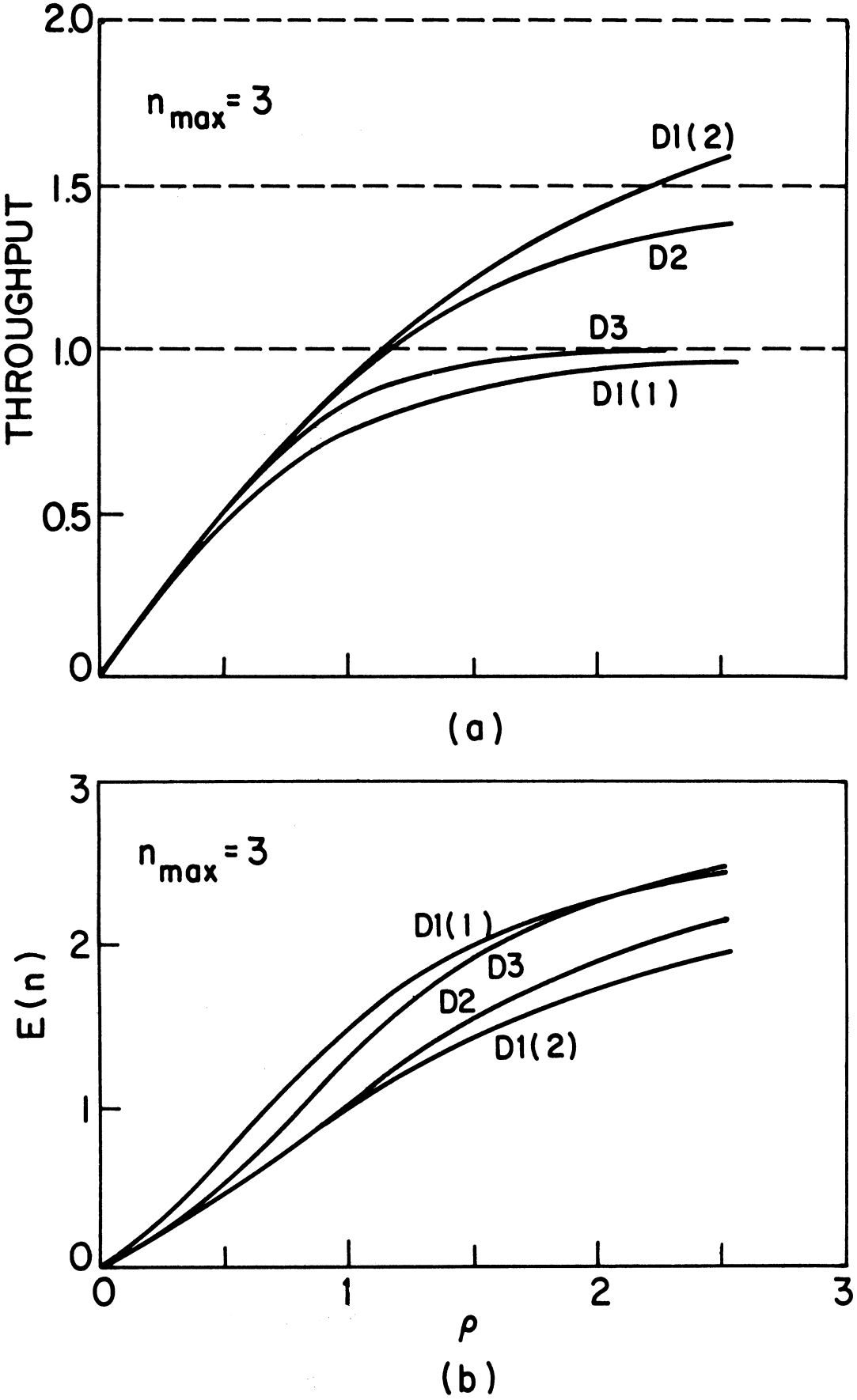

Fig. 20(i). (a) Normalized throughput per computer v ρ.
(b) Mean number of I/O requests per computer (queued or in service) v ρ.

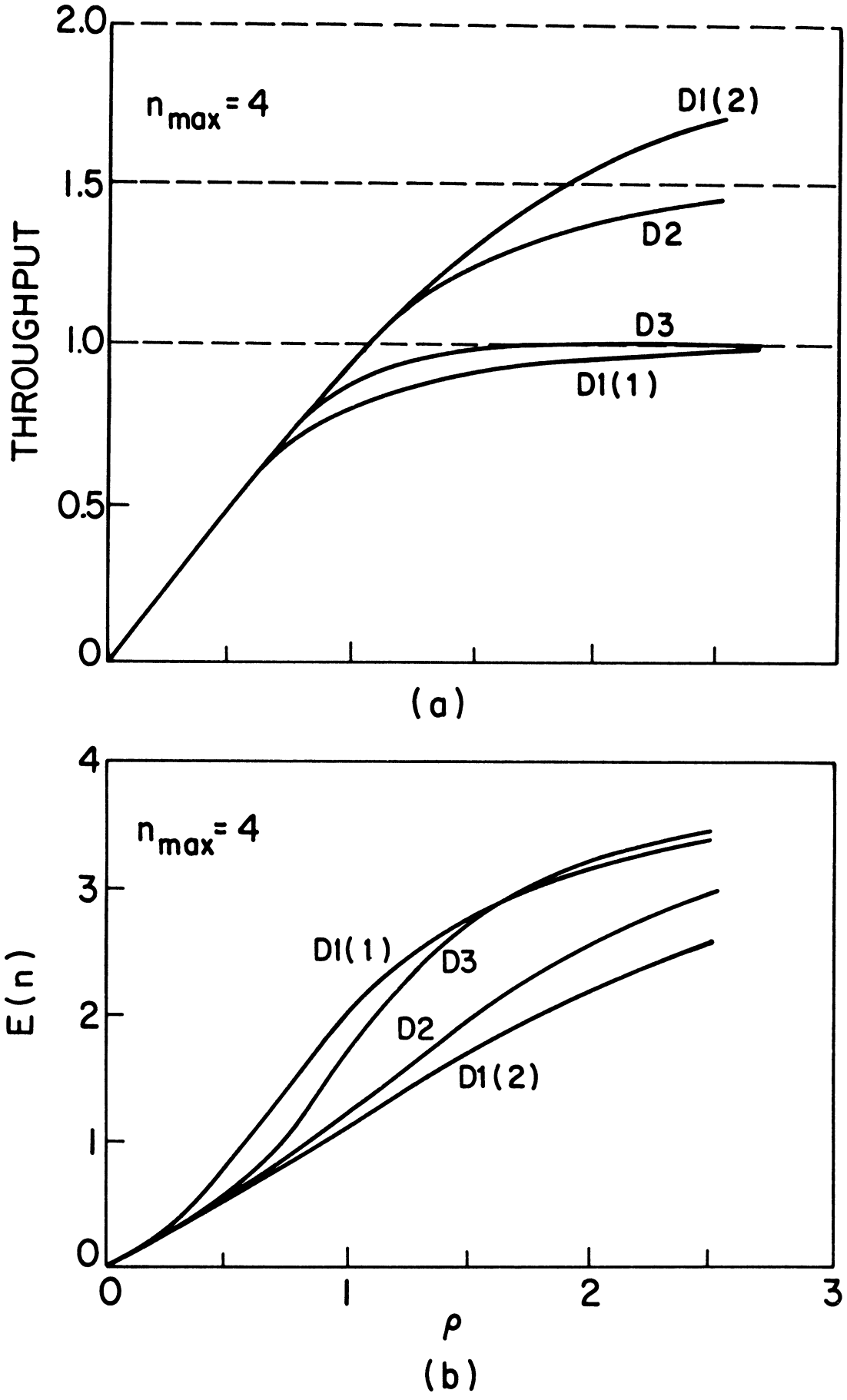

Fig. 20(ii). (a) Normalized throughput per computer v ρ.
(b) Mean number of I/O requests per computer (queued or in service) v ρ.

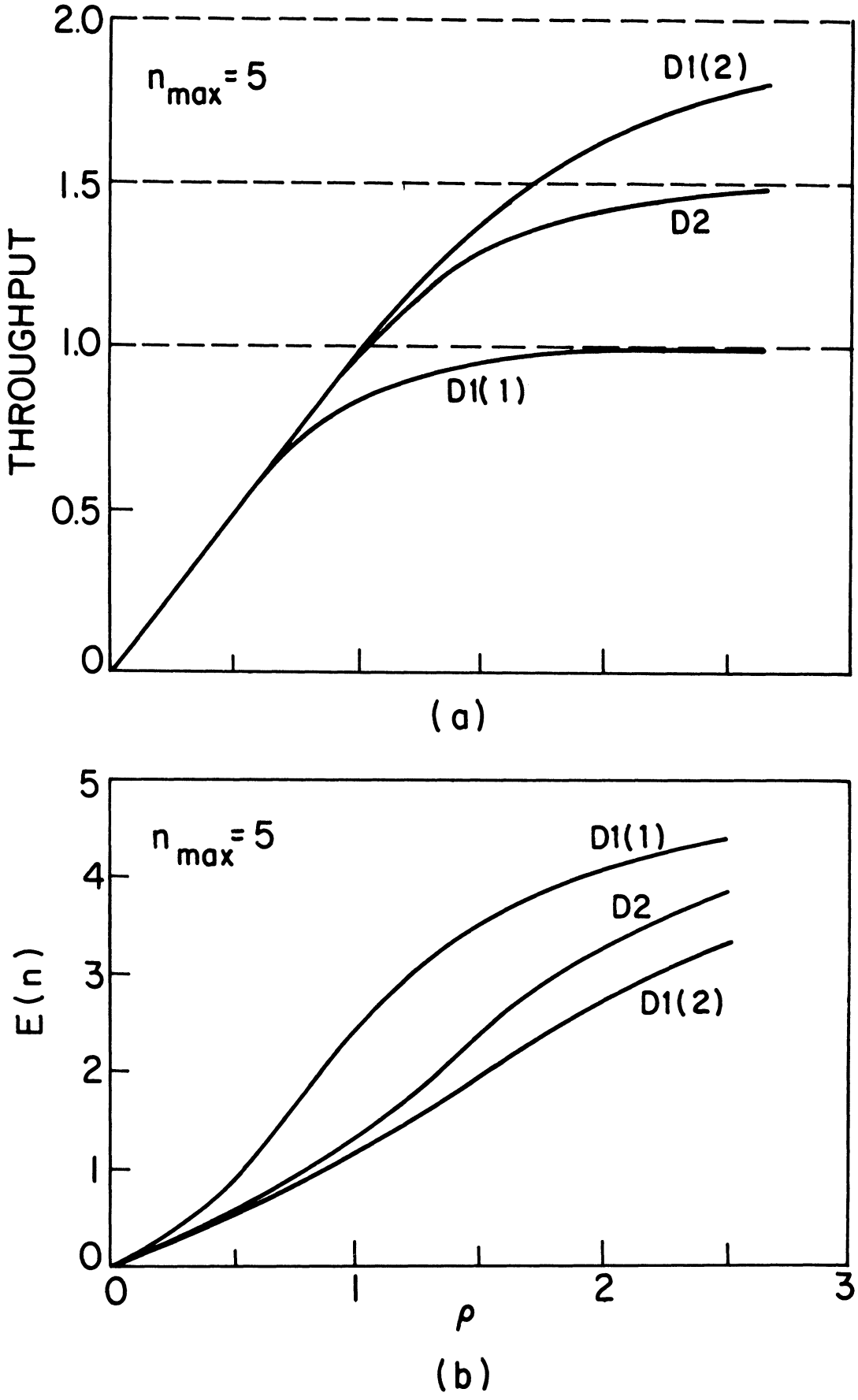

Fig. 20(iii). (a) Normalized throughput per computer v ρ.
(b) Mean number of I/O requests per computer (queued or in service) v ρ.

and D3 show how performance is improved by sharing the total transfer facilities according to the immediate requirements of each computer. Some improvement could have been expected, because if random request rates for each computer are allowed one computer may have two or more transfer requests while the other computers have a total of one. In this case the DFCU can still execute the maximum of three transfers.

In the two-computer system (Model D2), with $\rho = 1$ for each computer, performance is much better: both throughput and E(n) are close to the values obtained when two transfers can be executed for each computer at all times (Model D1(2)). In the three-computer system the improvement is not as good, for $n_{max} = 5$ throughput and E(n) are little better than under the ECP-1 strategy (Model D1(1)). For higher arrival rates ($\rho > 1$) there is always some advantage in implementing dual transfers for the two-computer system. We would expect this because we are in effect increasing by a factor of 1.5 the amount of channel usage we can devote to each computer. For three computers, however, higher arrival rates result in almost the same performance as when each computer is restricted to a single transfer operation. With a maximum of three transfers and queues of requests waiting in each computer, only one transfer can be allocated to each computer.

8. Conclusions

The implication of the analysis of the disc-channel models is clear. The advantage which could be obtained by the use of dual transfers for a computer in the three-computer systems is small. Considering the increased

complexity of the software and the further reduction in performance resulting from conflicts for use of the same disc or disc-memory unit, it would not be worth implementing. On the other hand, the advantage in a one- or two-computer system subject to high usage is significant. One would conclude that, should disc-channel capacity prove a bottleneck in the single-thread system under ECP-1, consideration should be given to implementing the dual-transfer capability of the disc channel in future one-/and two-computer systems.

Without user statistics it is difficult to draw absolute conclusions from the analysis of the single-thread system. The discussion in the latter part of Section 5. 3. 2. has shown how the performance characteristics plotted may be considered as static operating curves. A dynamic operating characteristic giving the amount of disc-channel usage per program (p/r) as a function of the number of users is required to predict true performance. With such data, and with the statistics necessary to determine the other parameters of the model, we could draw definite conclusions from curves such as those plotted. In Fig. 12(b), for example, if the dynamic operating curve passed through the parameter values (N = 4, p/r = 20) and (N = 5, p/r = 40) the program throughput would actually diminish between N = 4 and N = 5. This would provide strong argument that four consoles would fully load the single-thread system.

We have noted that fixed-head-per-track disc accessing in the loading operation improves the system's performance. Whether the improvement is large enough to warrant the extra cost is uncertain. For the statistics assumed in this report, the disc traffic other than loading traffic has been more significant. This suggests that the fixed-head-per-track media should

also be considered for use in task program swapping operations. In terms of the model parameters this would have the effect of reducing the value of $1/\mu_2$.

Again the necessity for measurements of program and user characteristics for effective use of modeling is emphasized. The models described in this report were derived with much use of approximation and of assumption about the operation of the corresponding systems. Only results from actual operation of these systems can confirm the validity of the models. However, experience to date has shown that, when one is interested in mean value performance, liberal approximation and lumping of components in the modeling process can be made without changing the significant results. We therefore believe that this initial attempt at large-scale Markov modeling of these computer systems will provide some useful performance guides. However, there is need for more accurate modeling; some obvious extensions are mentioned below.

The most serious weakness of the models of the single-thread system presented in Section 4 is that they do not provide for distinction between the use of system processors by the ECP and their use by task programs. If task programs are making heavy use of the system, then ECP requirements on the CPU, high speed memory, and I/O channels by the ECP can significantly affect performance. Also, ECP processor time will be related to the number of users of the system. Hence it is very desirable to have models in which certain parameters represent ECP characteristics only.

The high speed memory sharing is the most distinctive property of all time-sharing system designs. Closely associated with it is the ability to transfer information between the low-speed, large-capacity memories and the high speed memory. We therefore need more effective modeling of the scheduling algorithms for assigning use of the high-speed memory and the CPU, to incorporate the idea of fixed times rather than completely random ones. Further, the direct effects of the limited capacity of the high-speed memory must be taken into account by modeling parameters to represent swapping.

The disc channels will be much used in search operations in some applications. Since this operation is different and time-consuming, it will be useful to model it so that it can be distinguished from the normal block-transfer operations.

Finally, the large systems proposed involve multiple processors, multiple core modules, etc. It is evident that modeling such systems in full will produce unnecessarily large models. Exploration of useful symmetrical properties of the models will produce more tractable ones.

REFERENCES

1. V. L. Wallace, R. S. Rosenber, The Recursive Queue Analyzer, Tech. Report No. 2, Systems Engineering Laboratory, The University of Michigan, February 1966.

2. C. A. Christensen, Development of an Electronic Processing Module for Mobile Wing Reconnaissance Technical Squadron, Tech. Report No. RADC-TR-64-520, March 1965, Rome Air Development Center, Rome, New York.

3. Mobile Wing Reconnaissance Technical 60 Day Systems Report Center for RADC, Contract AF 30(602)-3171, prepared by Univac, St. Paul, Minnesota, 15 August 1963.

4. P. M. Morse, Queues, Inventories and Maintenance, Wiley, New York 1963.

5. A. L. Scherr, "An Analysis of Time-Shared Computer Systems," Doctoral Thesis, Department of Electrical Engineering, M. I. T., Cambridge, Mass., June 1965.

6. V. L. Wallace, D. W. Fife, R. F. Rosin, A Study of Information Flow in Multiple-Computer and Multiple-Console Data Processing Systems, Technical Documentary Report No. RADC-TDR-64-427, Dec. 1964, Rome Air Development Center, Rome New York.