QUEUEING MODELS

FOR

DISTRIBUTED COMPUTER SYSTEMS

A Dissertation Presented

By

Yuan-chieh Chow

Submitted to the Graduate School of the
University of Massachusetts in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 1977

Department of Computer and Information
Science

QUEUEING MODELS
FOR
DISTRIBUTED COMPUTER SYSTEMS

A Dissertation Presented

By

YUAN-CHIEH CHOW

Approved as to style and content by:

Walter H. Kohler
Dr. Walter H. Kohler, Chairperson of Committee

Richard H. Eckhouse Jr.
Dr. Richard H. Eckhouse, Jr., Member

Harold S. Stone
Dr. Harold S. Stone, Member

Conrad A. Wogrin
Dr. Conrad A. Wogrin, Member

Robert M. Graham
Prof. Robert M. Graham, Department Head
Computer and Information Science

August 1977

ACKNOWLEDGEMENT

The author would like to express deep gratitude to Professor Walter H. Kohler for his suggestion of the research topic, technical contributions to the preparation of the dissertation, constant support and encouragement, helpful advice, and demonstration of significant ability to direct research project. The research was supported in part by National Science Fundation Grant MCS 76-03667. The author is also indebted to the members of his dissertation committee, Dr. Richard H. Eckhouse, Jr., Dr. Harold S. Stone, and Dr. Conrad A. Worgrin for their patience and constructive advice. Special thanks go to Dr. Donald F. Towsley for providing the author many technical suggestions and insight in this research area. The dissertation could not have been completed without the excellent typing of Mrs. Olanyk and my wife Taiying.

ABSTRACT

Queueing Models
for
Distributed Computer Systems

August 1977

Yuan-chieh Chow, B.S.E.E., National Chiao Tung University,
M.S. and Ph.D., Computer and Information Science,
University of Massachusetts

Directed by: Professor Walter H. Kohler

Queueing models are developed for distributed computer systems that dynamically balance the workload among cooperating autonomous processors. Workload balancing is achieved by using adaptive job scheduling policies. Several adaptive state dependent job routing strategies are proposed. Examples are used to demonstrate that system performance as indicated by job turnaround time and throughput rate can be improved significantly by dynamically balancing the workload.

Numerical solution techniques for the analysis of simple homogeneous and heterogeneous models are developed. In particular, the method of analysis for two-processor models with well defined deterministic job routing strategies is shown to be dependent on special properties of the two-dimensional state diagrams. An efficient recursive solution method for computing the state

equilibrium probabilities is introduced for these cases. In other cases the Coxian staging method is used to derive the new job inter-arrival time distribution created by the job routing policy.

Finally, the relationships between system reliability and performance is briefly considered. It is shown that a processor with exponential failure and repair functions can be replaced by an equivalent failure free processor. The equivalent service time distribution is derived by using the Coxian staging method. The approach illustrates that the consideration of system reliability can be embedded into performance measures.

TABLE OF CONTENTS

CHAPTER PAGE

CHAPTER I

INTRODUCTION

Multiple-processor computer systems have become an important research area in computer system design and data processing. The concept of such systems evolves from the desire to share resources (data, programs, hardware), the need to achieve very high-speed computation, and the requirement of high reliability in some applications. One of the crucial requirements in the design of a multiple processor computer system is to estabilish control over the flow of jobs in the system to improve overall system performance.

This dissertation introduces and analyzes queueing models for distributed computer systems (DCS), a special class of multiple processor systems with state dependent job routing policies. The use of queueing networks as the basis for modeling computer systems has been studied extensively in recent years and has proved to be a powerful tool for system analysis. However, the modeling of multiple-processor computer systems with state dependent job routing policies is a relatively new and undeveloped research area. In the dissertation we propose and analyze queueing models with state dependent routing strategies for both homogeneous and heterogeneous distributed systems. These models include closed and open systems and are motivated by our desire to analyze real multiple-processor systems that share resources to improve performance. The results of this analysis can be used to influence the implementation of such systems. Both deterministic and nondeterministic state dependent job routing policies

that attempt to balance the overall system workload are proposed and formally defined. These job routing strategies are shown to be effective in improving system performance. Essential to the analysis of queueing models is the development of solution techniques which can provide efficient and accurate solutions. We investigate the existing solution methods and their limitations. Some analytical solution techniques are extended and shown to be useful for the analysis of our models.

Chapter II contains our definition of distributed computer systems (DCS), some background material, and a survey of previous work. Chapter III presents the general concept of a DCS model. It also includes a survey of existing solution methods and their limitations. In Chapter IV we discuss homogeneous distributed systems. Multiple and single-processor systems are analyzed and compared, and the concept of load balancing through a communication channel is introduced. Chapter V presents various job routing policies for heterogeneous systems. The proposed job routing strategies are formally described and a recursive solution method is illustrated and shown to be effective for the heterogeneous models with deterministic state dependent job routing policies. The effects of these routing strategies on system performance are compared. Chapter VI presents an initial investigation of the relationship between system reliability and performance. Chapter VII summarizes the results of our study and suggests directions for future research.

CHAPTER II

GENERAL CONCEPTS AND PREVIOUS WORK

2.1. Definition of Distributed Computer Systems

Distributed computer systems (DCS) are a special class of multiple-processor systems. Multiple-processor systems are sometimes categorized as "multiprocessors", "distributed function computers", "distributed systems", and "computer networks", depending upon their degree of decentralization of resources, control, and data. They range in organization from two processors sharing a common memory to a large number of relatively independent but interconnected computers which are geographically separated. Various criteria can be used to classify multiple-processor systems [FULL 73, ANDE 75, ENSL 76]. One that suits our study is the concept of "coupling" [FULL 73]; that is, the relative overhead of inter- vs. intra-processor transactions. A system is measured as "tight", "intermediate", or "loose", etc., with respect to the degree of coupling.

For example, array processors like ILLIAC IV [BARN 68] would be classified as tightly coupled. Identical processing elements (PEs) synchronously execute a common instruction stream under the control of a single control unit while a direct data path exists from each PE to its orthogonally neighboring PE. Multiprocessor systems in which all processors execute asynchronously while sharing common peripherals and main memory are somewhat less tightly coupled. However, they rely on a common supervisory program for synchronization. C.mmp [WULF 72] is one state-of-the-art

example and many others can be cited [BAER 73].

In contrast to these systems we have existing computer networks which are loosely coupled. Whole programs and data files are transferred between processors sometimes via a separate communications subnetwork. Resources can be shared in this way, but the overhead of each interprocessor transaction is high. ARPANET is a prime example of a very loosely coupled multiple-processor system. Not only are the host processors geographically distributed, but they are also inhomogeneous and thus interprocessor transactions can occur only indirectly by using a common procotol [KAHN 72]. The resource sharing advantages of a network architecture are now available to minicomputer users through host-satellite configurations [HEWL 74, MODU 75]. The processor interconnections may be either hardwired or via modems and the telephone network. While capable of being more tightly coupled than the ARPA Network, these multiple-processor systems are still loosely coupled compared to multiprocessor systems like C.mmp. The host-satellite systems provide the ability for centralized control of a distributed system and the sharing of files and high cost peripherals. These systems have already found wide application in real-time process and inventory control.

Intermediate coupling, the area between these two extremes, has not gone unnoticed [STON 75, VAND 74]. This area includes systems where the control, storage, and execution of program and data modules are dynamically distributed over multiple processors. For example, modules might be combined into a local working set

resident on some processor. As the working set and/or system load changes, execution might shift dynamically from one processor to another. To minimize the overhead of interprocessor transaction, simple interprocessor protocol is necessary. The minicomputer/ multiprocessor Interface Message Processor designed for ARPANET by Bolt BERANEK and Newman [HEAR 73] is an example of a special purpose system that includes some of these features. In this case the system program is finely partitioned into subtasks which are dynamically ordered by priority. The identity of the highest priority task is passed to the next available processor which then loads the necessary data and program from a common main store and proceeds to execute the task locally. This is an example of intermediate to tight coupling. Thomas [THOM 73] and Cosell et al. [COSE 75] describe the resource sharing capability in ARPANET made possible by a distributed executive program (RSEXEC) and a companion service program for TIP users (TIPSER) that run on TENEX hosts and allow users to transparently access all TENEX subsystems and a virtual file system distributed over multiple PDP-10 TENEX hosts. The load is dynamically balanced by broadcasting user service requests and then selecting the most responsive host to provide the service. Although this system has some of the features characteristic of intermediate coupling, the existing ARPANET host-to-host protocol imposes interprocessor communication overhead that restricts its effectiveness. Another prototype that approaches intermediate coupling is the host/satellite graphics system described by Van Dam [VAND 74] and stabler [STAB 74].

In this research we define distributed computer systems (DCS) as the class of multiple-processor systems with the following characteristics:

1. intermediately coupled;
2. interconnected set of decentralized compatible processing elements (PE);
3. each processing element is usable both as a stand alone system and with other PE's for resource and load sharing;
4. the control, storage, and execution of program and data modules can be dynamically distributed over all processing elements.

In general, these systems fall into a category between two extremes: multiprocessors and computer networks. It is further assumed that these systems are generally reconfigurable and consist of slow PE's. That is, each PE has medium to low memory-processor bandwidth. Distributed computer systems as defined above are motivated by the following objectives:

1. utilization of available low-cost minicomputers and microprocessors;
2. sharing of resources;
3. enhancement of performance;
4. high reliability;
5. modular growth;
6. use of distributed data bases.

The distributed computer system models proposed and investigated in the dissertation are based on the above definition and motivations.

Multiple-processor system development will continue to find impetus from advances in hardware technology [FULL 73]. In particular, the availability of potentially powerful yet inexpensive microprocessors with local memory and control provides strong motivation to design multiple-processor systems utilizing these modules as building blocks. By using these modules to design systems that support the distributed control, storage, and execution of user programs, we may obtain significant advantages in modularity, expandability, reliability, and cost/performance.

2.2. Queueing Networks as Models for DCS

We are interested in the performance measures of distributed computer systems. These measures usually consist of the average system throughput rate, average job turnaround time, average length of waiting lines for devices, processor utilizations, etc. To obtain such measures one needs a model which can adequately describe the behavior of the system. The behavior of a computer system depends on the interaction between the workload and the system componenents. The workload of a system is characterized by the job distribution among system components, the job arrival pattern and their service requirements. The system components include central processors, input/output devices, storage, channels, and controllers.

Queueing models have proved helpful in modeling the behavior of computer systems. Each component or group of components in a distributed computer system can be viewed as a server with an associated waiting line (queue). The job arrival pattern and the service

time of a server are both described as random processes. The service process depends on the job characteristics and the system components which represent the server. The scheduling discipline of each queue is determined primarily by the server associated with the queue. The control of job flow in the system can be characterized by the job branching probabilities among system components. The control is also called the job routing policy.

We assume that a distributed computer system is modeled as an multiple server queueing system. Sever/queue pairs are interconnected to form a queueing network. Given the above descriptions of the system components and workload, the behavior of the computer system can be described as a discrete-state-continuous-time stochastic process. The state of the system is a snapshot of the workload distribution among the system components at time t and can be expressed as

$$S(t) = (n_1(t), n_2(t), \ldots, n_m(t))$$

where $n_i(t)$ is the number of jobs in the ith queue at time t and m is the total number of queues. The central problem in the analysis of a queueing network is to solve for the state probabilities P(S(t)). Performance measures of a system can be obtained if the P(S(t)) are available. If we are only interested in the steady state behavior of the system, and the system does reach equilibrium, then we can represent the equilibrium state probabilities P(S) as

$$P(S) = \lim_{t \to \infty} P(S(t)).$$

P(S) are obtained by solving the equilibrium state equations that

describe the steady state behavior of the system.

2.3. Previous Work

Queueing theory [COX 61, KLEI 75, SAAT 61, TAKA 62] has provided the basic framework for stochastic models that have been successfully used in evaluating alternative scheduling policies in time-sharing systems [BABA 75, COFF 68, COFF 73, KLEI 70A, McKI 69], in predicting the performance of memory subsystems in multiprogramming systems [COFF 73, FULL 75], in understanding the interaction of CPU and I/O processors [COFF 73, HOFR 75, SENC 73], and in designing communication subnetworks [FRAN 72, KLEI 70B, McGR 75]. The early queueing theory results for networks of queues [GORD 67, JACK 57, JACK 63] have been generalized to include a wider range of computer system models [BASK 75, REIS 75].

The desire to model the performance of computer systems effectively has also generated interest in other modeling techniques. Kobayashi [KOBA 73] applied the diffusion method to the analysis of queueing networks. Stone [STON 77] formulated the problem of scheduling jobs to processors as a combinatorial optimization problem. Buzen [BUZE 76] took the operational approach to the measurement and evaluation of real systems. Various approximation techniques have also been introduced to analyze the behavior of some systems [MUNT 74, GELE 74]. Although significant progress has been made in the modeling and analysis of computer system performance, much more needs to be done before we can adequately

deal with the complexities of multiple-processor distributed systems.

The existing literature in the modeling of computer systems is rich. However, the specific problem unique to distributed computer systems, state dependent job routing, seldomly has been addressed. Koenigsberg [KOEN 66] first suggested and analyzed a join-the-shorter-queue policy for two-processor systems with instantaneous jockeying between queues. Using the transformation method, Flatto [FLAT 76] analyzed a homogeneous two-processor model with a join-the-shorter-queue job arrival policy. Towsley [TOWS 75] investigated queueing models with job branching probabilities that belong to a special class of state dependent functions. Kohler [KOHL 75] introduced more sophisticated state dependent routing strategies for distributed computer systems. In this dissertation we extend the above work to cover a wider range of job scheduling policies in multiple-processor computer systems.

As a by-product of the interest in modeling computer systems, much effort has been devoted to the development of efficient analytical and numerical solution methods. Transformation (Laplace transforms, Z-transform, and etc.) is a traditional technique in the analysis of computer systems [KLEI 75, KOEN 66, NAKA 71]. Wallace [WALL 66] elaborated the power iteration method for computing the steady state probabilities for queueing models. Jackson [JACK 63] first suggested the product form solution for open models with exponential service distributions. Gordon and Newell [GORD 67] extended this result to

closed queueing models with queue length dependent service distributions. Chandy [CHAN 72] introduced the concept of local balance and showed that networks with certain queueing disciplines satisfy local balance and have product from solutions. Baskett, Chandy, Muntz, and Palacios [BASK 75] extended these results to a broader class of queueing models. Buzen [BUZE 73] developed efficient algorithm for computing the normalizing constant for steady state probabilities that have product from solutions. Muntz and Wong [WUNT 74] introduced a recursive solution technique which drastically reduced the complexity of computing the equilibrium state probabilities of some queueing models. Part of the effort in our research is also devoted to extending the existing solution techniques for use in the analysis of distributed computer systems.

CHAPTER III

A DCS MODEL

In this chapter a simple DCS model will be formulated. This model reflects many design objectives of distributed computer systems. Among these implementation objectives are load balancing, resource sharing, and reliability of the system. The mathematical representation of the DCS model is described. Various solution methods are then discussed to demonstrate the limitations of current modeling techniques for the proposed DCS model.

3.1 Proposed Model

Consider a simple central server system as in Figure 3-1. The model

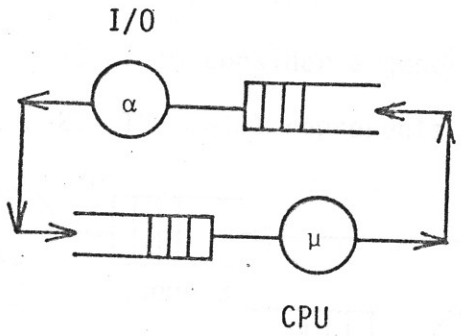


Figure 3-1: A Simple Central Server System.

represents a basic closed queueing network in which servers are denoted by circles and queues are denoted by rectangles. It is assumed that the queueing disciplines are both first-come-first-served (FCFS) and the service time distributions of CPU and I/O are exponential with means $1/\mu$ and $1/\alpha$ respectively. There are a fixed number of jobs in the system. A job is transferred to the I/O upon its completion of service at the CPU. This

model may be interpreted as an example of the central server model for a multiprogramming system with fixed degree of multiprogramming [ADIR 72, BUZE 73, KLEI 76].

Several questions arise when we consider putting two or more such multiprogramming systems together to form a DCS for the purpose of sharing resources (for example, sharing the I/O). Among them are:

1. How can these systems be interconnected to form a DCS?
2. How can the jobs in the DCS be distributed if we allow each job to be executed on any CPU?
3. How will the performance of the DCS compare with the individual systems?
4. How can we improve the total system performance by altering the interconnection structure and/or the workload distribution policy of the system?

These questions lead us to consider a general two-processor DCS modes as shown in Figure 3-2. The model represents two interconnected central

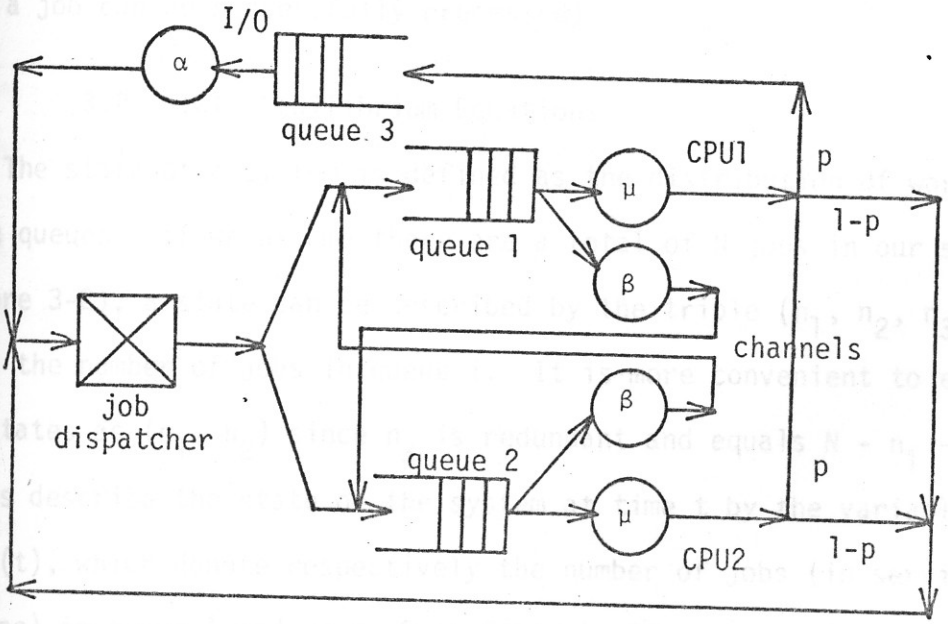


Figure 3-2: A Two-Processor DCS.

server systems with shared I/O. The job dispatcher serves to schedule the jobs to both CPU's. The scheduling policies may be state independent, such as random scheduling and fixed probability scheduling, or state dependent, such as a join-the-shorter-queue policy. Other scheduling policies may be job-dependent (e.g. special classes of jobs are always scheduled on a particular CPU). The channels serve as job transfer devices and can be viewed as processors. The job transfer time distributions are also assumed to be exponential with means $1/\beta$. The transfer policies can be either state dependent or state independent as in the job dispatcher case. The job dispatcher and channels are added to the system for two reasons: first, to balance the work load; and second, to satisfy particular applications of the DCS. This DCS fits our definition of a distributed computer system as described in Chapter II. The design is motivated by the objectives of resource sharing, load balancing, and reliability (in this case, the probability that a job can be successfully processed).

3.2 State Equilibrium Equations

The state of a system is defined as the distribution of workload among queues. If we assume there are a total of N jobs in our system (Figure 3-2), a state can be described by the triple (n_1, n_2, n_3), where n_i is the number of jobs in queue i. It is more convenient to express the states as (n_1, n_2) since n_3 is redundant and equals $N - n_1 - n_2$. Let us describe the state of the system at time t by the variables X(t) and Y(t), which denote respectively the number of jobs (in service and waiting) in queue 1 and queue 2 at time t. If we now define

$$P_{ij}(t) \triangleq \Pr[X(t) = i,\ Y(t) = j)],$$

Then because we have assumed independent exponential interarrival and service time distributions, the state is a Markov process and the state transition equations can be written as

$$\begin{aligned}\frac{d}{dt} P_{ij}(t) = &- (\alpha + \mu_{ij} + \beta_{ij})\, P_{ij}(t) \\ &+ \alpha_{i-1,j}\, P_{i-1,j}(t) + \alpha_{i,j-1}\, P_{i,j-1}(t) \\ &+ \mu\, P_{i+1,j}(t) + \mu\, P_{i,j+1} \\ &+ \beta_{i+1,j-1}\, P_{i+1,j-1}(t) + \beta_{i-1,j+1}\, P_{i-1,j+1}(t)\end{aligned} \tag{3-1}$$

where

$$\mu_{ij} \triangleq \begin{cases} 0 & i,j = 0 \\ \mu & i \geq 1,\ j = 0 \quad \text{or} \quad i = 0,\ j \geq 1 \\ 2\mu & i \geq 1,\ j \geq 1 \end{cases}$$

$$P_{ij}(t) \triangleq 0 \qquad \text{whenever } i \text{ or } j < 0 \quad \text{or} \quad i + j > N$$

and α_{ij}, β_{ij} are dependent upon the job scheduling and transfer policies respectively.

We will usually be interested in the behavior of the system during statistical equilibrium (steady-state) and in this case we will let

$$P_{ij} \triangleq \lim_{t\to\infty} P_{ij}(t)$$

denote the stationary probabilities whenever they exist. The equilibrium equations are obtained from (3-1) by setting $\frac{d}{dt} P_{ij}(t) = 0$ and substituting the appropriate stationary probabilities on the right-hand side. Thus, we

get the set of equations

$$(\alpha + \mu_{ij} + \beta_{ij})\, P_{ij} = \alpha_{i-1,j}\, P_{i-1,j} + \alpha_{i,j-1}\, P_{i,j-1} + \mu\, P_{i+1,j} + \mu\, P_{i,j+1} + \beta_{i+1,j-1}\, P_{i+1,j-1} + \beta_{i-1,j+1}\, P_{j-1,j+1} \qquad (3\text{-}2)$$

for states (i,j) which fully describe the steady state behavior of the system.

3.3 Parameterization of the Job Routing Policies

Jobs in the system are routed through the job dispatcher and the transfer channels to achieve load balancing. We now consider a special case of state dependent job routing. The routing rules are defined as follows:

1. The dispatcher assigns an arriving job to the shorter queue;
2. If both queues have an equal number of jobs (including empty), the arriving job is dispatched randomly to one of the two queues with equal probabilities;
3. The communication channels initiate a job transfer from queue i to queue j whenever the number of jobs in queue i is two or more greater than the number of jobs in queue j. The channels can only service one job at a time;
4. The transfer of a job is discontinued if the imbalance condition in (3) changes before the channels complete the transfer.

These routing rules imply that

$$\alpha_{ij} = \begin{cases} 0 & i > j \\ \alpha/2 & i = j \\ \alpha & i < j \end{cases}$$

and

$$\beta_{ij} = \begin{cases} 0 & |i-j| \leq 1 \\ \beta & |i-j| > 1. \end{cases}$$

3.4 State-Transition-Rate Diagram

The behavior of the system can be equivalently described by means of the state-transition-rate diagram shown in Figure 3-3. Here the (i,j)th

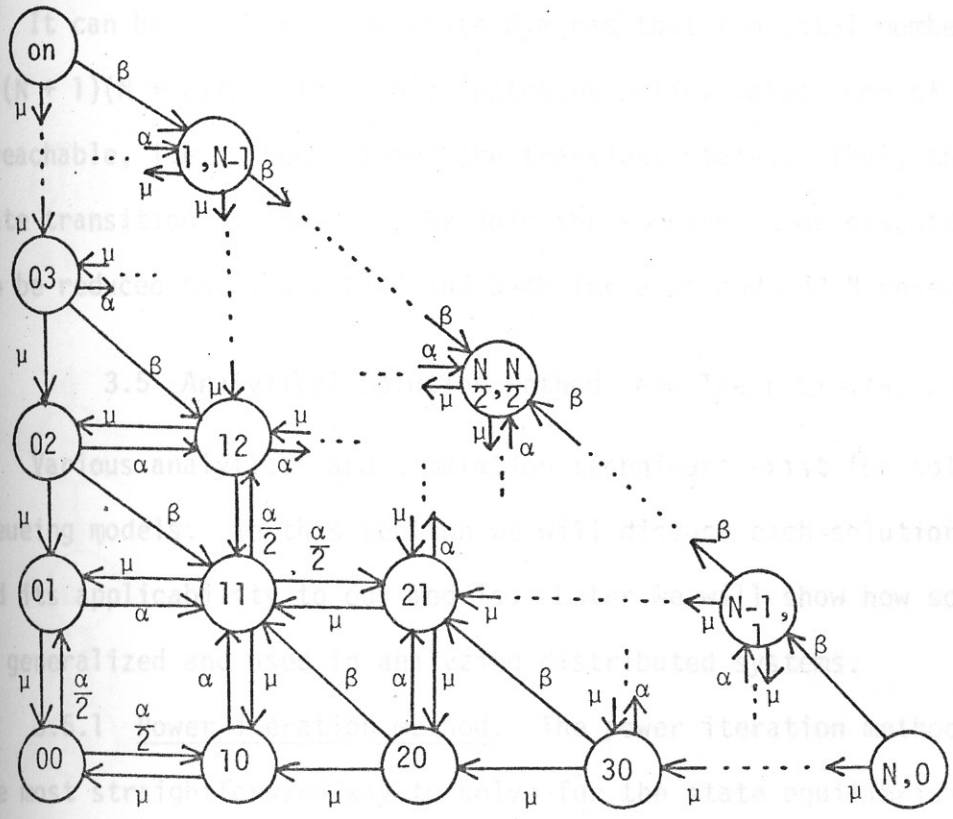


Figure 3-3: State-Transition-Rate Diagram.

node represents the state X(t) = i, Y(t) =j. The branches identify the allowable state transitions and the branch labels specify the rate at which the transitions proceed. The state transition equations (as in (3-1)) can be written immediately by noting that the rate of change of flow into state (i,j), i.e. $\frac{d}{dt} P_{ij}(t)$, must be equal to the difference between the rate at which the system enters state (i,j) and the rate at which the system leaves state (i,j). The state equilibrium equation obtained by equating the total flow out of the state to the total flow into the state is called the global balance equation (GBE) for the state. Performance measures of the system can be computed if the state equilibrium probabilities P_{ij} are known.

It can be seen from the state diagram that the total number of states is (N + 1)(N + 2)/2. The job dispatching policy makes some of the states unreachable, i.e., these states are transient states. Thus, the steady state transition diagram for the join-the-shorter queue dispatching policy can be reduced to Figure 3-4A and 3-4B for even and odd N respectively.

3.5 Analytical Solution Methods and Their Limitations

Various analytical and simulation techniques exist for solving queueing models. In this section we will discuss each solution method and its applicability to our models. Later we will show how some can be generalized and used in analyzing distributed systems.

3.5.1 Power iteration method. The power iteration method is perhaps the most straightforward way to solve for the state equilibrium probabilities. It was first used by Wallace [WALL 66] for the analysis of computer systems. Let A be the matrix of transition rates for the set of simulatan-

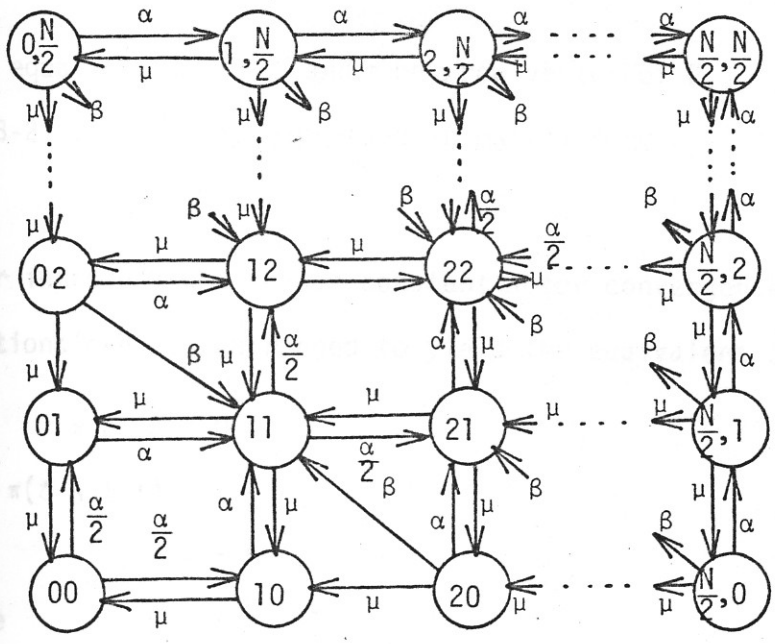


Figure 3-4A: State-Transition-Rate Diagram for Join-the-Shorter-Queue Policy (N = even).

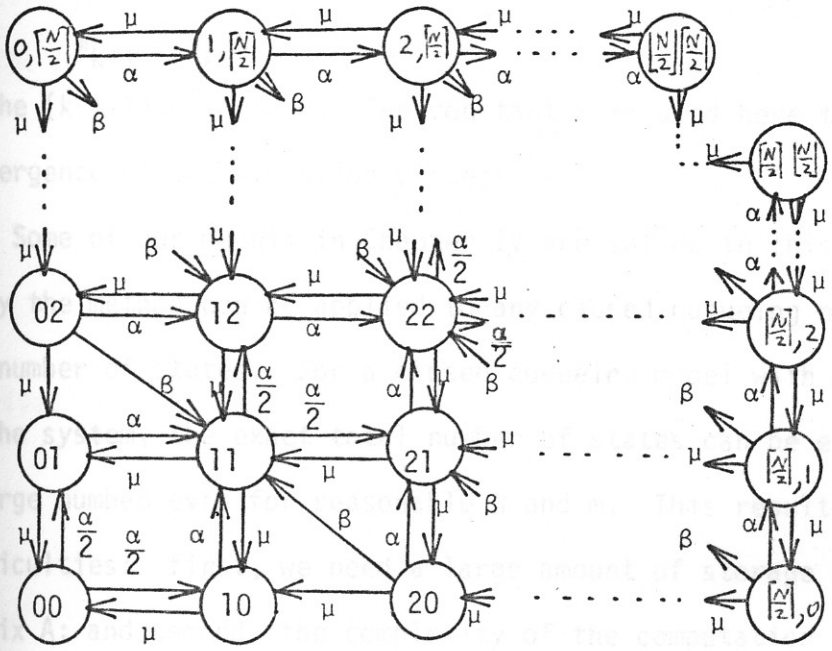


Figure 3-4B: State-Transition-Rate Diagram for Join-the-Shorter Queue Policy (N = odd).

eous equations in (3-2) and π be a row vector of P_{ij}. The system equations in (3-2) can then be expressed in matrix form as

$$\pi A = 0$$

After introducing a scalar constant Δ for convergence purposes, the equations can be rearranged to yield the equivalent system

$$\pi \Delta A = 0$$

$$\pi(\Delta A + I) = \pi$$

$$\pi = \pi Q$$

where $Q = \Delta A + I$

and I is the identity matrix.

The power iteration method is used to calculate a solution π. If π_k is the kth iterate, then

$$\pi_{k+1} = \pi_k Q$$

is the $(k + 1)$th iterate. The constant Δ is used here to ensure proper convergence of the iteration process.

Some of our models in Chapter IV are solved in this way. Theoretically the method can be applied to any closed queueing network with a finite number of states. For a closed queueing model with N jobs and m queues in the system, the exact total number of states can be expressed as $\binom{N+m-1}{m-1}$, a large number even for reasonable N and m. This results in two potential difficulties: first, we need a large amount of storage to represent the matrix A; and second, the complexity of the computation increases exponentially as N and m increase, Since the states in a system have transitions

to their neighboring states only, the matrix A is usually a sparse matrix with many identical elements. Techniques to optimize the storage representation of A can be developed [WALL 66]. However,the power iteration method is still limited to small and specific queueing models [CHOW 76].

3.5.2 Transformations. Transformation methods (transforms) have been used in the study of many interesting physical systems. As mentioned earlier the behavior of a queueing system can be described by the set of state equilibrium equations. The number of such equations is usually large and in an open system it is infinite. By defining a proper generating function, which is a function of the state equilibrium probabilities, we can map the large set of simultaneous equations into a much simpler form. If the solution of the generating function and its inverse transform exist, we can derive expressions for the state equilibrium probabilities from the generating function. This process will be illustrated for the open queueing model shown in Figure 3-5. The model represents two

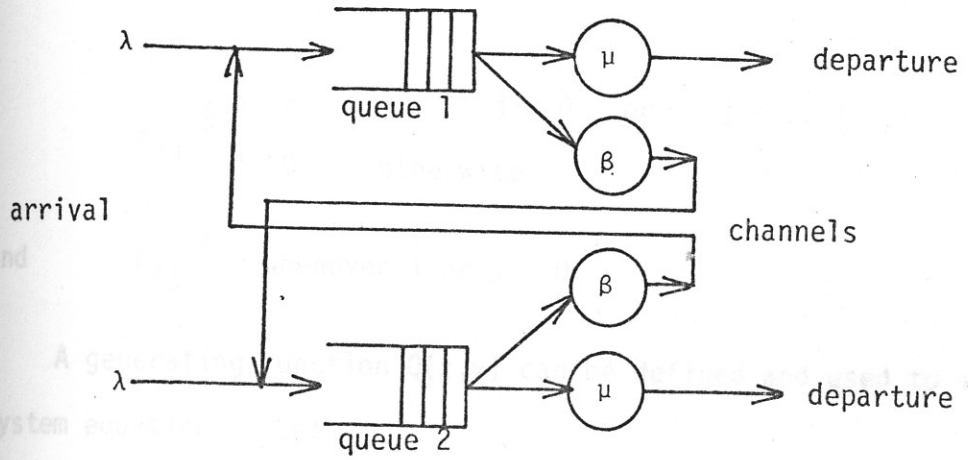


Figure 3-5: An Open Queueing Model with State-Dependent Job Transfers.

CPU's sharing the load through a communication channel. All distributions are assumed exponential and the queueing disciplines are FCFS. The channel is used to transfer a job from queue m to queue n, but the transfer proceeds only if queue m contains more than one job and queue n is empty. The transfer is discontinued if this condition changes before the channel completes the transfer. The state equilibrium equations can be written as

$$\begin{aligned}(2\lambda + \mu_{ij} + \beta_{ij})P_{ij} = {} & \lambda(P_{i-1,j} + P_{i,j-1}) \\ & + \mu(P_{i+1,j} + P_{i,j+1}) \\ & + \beta_{i-1,j+1}\, P_{i-1,j+1} \\ & + \beta_{i+1,j-1}\, P_{i+1,j-1}\end{aligned} \tag{3-3}$$

where

$$\mu_{ij} \triangleq \begin{cases} 0 & i,j = 0 \\ \mu & i \geq 1,\ j = 0 \quad \text{or} \quad i = 0,\ j \geq 1 \\ 2\mu & i \geq 1,\ j \geq 1 \end{cases}.$$

$$\beta_{ij} \triangleq \begin{cases} \beta & i \geq 1,\ j = 0 \quad \text{or} \quad j = 0,\ i > 1 \\ 0 & \text{otherwise} \end{cases}$$

and $P_{ij} \triangleq 0$ whenever i or $j < 0$.

A generating function $Q(x,y)$ can be defined and used to simplify the system equations. Let

$$Q(x,y) \triangleq E[x^i y^j] = \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} P_{ij}\, x^i y^j \tag{3-4}$$

where $x,y < 1$ and E represents the mathematical expectation function.

From the definition of $Q(x,y)$ it can be seen that

$$P_{ij} = \left[\frac{\partial^{i+j} Q(x,y)}{\partial x^i \, \partial y^j} / i! \; j!\right]_{\substack{x=0 \\ y=0}} \tag{3-5}$$

if the derivative exists. We also notice that

$$Q(0,0) = P_{00},$$

$$Q(1,1) = 1,$$

$$Q(0,1) = \sum_{j=0}^{\infty} P_{0j}$$

$$= \Pr[\text{CPU 1 is idle}],$$

$$Q(1,0) = \sum_{i=0}^{\infty} P_{i0}$$

$$= \Pr[\text{CPU 2 is idle}],$$

and

$$Q_x(1,1) = \left[\frac{\partial}{\partial x} Q(x,y)\right]_{x=y=1}$$

$$= \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} i \, P_{ij}$$

$$= \text{average queue length of queue 1},$$

$$Q_y(1,1) = \left[\frac{\partial}{\partial y} Q(x,y)\right]_{x=y=1}$$

$$= \sum_{i=0}^{\infty} \sum_{j=0}^{\infty} i \, P_{ij}$$

$$= \text{average queue length of queue 2}.$$

Given the definition of $Q(x,y)$ in (3-4) we may multiply the equations for P_{ij} in (3-3) by $x^i y^j$ and sum over all values of i and j. We therefore obtain, after collecting terms, the following simplified equation for $Q(x,y)$.

$$\begin{aligned} 0 = {} & [-2xy(\lambda + \mu) + \lambda xy(x + y) + \mu(x + y)]\, Q(x,y) \\ & + [\mu x(y-1) + \beta y(y-x)]\, Q(x,0) \\ & + [\mu y(x-1) + \beta x(x-y)]\, Q(0,y) \\ & - \beta(x-y)^2\, Q(0,0) \end{aligned} \tag{3-6}$$

The remaining job is to find an expression for $Q(x,y)$ as a function of x and y. Unfortunately there exists no general approach for solving implicit equations of this kind which involve more than one variable. However, we do know the behavior of the system for two limiting cases:

1. $\lim \beta \to 0$ gives us two independent systems; and
2. $\lim \beta \to \infty$ gives us two processors sharing a common queue.

In the first case we have two independent M/M/1 queues and

$$Q(x,y) = \frac{(1-\frac{\lambda}{\mu})^2}{(1-\frac{\lambda}{\mu}x)(1-\frac{\lambda}{\mu}y)}$$

is the solution to (3-6). P_{ij} can be derived using equation (3-5) and becomes

$$P_{ij} = (1 - \frac{\lambda}{\mu})^2 (\frac{\lambda}{\mu})^i (\frac{\lambda}{\mu})^j \ ,$$

while in the second case we have an M/M/2 queue since we can view the system as two processors sharing a same queue due to the instantaneous

channel transfer between queues. There exists standard solutions for M/M/2 [KLEI 75].

Various transformation techniques (e.g. z-transform, Laplace transform, etc.) can be used for the analysis of queueing models. In particular, we have shown the z-transform method for the distributed example above. Although the method seems simple, and if successful yields closed form solutions, it has some major deficiencies:

1. The definition of the generating function is not unique;
2. There is no guarantee that the system equations can be simplified through the defined generating function;
3. There is no general approach to solve for the generating function with two or more variables;
4. The resulting generating function may not be rational and thus the inverse transform does not exist.

3.5.3 Product form solutions. Product form solutions for state equilibrium probabilities exist for many classes of queueing networks. A product form solution of a state means that the joint state probability can be expressed as the product of the marginal state probabilities and a normalizing constant. That is

$$P(n_1,n_2,\ldots,n_m) = C \times P(n_1) \times P(n_2) \times \ldots P(n_m)$$

where n_i is the number of jobs in queue i and C is the normalizing constant. Since marginal state probabilities are easier to compute, this product form allows computationally efficient analysis of large queueing networks.

The product form solution was first suggested by Jackson [JACK 63] for open models with exponential service distributions. Gordon and Newell [GORD 67] extended this result to closed queueing models with state dependent service distributions. Chandy [CHAN 72] introduced the concept of local balance and showed that networks with certain queueing disciplines (FCFS, PS, LCFSPR) that satisfy local balance have product form solutions. Baskett, Chandy, Muntz, and Palacios [BASK 75] further generalized these results to networks with multiple classes of customers. Buzen [BUZE 73] developed efficient algorithms for computing the normalizing constant. Towsley [TOWS75] introduced station balance that explains why certain queueing disciplines yield to product form solutions for queueing models with nonexponential service time distributions.

Product form solutions imply a certain degree of independence among queues. Unfortunately, many models that confront us have state dependent routing policies. This usually results in a different set of states, since some states are not reachable and the local balance property does not hold in these models. The method is not applicable for the class of distributed computer systems that we have investigated.

3.5.4 Norton's theorem. Often in the study of queueing network models, one is interested only in the statistics of a particular queue. It is thus desirable to construct an equivalent queueing network in which all the queues except the one of interest are replaced by a single composite queue. This concept and technique was developed by Chandy, Herzog, and Woo [CHAN 75]. It is called the Norton theorem of queueing theory due to its analogy with the Norton theorem of electrical circuit

theory. This method can be illustrated by the follwoing example in which we are interested in analyzing the CPU queue of a central server model with two I/O's (Figure 3-6).

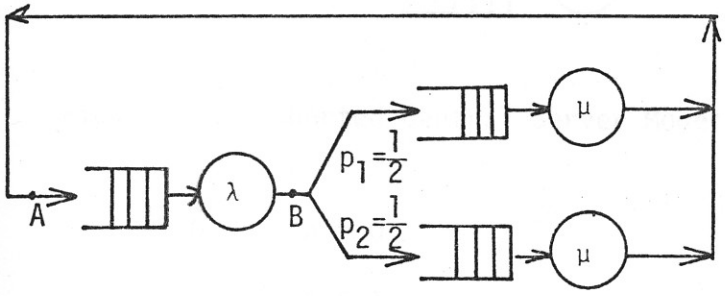


Figure 3-6: Central Server Model.

We wish to replace this model with an equivalent network (Figure 3-7)

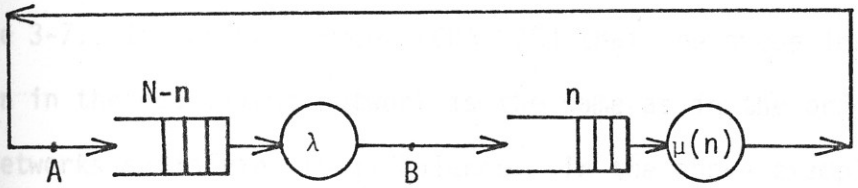


Figure 3-7: Equivalent Central Server Model.

in which the statistics for the CPU in the equivalent model are the same as in the orginal system. The equivalent service rate $\mu(n)$ of the composite queue is a function of n, the composite queue length. This new service rate can be computed and found to be equal to the throughput along AB in Figure 3-8 when AB is shorted. That is,

$$\mu(n) \triangleq \text{Throughput}_{AB} \text{ (n jobs in the system).}$$

Given $\mu = 0.5$ and $N = 3$ we get

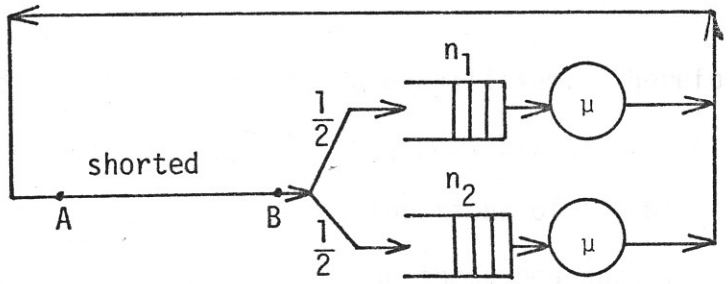


Figure 3-8: Shorted Central Server Model.

$$\mu(n) = \begin{cases} 0 & n = 0 \\ 1/2 & n = 1 \\ 2/3 & n = 2 \\ 3/4 & n = 3 \end{cases}$$

CPU statistics can be more effectively computed from the model in Figure 3-7. It has been shown [CHAN 75] that the queue length distribution in the equivalent network is the same as in the original network for networks satisfying local balance. In the above example the branching probabilities are constants (1/2). It can be seen that the throughput along AB in Figure 3-8 can not be derived if we choose a join-the-shorter-queue branching policy, since we will get only one possible state (n_1, n_2) where $|n_1-n_2| \leq 1$ and $n_1 + n_2 = n$. Throughput $\mu(n)$ will be intractable. however, the Norton theorem is still valid for some models with a special class of stochastic state dependent branching probabilities [TOWS 75]. In these cases the local balance property happens to exist.

3.5.5 Recursive method. The recursive solution technique was introduced by Herzog, Woo, and Chandy [HERZ 75]. The main idea of the method is that the state probabilities of a system can be expressed as linear

combinations of other state probabilities. Therefore by properly choosing some boundary states, it is possible to derive expresseions for all other state probabilities as functions of the boundary values. The reduced system of equations for these boundaries can then be solved and finally all system states can be evaluated and normalized. The set of chosen boundary states will affect the complexity of the computation and is dependent upon the structure of the state diagram. This technique is significantly different from traditional solution techniques for queueing models in the sense that it manipulates the state diagram regardless of the orginal characteristics (distributions, queueing disciplines, routing strategies) of the system. We will demonstrate the method with a simple closed central server model (Figure 3-9) in which one server has an

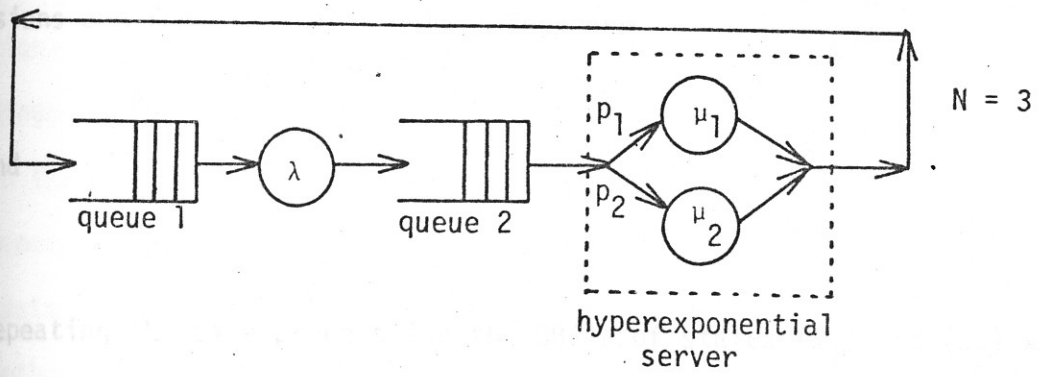


Figure 3-9: Central Server Model with DMP = 3.

exponential service time distribution with mean $1/\lambda$, the other server has a hyperexponential service time distribution with mean $(\frac{P_1}{\mu_1} + \frac{P_2}{\mu_2})$, and the degree of multiprogramming is three. The system state can be defined as (n_s) where n is the number of jobs in queue 2 and s is either 1 or 2 depending on which server is being used. The state diagram is

shown in Figure 3-10. By choosing $P_3 = 1$ (arbitrary constant) as the

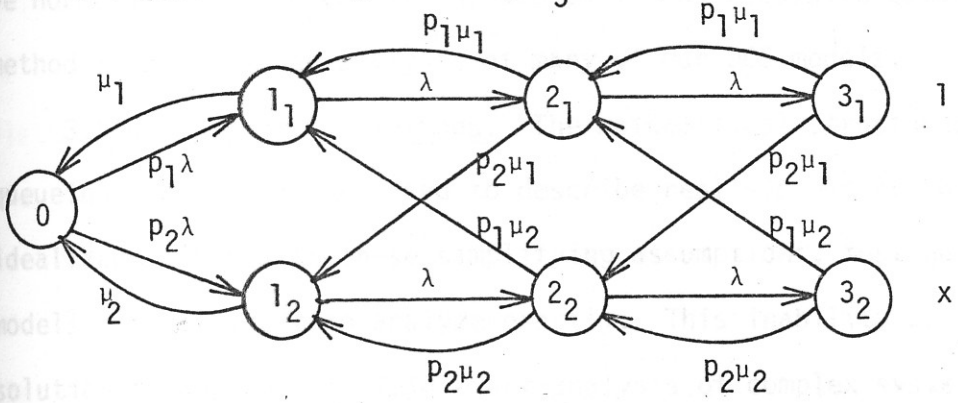


Figure 3-10: State Diagram for Figure 3-9.

prenormalized probability for boundary state 3_1 and $P_{3_2} = x$ (unknown) as the prenormalized probability for boundary state 3_2, the GBE's for state (3_1) and (3_2) can be used to derive expressions for state probabilities P_{2_1} and P_{2_2} as linear combinations of P_{3_1} and P_{3_2}. These expressions are functions of x only. They are

$$P_{2_1} = \frac{\mu_1}{\lambda}$$

and

$$P_{2_2} = \frac{\mu_2}{\lambda} x.$$

Repeating the same process for the GBE's of states (2_1) and (2_2) we obtain

$$P_{1_1} = \frac{1}{\lambda} [(\lambda+\mu_1) \frac{\mu_1}{\lambda} - p_1\mu_1 - p_1\mu_2 x]$$

and $$P_{1_2} = \frac{1}{\lambda} [(\lambda+\mu 2) \frac{\mu_2}{\lambda} x - p_2\mu_1 - p_2\mu_2 x].$$

Finally, P_0 can be expressed as a linear function of x by using the GBE for state (1_1) and x can be evaluated from the GBE of state (1_2). The initial assumption that $P_{3_1} = 1$ is arbitrary since all probabilities will

be normalized at the end of the process. The recursive solution method is used in the analysis of many of our DCS models.

3.5.6 Appromation methods. The mathematical structures of queueing theory that are used to describe real systems are often idealized. Even with these simplifying assumptions, most queueing models are difficult to analyze exactly. This inability of existing solution techniques to yield exact analysis of complex system models raises the important question of approximations. It is reasonable to find approximate or bounding behavior for real systems.

Approximation methods have long been used in various aspects of queueing models. Erlang and Cox [COX 55] used the staging method to approximate the distributions of random processes. This staging method is employed in the later sections of our work. The heavy traffic condition of G/G/1 was studied by Kingman [KING 62] and Kobayashi [KOBA 73]. Kingman developed upper and lower bounds for the average waiting time in the system. Kobayashi used the fluid approximation (diffusion process) to describe fairly adequately the behavior of single server systems under heavy load and saturated conditions. Approximation methods for queueing networks have not gone unnoticed [MUNT 74, CHAN 75, SAUE 75]. Muntz and Wong [MUNT 74] studied the asymptotic properties of resource utilization and mean response time for a general network model of a time-sharing system. System saturation points can be estimated from the approximate asymptotic characteristics of the systems. Chandy, Herzog, Sauer, and Woo [CHAN 75, SAUE 75] used Norton's theorem to approximate queueing networks that do not have local balance properties. They demonstrated that queueing networks with FCFS queueing

disciplines and non-exponential service distributions can be approximated by first finding an equivalent model under the assumption that all service distributions are exponential and then solving the simplified model using the approximate service distribution with equivalent mean and coefficient of variation for the reduced server.

Although there is no general approximation technique to cover all classes of queueing models, it is our belief that approximation methods will provide fruitful results in modeling system behavior. Our study of DCS models uses both exact and approximate approaches.

3.5.7 Simulation. Simulation has long been a useful tool for performance evaluation in computer systems [BOWD 73, MCDO 70]. It is often used in verifying analytical results and sometimes is the only way to approach system modeling due to the complexity of the analytical analysis. Various simulation techniques [BOWD 73, KLEI 69] exist for performance prediction and analysis of computer networks. Theories and developments [LAVE 75] are also available for estimating confidence intervals when simulating stochastic system that have a regenerative structure (a regenerative point is an instance of a simulation which can be considered as a pseudo initial condition of the simulation). Simulations will be used in our analysis of DCS models when analytical solutions are intractable.

CHAPTER IV

HOMOGENEOUS SYSTEMS

4.1 Introduction

In recent years much attention has been focused on the development of homogeneous multiple-processor computer systems [FARB 73, MANN 76]. This interest is motivated by the desire to share resources and the need to achieve higher system performance and reliability. In this chapter we will investigate the possibility of replacing a single central processor with multiple slow processors in a closed central server model. The results indicate that multiple slow processors may sometimes be used to replace a fast central processor without significant performance degradation. We will also study the problem of scheduling of jobs among homogeneous processors. The objective of the scheduling of jobs is to achieve system balance, with a resultant performance increase, by automatically shifting jobs from heavily loaded processors to lightly loaded processors in the system. This load balancing can be done statically at system configuration time by preassigning certain jobs or classes of jobs to specific processors, or dynamically as the load and state of the system changes by automatically transferring jobs from one processor to another. The analysis shows that simple load balancing policies can significantly improve system performance.

4.2 Closed Models for Multiple-Processor Central Server Systems

We are interested in what is to be gained (or lost) by using multiple slow processors to replace a single, fast central processor in a

multiple-processor computer system. This is illustrated by the analysis of serveral multiple processor models. We consider again the basic central server model as in Figure 3-1 except with the modification of a feedback path as shown in Figure 4-1. If we interpret the model as a multiprogramming system with fixed degree of multiporgramming, then service center 1

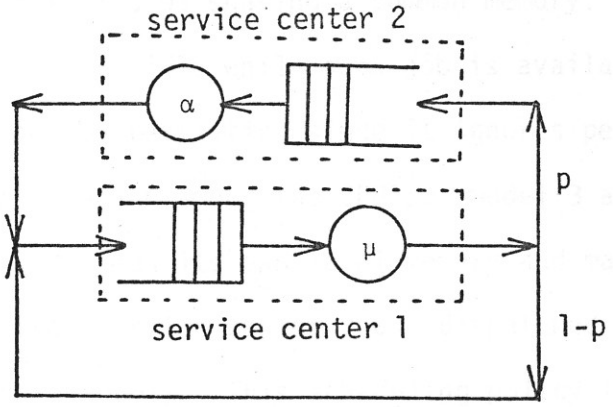


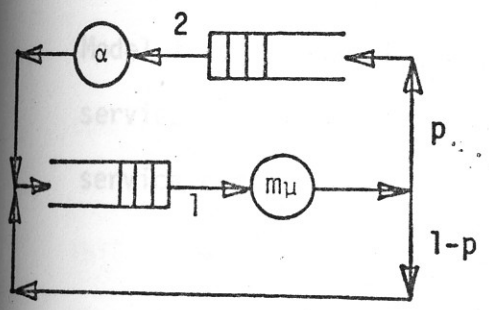
Figure 4-1: Basic Central Server System.

represents the central processing unit (CPU) where computation are performed, and service center 2 represents the input/output (I/O) unit. Each job residing in main memory is in one of four states: waiting for the CPU, computing, waiting for I/O, and performing I/O activity. A job cycles through these four states until completed. The parameter p is the probability that a job does not complete after its current service by the CPU. When a job does complete, a new job immediately enters service center 1. We present five variations of the model each with equivalent maximum processing power. Their job turnaround times and job throughput rates are then analyzed and compared. This analysis is described in sections 4.2.1 to 4.2.7.

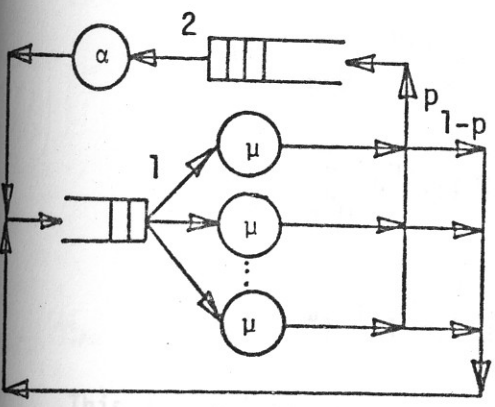
4.2.1 System Models and Analysis

Each of the five models is shown in Figure 4-2. Model 1 is the single fast CPU with a processing rate of $m\mu$, i.e., the mean service time on the CPU is $1/m\mu$. The speed factor m is introduced to compare this model with the other four which incorporate m parallel CPU's each of rate μ. Model 2 represents m parallel CPU's sharing a common memory. The common queue ensures that no CPU is idle while some job is available for processing. This model may not be realistic, since it ignores performance degradation due to memory contention among the CPU's. Model 3 assumes that each of the m parallel CPU's has its own local memory and may be multiprogrammed. Jobs returning from service center 2 are dispatched to the shortest queue, with ties broken randomly. This scheduling policy tends to balance the load, but the model ignores the time required to switch a job from one CPU to another. Model 4 is similar except that jobs do not switch CPU's. In this case we introduce m different classes of jobs. Although all jobs share service center 2, class i jobs can only be processed by the ith processor in service center 1. Upon completion of a job, a new job of the same class immediately enters the network. The initial distribution of jobs is such that each class has, as nearly as possible, the same number of jobs. This assumption offers some degree of load balancing. Model 5 has the same configuration as in Model 3 and Model 4, but in this model no attempt is made to balance the load. A returning job joins any one of the m queues at random.

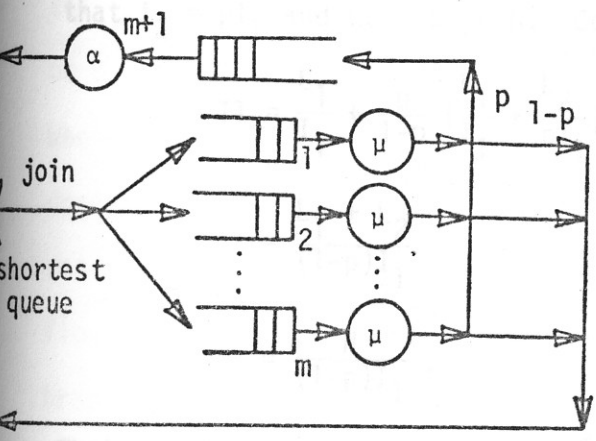
For each of the models, we can calculate the mean job turnaround time (TT) and mean job throughput rate (TP) as a function of parameters α, μ, p, m, and N, where N is the total number of jobs in the system. Consider



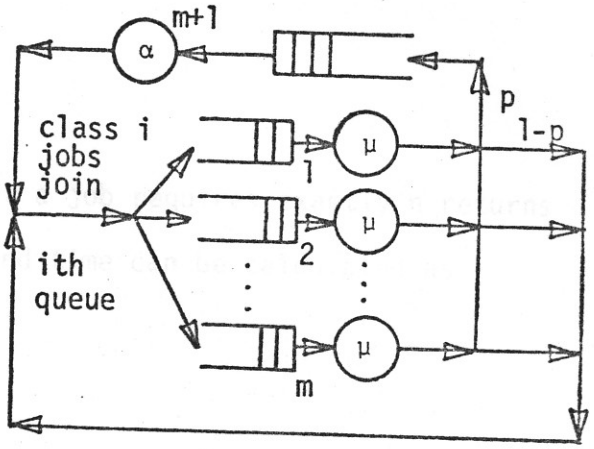
a) Model 1 - Single CPU.



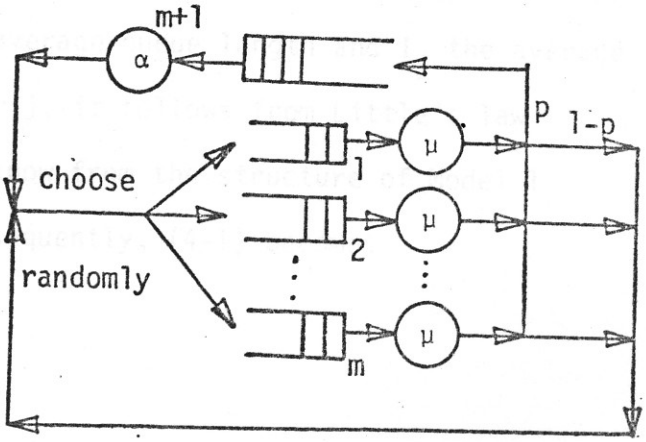
b) Model 2 - Multiple CPU's with Shared Memory.



Model 3 - Multiple CPU's with Local Memory and Load Balancing.



d) Model 4 - Multiple CPU's with Local Memory and Different Job Classes.



e) Model 5 - Multiple CPU's with Local Memory.

Figure 4-2: Five Central Server Models.

Model 1 as an example. By defining ω_j as the average waiting time in service center j, the mean time required by a job with n feedbacks through service center 2 can be expressed as

$$T(n) = \omega_1 + n(\omega_1 + \omega_2)$$

Since $p^n(1-p)$ is the probability that a job requires exactly n returns before completing, the mean turnaround time can be calculated as

$$TT = \sum_{n=0}^{\infty} p^n(1-p)\, T(n)$$

$$= \sum_{n=0}^{\infty} p^n(1-p)[\omega_1 + n(\omega_1 + \omega_2)]$$

$$= \omega_1 + \frac{p}{1-p}(\omega_1 + \omega_2) \qquad (4\text{-}1)$$

This is similar to the analysis of the interactive system persented in [NAKA 71]. With L_j defined as the average queue length and I_j the average job departure rate at service center j, it follows from Little's law [KLEI 75] that $I_j\omega_j = L_j$. We also know from the structure of Model 1 that $I_2 = pI_1$ and $L_1 + L_2 = N$. Consequently, (4-1) becomes

$$TT = \frac{L_1}{I_1} + \frac{p}{1-p}\left(\frac{L_1}{I_1} + \frac{L_2}{pI_1}\right)$$

$$= \frac{L_1 + L_2}{(1-p)I_1}$$

$$= \frac{N}{(1-p)I_1}. \qquad (4\text{-}2)$$

The mean job throughput rate (job completion rate) is equal to the traffic intensity along the (1-p) path. This can be expressed as

$$TP = (1-p)\, I_1 \tag{4-3}$$

comparing (4-2) and (4-3) we see that

$$TT * TP = N \tag{4-4}$$

This relation is similar to the Little's law ($\lambda w = L$) for open systems where TT, TP, and N correspond to w, λ, and L respectively. By defining U_1 as the utilization factor of service center 1, i.e., $U_1 = I_1/m\mu$, (4-2) and (4-3) can be reduced to

$$TT = \frac{N}{(1-p)m\mu U_1} \tag{4-5}$$

$$TP = (1-p)m\mu U_1 \tag{4-6}$$

The derivations of equations (4-2), (4-3), (4-5), and (4-6) are slightly different for other models. The details will be shown in the remainder of this section where the five different configurations of service center 1 are analyzed. For each of the five models, TT and TP can be found if either I_1 or U_1 is determined. The following analysis primarily involves solving for the equilibrium state probabilities of the models to obtain I_1 or U_1. A state is normally defined as a vector $s = <n_1, n_2, \ldots, n_i, \ldots, n_m>$, where n_i is the number of jobs in the ith queue. All models can be fully described by either a state-transition-rate diagram or a state-transition-rate matrix. Each element a_{ij} of the matrix represents the rate of job flow from state i to state j. The non-zero elements are shown as directed branches in the corresponding state-transition-rate diagram. The equilibrium state probabilities are the limiting ($t \to \infty$) state probabilities which are independent of the initial state distribution [KLEI 75].

4.2.2 Single CPU - Model 1.

This model as shown in Figure 4-2(a) has only two queues. Since the total number of jobs is fixed at N, the state s can be defined as s = <i>, where i is the number of jobs in queue 1 (service center 1). The state-transition-rate diagram is given by Figure 4-3. The state equilibrium

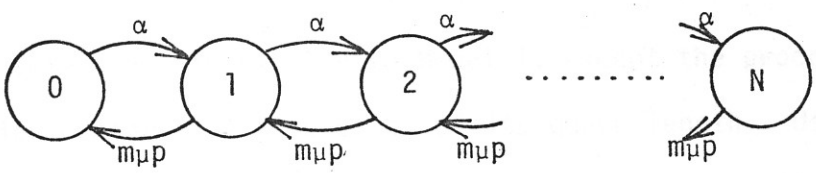


Figure 4-3: State-Transition-Rate Diagram of Model 1.

equations which form the state-transition-rate matrix can be obtained from the state-transition-rate diagram with P_i defined as the equilibrium probability of state <i>. The equations become

$$\alpha P_0 = m\mu p P_1$$

$$(\alpha + m\mu p) P_i = \alpha P_{i-1} + m\mu p P_{i+1} \qquad 0 < i < N$$

$$m\mu p P_N = \alpha P_{N-1}$$

It is well known [KLEI 75] that for the above system, the P_i's have the closed-form solution

$$P_i = \left(\frac{\alpha}{m\mu p}\right)^i P_0, \quad \text{where}$$

$$P_0 = \left[1 + \frac{\alpha}{m\mu p} + \left(\frac{\alpha}{m\mu p}\right)^2 + \ldots + \left(\frac{\alpha}{m\mu p}\right)^N\right]^{-1}$$

The utilization of service center 1, U_1, is the probability that queue 1 is not empty. Thus,

$$U_1 = 1 - P_0.$$

Using equations (4-5) and (4-6), TT and TP can be computed directly for given α, μ, m, p, and N.

4.2.3 Multiple CPU's with shared memory - Model 2.

This model is similar to Model 1, except the processing rate of service center 1 is a function of its queue length. Using the same definition of a state, s = <i>, it can be seen that the equivalent processing rate μ_e of service center 1 is

$$\mu_e = \begin{cases} i\mu , & i \leq m \\ m\mu , & i > m , \end{cases}$$

where i is the length of queue 1. Figure 4-4 is the state-transition-

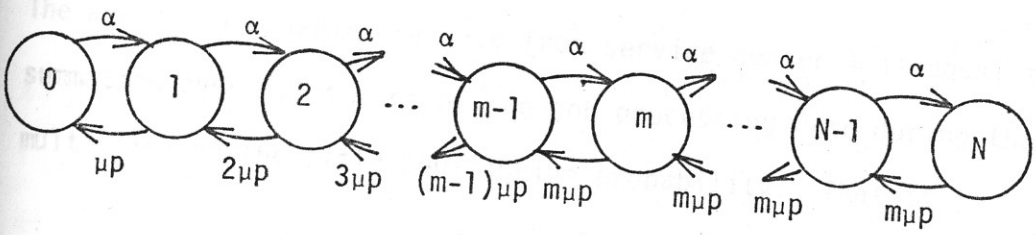


Figure 4-4: State-Transition-Rate Diagram of Model 2.

rate diagram for this model and the equilibruim equations can be expressed as

$$\alpha P_0 = \mu p P_1$$

$$(\alpha + i\mu p)\, P_i = \alpha P_{i-1} + (i+1)\mu p P_{i+1} , \quad 1 \leq i < m$$

$$(\alpha+m\mu p)P_i = \alpha P_{i-1} + m\mu p P_{i+1} \quad , \quad m \le i < N$$

$$m\mu p P_N = \alpha P_{N-1}$$

The simple M/M/m queueing system has a similar, but infinite, set of state equations [KLEI 75] and it can be easily verified that the equilibrium state probabilities are given by

$$P_i = \begin{cases} (\frac{m\rho}{i!})^i P_0 & , \quad i \le m \\ \frac{\rho^i m^m}{m!} P_0 & , \quad i > m, \text{ where } \rho = \frac{\alpha}{m\mu p} . \end{cases}$$

From the conservation relation on P_i it follows that

$$P_0 = [1 + \sum_{i=1}^{m-1} (\frac{m\rho}{i!})^i + \sum_{i=m}^{N} \frac{\rho^i m^m}{m!}]^{-1} .$$

The average job departure rate from service center 1 is equal to the summation over all states of the job processing rate during that state multiplied by the state equilibrium probability. That is

$$I_1 = \sum_{i=0}^{N} \mu_e P_i$$

$$= \sum_{i=1}^{m-1} i\mu P_i + \sum_{i=m}^{N} m\mu P_i$$

Using equations (4-2) and (4-3) with I_1 defined as above, we have an expression for TT and TP.

4.2.4 Multiple CPU's with local memory and load balancing - Model 3.

In this model (shown in Figure 4-2(c)) a state can be defined as a vector $s = \langle n_1, n_2, \ldots, n_i, \ldots, n_m \rangle$, where n_i is the number of jobs in the ith queue. n_{m+1} is redundant since $n_{m+1} = N - \sum_{i=1}^{m} n_i$. If we denote the set of all states by S then number of states $||S|| = \binom{N+m}{m}$. This result is obtained by analogy with the problem of computing the number of possible arrangements of N indistinguishable balls in m+1 cells [FELL 68]. The state-transition-rate diagram for a given state s_i takes the general form of Figure 4-5,where r_{ij} is the transition rate from state s_i to state s_j

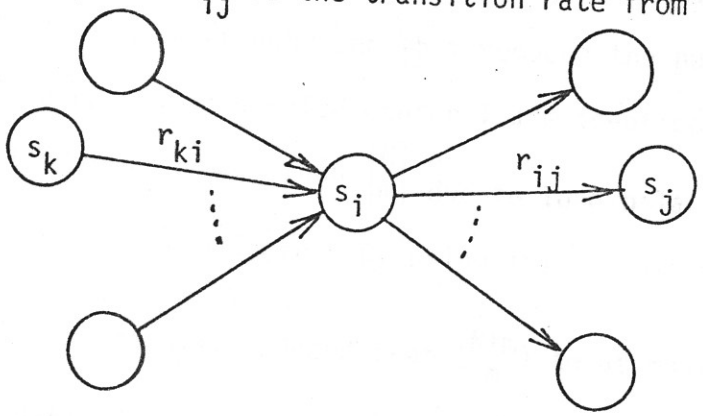


Figure 4-5: State-Transition-Rate Diagram of Model 3.

and r_{ki} is the transition rate from s_k to s_i. In equilibrium the input flow must equal the output flow from each state. Consequently, the equilibrium equations for any given state s_i are obtained by equating $\sum_j r_{ij}P(s_i)$ to $\sum_k r_{ki}P(s_k)$, where $P(s_i)$ denotes the equilibrium probability of state s_i. Since this must be true for all possible states, the system equations can be written as

$$\sum_{\substack{\text{all } j \\ r_{ij} \neq 0}} r_{ij}P(s_i) - \sum_{\substack{\text{all } k \\ r_{ki} \neq 0}} r_{ki}P(s_k) = 0 \quad , \text{ for all states } s_i. \quad (4\text{-}7)$$

It can be shown that this system does not exhibit local balance[CHAN 72]

and the equilibrium state probabilities do not have a product form solution [BASK75, GORD67]. However, the equilibrium probabilities can be numerically computed by the power iteration method [WALL66] as described in Chapter III.

If Model 3 is solved in this straightforward way, the total number of states $||S||$ is $\binom{N+m}{m}$, a large number even for reasonable N and m. The size of the matrix Q thus becomes $||S|| \times ||S||$. Since Q is normally a sparse matrix and its elements usually have the same values, special methods can sometimes be employed to reduce the required computation [WALL66]. We used a different approach, one that reduced the number of states. Since all processors in service center 1 are identical and no queue can have a waiting line longer than $\lceil \frac{N}{m} \rceil$ due to load balancing, a new scheme can be used to define a state. By using this scheme, the total number of states is drastically reduced from $\binom{N+m}{m}$ to at most $\binom{\lceil \frac{N}{m} \rceil + m}{m}$. State s is defined as $<q_0, q_1, \ldots, q_i, \ldots, q_{\lceil \frac{N}{m} \rceil}>$, where q_i denotes number of queues in service center 1 that contain exactly i jobs. The total number of states is then equal to the number of possible arrangements of m processors into $\lceil \frac{N}{m} \rceil + 1$ groups, which is given by $\binom{\lceil \frac{N}{m} \rceil + m}{m}$. Based on this definition of s, the state-transition-rate diagram of a given state $<q_0, q_1, \ldots, q_i, \ldots, q_{\lceil \frac{N}{m} \rceil}>$ is shown in Figure 4-6. For a given state it is

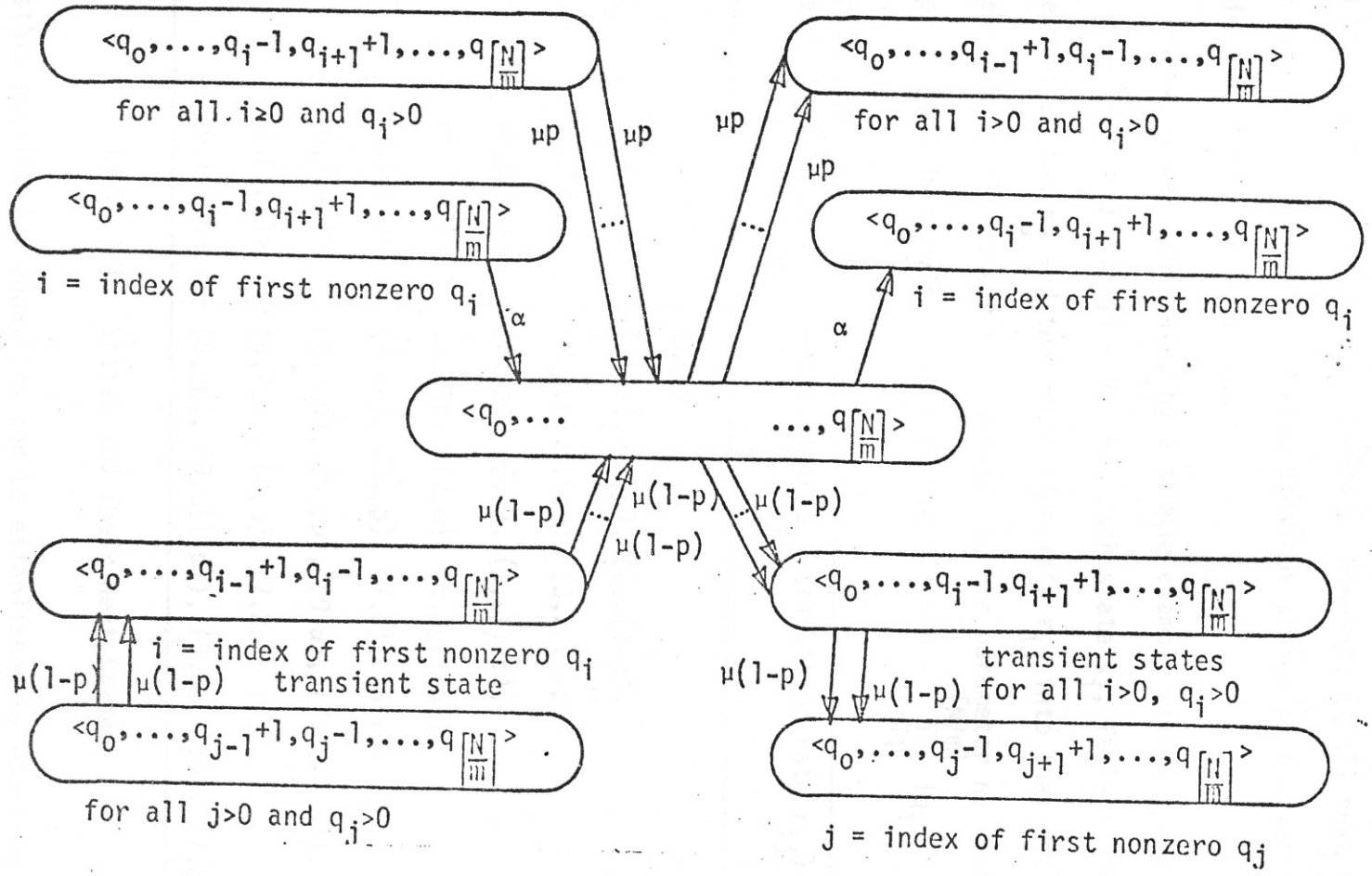


Figure 4-6: State-Transition-Rate Diagram of Model 3 with New State Definition.

easy to find the next states, but rather tedious to find the previous states from the state-transition-rate diagram. However, if all next states of each state are computable, all the previous states of a given state can be obtained by simple table lookup.

This will be illustrated by a simple example with N=5 and m=3. In this case $\left\lceil \frac{N}{m} \right\rceil = 2$ and $||S|| = 9$. For each state s_i, Table 1 lists the possible next states s_j and the transition rate r_{ij} into that state. For convenience the state vector is augmented by the parameter n, where n is the number of jobs in service center 2, i.e., $n = N - \sum_{i=1}^{\lceil N/m \rceil} iq_i$.

State s_i	State Vector $<q_0,q_1,q_2,n>$	(next state s_j, transition rate r_{ij})
1	<3,0,0,5>	(2,α)
2	<2,1,0,4>	(1,μp), (3,α), (2,μ(1-p))
3	<1,2,0,3>	(2,2μp), (5,α), (3,2μ(1-p))
4	<2,0,1,3>	(2,μp), (6,α), (3,μ(1-p))
5	<0,3,0,2>	(3,3μp), (7,α), (5,3μ(1-p))
6	<1,1,1,2>	(4,μp), (3,μp), (7,α), (6,μ(1-p)), (5,μ(1-p))
7	<0,2,1,1>	(6,2μp), (5,μp), (9,α), (7,2μ(1-p)), (7,μ(1-p))
8	<1,0,2,1>	(6,2μp), (9,α), (7,2μ(1-p))
9	<0,1,2,0>	(8,μp), (9,μ(1-p)), (9,2μ(1-p)), (7,2μp)

Table 4-1: States and Their Next States

The state equilibrium equations can be established by equating the flow out of state s_i with the flow into state s_i. For this example,

flow out = flow in

becomes

$$\sum_j r_{ij} P(s_i) = \sum_k r_{ki} P(s_k) \quad ,$$

where $$\sum_j r_{ij} = \begin{cases} \alpha + (m - q_0)\mu & , \text{ when } n \neq 0 \\ (m - q_0)\mu & , \text{ when } n = 0 \end{cases}$$

The state equations can be written down more easily if we first invert Table 4-1 to produce Table 4-2, which lists for each state s_i the previous states s_k along with the transition rate r_{ki}. Thus, the state equilibrium

State s_i	State Vector $<q_0,q_1,q_2,n>$	(previous state s_k, r_{ki})
1	<3,0,0,5>	(2,μp)
2	<2,1,0,4>	(1,α), (3,2μp), (4,μp)
3	<1,2,0,3>	(2,α), (4,μ(1-p)), (5,3μp), (6,μp)
4	<2,0,1,3>	(6,μp)
5	<0,3,0,2>	(3,α), (6,μ(1-p)), (7,μp)
6	<1,1,1,2>	(4,α), (7,2μp), (8,2μp)
7	<0,2,1,1>	(5,α), (6,α), (8,2μ(1-p)), (9,2μp)
8	<1,0,2,1>	(9,μp)
9	<0,1,2,0>	(7,α), (8,α)

Table 4-2: States and Their Previous States

equations that represent the system can be written directly from Table 4-2 and become, using P_i as a shorthand notation for $P(s_i)$

$$\alpha P_1 = \mu p\ P_2$$

$$(\alpha + \mu)P_2 = \alpha\ P_1 + 2\mu p\ P_3 + \mu p\ P_4$$

$$(\alpha + 2\mu)P_3 = \alpha\ P_2 + \mu(1-p)P_4 + 3\mu p\ P_5 + \mu p\ P_6$$

$$(\alpha + \mu)P_4 = \mu p\ P_6$$

$$(\alpha + 3\mu)P_5 + \alpha\ P_3 + \mu(1-p)P_6 + \mu p\ P_7$$

$$(\alpha + 2\mu)P_6 = \alpha\ P_4 + 2\mu p\ P_7 + 2\mu p\ P_8$$

$$(\alpha + 3\mu)P_7 = \alpha\ P_5 + \alpha\ P_6 + 2\mu(1-p)P_8 + 2\mu p\ P_9$$

$$(\alpha + 2\mu)P_8 = \mu p\ P_9$$

$$(3\mu)P_9 = \alpha\ P_7 + \alpha\ P_8\ .$$

The example above illustrates how the state equilibrium equations can be derived when m and N are known. By applying the power iteration process we can compute the equilibrium state probabilities, P(s), for a particular choice of parameter values α, μ, and p. We will now show how the job turnaround time and throughput rate can be obtained once the state probabilities are known.

Since each processor in service center 1 is identical and a join-the-shortest-queue routing policy is used to balance the queue lengths, the processor utilization, average queue length, and job departure rate of each of the m parallel processor/queue pairs must also be identical, i.e.,

$$U_1 = U_2 = \ldots = U_j = \ldots = U_m,$$

$$L_1 = L_2 = \ldots = L_j = \ldots = L_m,$$

$$I_1 = I_2 = \ldots = I_j = \ldots = I_m\ .$$

Using ω_{m+1} and L_{m+1} to denote the mean waiting time and queue length respectively in service center 2, equation (4-1) for the mean job turnaround time can be written as

$$TT = \omega_j + \frac{p}{1-p}(\omega_j + \omega_{m+1})$$

$$= \frac{L_j}{I_j} + \frac{p}{1-p}\left(\frac{L_j}{I_j} + \frac{L_{m+1}}{mpI_j}\right)$$

$$= \frac{mL_j + L_{m+1}}{(1-p)mI_j}$$

$$= \frac{N}{(1-p)mI_j}$$

Likewise, following equation (4-3) the mean job throughput rate can be expressed as

$$TP = (1-p)mI_j \ .$$

The quantity $m\,I_j$ is the mean job departure rate from service center 1. This effective processing rate is the weighted average of the processing rates of all states and can be computed from the equilibrium state probabilities P(s) as

$$mI_j = \sum_{\substack{\text{all states} \\ s=<q_0,q_1,\ldots,q_{\lceil N/m \rceil}>}} P(s) \times (\text{departure rate given state } s)$$

$$= \sum_{\substack{\text{all states} \\ s=<q_0,q_1,\ldots,q_{\lceil N/m \rceil}>}} P(s) \times \left(\sum_{i=1}^{\lceil N/m \rceil} \mu\, q_i\right) \ .$$

4.2.5 Multiple CPU's with local memory and different job classes - Model 4.

Model 4 is shown in Figure 4-2(d). For the purpose of comparison with the other models, we will assume that the N jobs are evenly distributed among the m classes. If N is a multiple of m, the distribution of jobs will be $N_1 = N_2 = \ldots = N_i = \ldots = N_m = N/m$, where N_i is the total number of class i jobs. Otherwise, there will be $N - m\lfloor\frac{N}{m}\rfloor$ classes with $\lceil\frac{N}{m}\rceil$ jobs each and $m\lceil\frac{N}{m}\rceil - N$ classes with $\lfloor\frac{N}{m}\rfloor$ jobs each. We can define a state s as the vector $\langle N_1 - n_1, N_2 - n_2, \ldots, N_i - n_i, \ldots, N_m - n_m\rangle$, ehere n_i is the number of class i jobs in service center 2 (queue m+1). $N_i - n_i$ is then the number of class i jobs in the ith queue of service center 1. The number of distinct states $||S||$ becomes $\prod_{i=1}^{m} (N_i + 1)$.

This model can be treated as a particular example of a general class of models in which the state probabilities are known to have the product from solution [BASK 75]

$$P(s' = \langle y_1, y_2, \ldots, y_{m+1}\rangle) = C \prod_{i=1}^{m+1} g_i(y_i) \qquad (4\text{-}8)$$

C is a normalizing constant and each component y_i of the new state vector s' is itself a vector $(n_{i1}, n_{i2}, \ldots, n_{ir}, \ldots, n_{im})$, where n_{ir} is the number of class r customers in the ith queue. For our example, y_i reduces to

$$y_i = \begin{cases} (0, \ldots, 0, N_i - n_i, 0, \ldots, 0), & \text{for } i = 1,2,\ldots,m \\ (n_1, n_2, \ldots, n_m) & \text{for } i = m+1 \end{cases}$$

In a closed queueing system, y_{m+1} is redundant and the definition of state s' for this model can be reduced to that of s.

Function $g_i(y_i)$ in equation (4-8) is given by [BASK 75]

$$g_i(y_i) = n_i' ! \{ \prod_{r=1}^{m} (\frac{1}{n_{ir}!})[e_{ir}]^{n_{ir}} \} (\frac{1}{\mu_i})^{n_i'} ,$$

where $n_i' = \sum_{j=1}^{m} n_{ij}$ = the total number of jobs in the ith queue

and e_{ir} = the relative arrival rate of class r jobs in the ith queue.

From Figure 4-2(c) it can be seen that

$$e_{ir} = \begin{cases} 1 , & i=r \text{ and } r=1,2,\ldots,m \\ p , & i=m+1 \text{ and } r=1,2,\ldots,m \\ 0 , & \text{otherwise.} \end{cases}$$

Therefore,

$$g_i(y_i) = (N_i-n_i)! \{ \frac{1}{(N_i-n_i)!} [1]^{N_i-n_i} \} (\frac{1}{\mu})^{N_i-n_i}$$

$$= (\frac{1}{\mu})^{N_i-n_i} , \quad i \neq m+1$$

and

$$g_{m+1}(y_{m+1}) = (n_1+n_2+\ldots+n_m)! \{ \prod_{r=1}^{m} (\frac{1}{n_i!}) p^{n_i} \} (\frac{1}{\alpha})^{n_1+n_2+\ldots+n_m}$$

$$= (\frac{p}{\alpha})^{n_{m+1}} \times \frac{(n_1+n_2+\ldots+n_m)!}{n_1!\, n_2! \ldots n_m!} .$$

P(s) becomes

$$P(s) = Cx(\frac{1}{\mu})^{(N_1+N_2+\ldots+N_m)-(n_1+n_2+\ldots+n_m)} \times (\frac{p}{\alpha})^{n_{m+1}} \times \frac{(n_1+n_2+\ldots+n_m)!}{n_1!n_2! \ldots n_m!}$$

$$= Cx(\frac{1}{\mu})^{N} \times (\frac{\mu p}{\alpha})^{n_1+n_2+\ldots+n_m} \times \frac{(n_1+n_2+\ldots+n_m)!}{n_1!n_2! \ldots n_m!}$$

Since the equilibrium state probabilities must sum to one, the normalization constant C can be computed as

$$C^{-1} = \sum_{n_1=0}^{N_1} \sum_{n_2=0}^{N_2} \cdots \sum_{n_m=0}^{N_m} \left(\frac{1}{\mu}\right)^N \left(\frac{\mu p}{\alpha}\right)^{n_1+n_2+\ldots+n_m} \times \frac{(n_1+n_2+\ldots+n_m)!}{n_1!\, n_2! \ldots n_m!} .$$

After computing the equilibrium state probabilities by the above method, TT and TP can be found using the approach outlined for Model 3.

4.2.6 Multiple CPU's with local memory - Model 5.

An analytical solution for this model (Figure 4-2(e)) can be derived using the approach of Gordon and Newell [GORD 67]. They showed that the state equilibrium probabilities for a particular class of models can be expressed as

$$P(s = \langle n_1, n_2, \ldots, n_m \rangle) = \frac{1}{G(N,m+1)} \prod_{j=1}^{m+1} (x_j)^{n_j} , \qquad (4\text{-}9)$$

where n_j is the number of jobs in queue j, G(N,m+1) is a normalizing constant, and the x_j's are the solutions to the set of linear equations

$$\mu_j x_j = \sum_{k=1}^{m+1} \mu_k x_k p_{kj} \quad , \quad j = 1,2,\ldots m+1 . \qquad (4\text{-}10)$$

Equation (4-9) is a specail case of equation (4-8) with only one class of jobs. p_{kj} is the probability that a job leaving processor k will join queue j. x_j can be interpreted as the relative probability that a job will join the processor queue j. Since the x's may be normalized [BUZE 73] and processors 1 through m are identical, we may set $x_1 = x_2 = \ldots = x_m = 1$ and $\mu_1 = \mu_2 = \ldots = \mu_m = \mu$. With

$$p_{kj} = \begin{cases} p & , \quad k = 1,2,\ldots m \text{ and } j = m+1 \\ \frac{1-p}{m} & , \quad k = j = 1,2,\ldots,m \\ \frac{1}{m} & , \quad k = m+1 \text{ and } j = 1,2,\ldots,m \\ 0 & , \quad \text{elsewhere} \end{cases}$$

equation (4-10) becomes

$$\mu = \frac{1}{m}\left[m\mu(1-p) + \alpha x_{m+1}\right] \quad , \quad j = 1,2,\ldots,m$$

$$\alpha x_{m+1} = m\mu p \quad , \quad j = m+1$$

Solving the above equations, we obtain $x_{m+1} = \frac{m\mu p}{\alpha}$.

Let U_{m+1} and U_j be the utilization factors of service center 2 and processor j in service center 1 respectively. Then

$$U_{m+1} = 1 - P(s|n_{m+1} = 0)$$

$$= 1 - \sum_{\substack{\text{all } s \text{ and} \\ n_{m+1} = 0}} \frac{1}{G(N,m+1)} (x_{m+1})^{n_{m+1}}$$

$$= 1 - \binom{N+m-1}{m-1} \frac{1}{G(N,m+1)} .$$

Since under equilibrium the job departure rate at a service center must be equal to the arrival rate, we have the following relation at service center 2:

$$\alpha U_{m+1} = m\mu p U_j \quad , \quad j \neq m+1 .$$

Consequently,

$$U_j = \frac{\alpha}{m\mu p} U_{m+1}$$

$$= \frac{\alpha}{m\mu p}\left[1 - \binom{N+m-1}{m-1} \frac{1}{G(N,m+1)}\right] \qquad (4\text{-}11)$$

The remaining work for this model is to compute the normalizing constant G(N,m+1). It was shown in [BUZE 73] that G(N,m+1) can be evaluated recursively as

$$G(N,m+1) = \sum_{n_{m+1}=0} \prod_{i=1}^{m+1} (x_i)^{n_i} + \sum_{n_{m+1}>0} \prod_{i=1}^{m+1} (x_i)^{n_i}$$

$$= G(N,m) + x_{m+1}\, G(N-1,m+1) \qquad (4\text{-}12)$$

For Model 5 it turns out that G(N,m+1) can be expressed in a simpler form. Using the property of conservation of probability, it follows from (4-9) that

$$G(N,m+1) = \sum_{\text{all } s} \prod_{i=1}^{m+1} (x_i)^{n_i}$$

$$= \sum_{\substack{\text{all } s \text{ such that} \\ n_1+n_2+\ldots+n_{m+1}=N}} x_1^{n_1} x_2^{n_2} \ldots x_{m+1}^{n_{m+1}} .$$

Since $x_1 = x_2 = \ldots = x_m = 1$, this can be expanded as

$$G(N,m+1) = \sum_{n_1+n_2+\ldots+n_m=N} x_{m+1}^0 + \sum_{n_1+n_2+\ldots+n_m=N-1} x_{m+1}^1 + \ldots$$

$$+ \sum_{n_1+n_2+\ldots+n_m=0} x_{m+1}^N$$

$$= \binom{N+m-1}{m-1} + \binom{N+m-2}{m-1} x_{m+1} + \ldots + \binom{m}{m-1} x_{m+1}^{N-1} +$$

$$+ \binom{m-1}{m-1} x_{m+1}^N , \qquad (4\text{-}14)$$

where $x_{m+1} = \frac{m\mu p}{\alpha}$.

The turnaround time TT and throughput rate TP can be derived in the same way as in Model 3 with U_j defined in (4-11).

4.2.7 Comparison of performance.

We have computed and plotted the turnaround time (TT) and throughput (TP) versus N (the degree of multiprogramming) for each of the 16 combinations of the following parameter values: $\mu = 1\ \text{sec}^{-1}$, $\alpha/\mu = 1,2,4,8$, $p = 0.5$, $m = 2,4,8,16$. Figures 4-7 to 4-10 show the comparisons of the five models when $\mu = 1\ \text{sec}^{-1}$, $\alpha/\mu = 4$, $p = 0.5$, and $m = 2,4,8,16$ respectively. The curves are marked by symbols O, Δ, +, $\diamond$ and x to represent Models 1, 2,3,4, and 5. If we define TT_k and TP_k as the turnaround time and throughput rate of Model k with parameter values α,μ,p,m, and N, then for each choice of parameter values

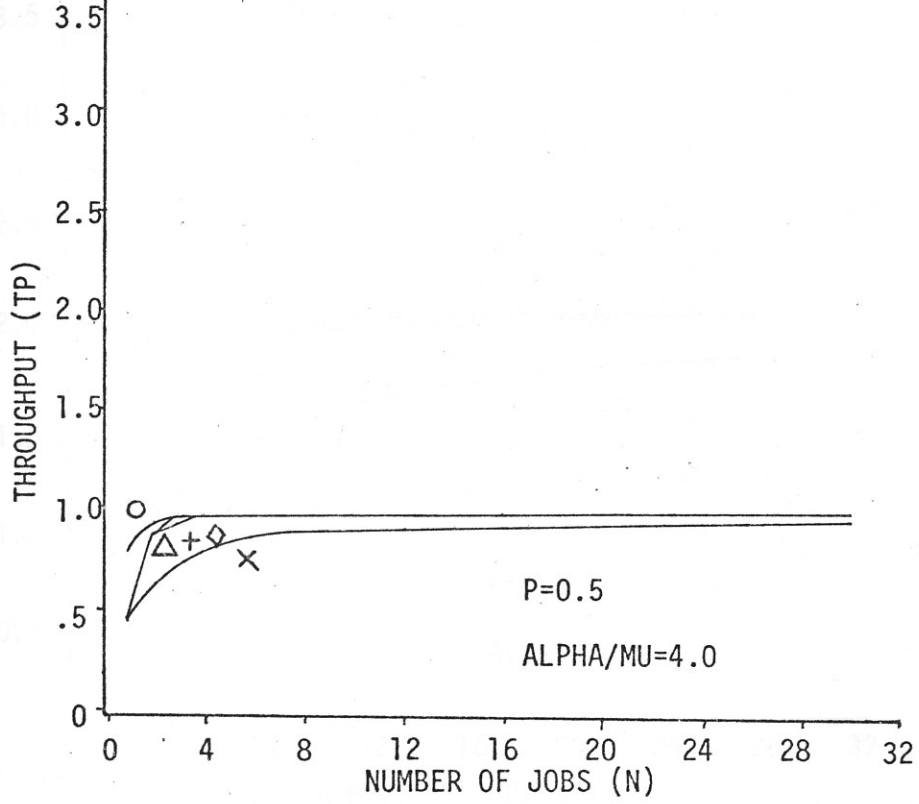
$$TT_1 \leq TT_2 \leq TT_3 \leq TT_4 \leq TT_5$$

and

$$TP_1 \geq TP_2 \geq TP_3 \geq TP_4 \geq TP_5 .$$

This linear ordering can be supported intuitively by comparing the processing rate of service center 1 under various customer distributions for each of the models. When there are a total of $c \geq 1$ customers in service center 1, Model 1 has a processing rate of $m\mu$, while model 2 has an effective rate of only $\min[c\mu, m\mu]$. This rate is further reduced in the case of Model 3, since one or more of the m processors may be idle while a customer is waiting without service at one of the other busy processors. This condition is more probable for Model 4 since its load is not dynamically balanced as in Model 3. Model 5 gives the poorest performance because no attempt is made to balance the load.

Figure 4-7 to 4-10 also show that the performance of model 3 closely approximates Model 2 and Model 4, with Model 2 as its upper bound and Model 4 as



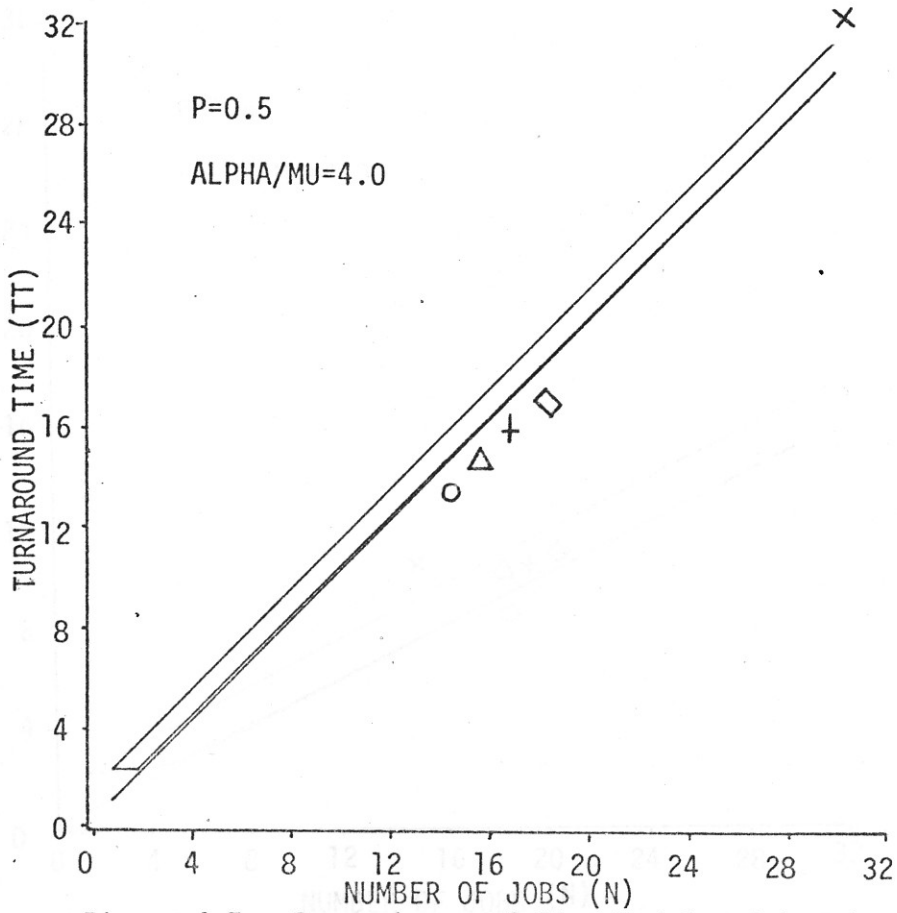
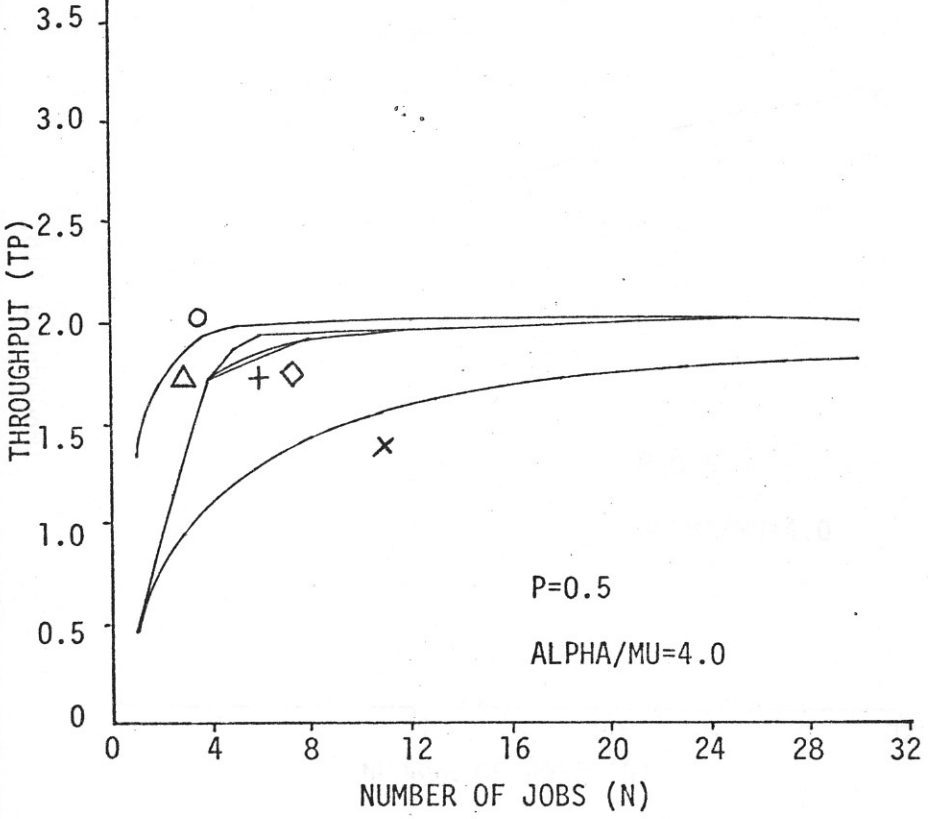


Figure 4-7: Comparisons of Five Models with m=2.



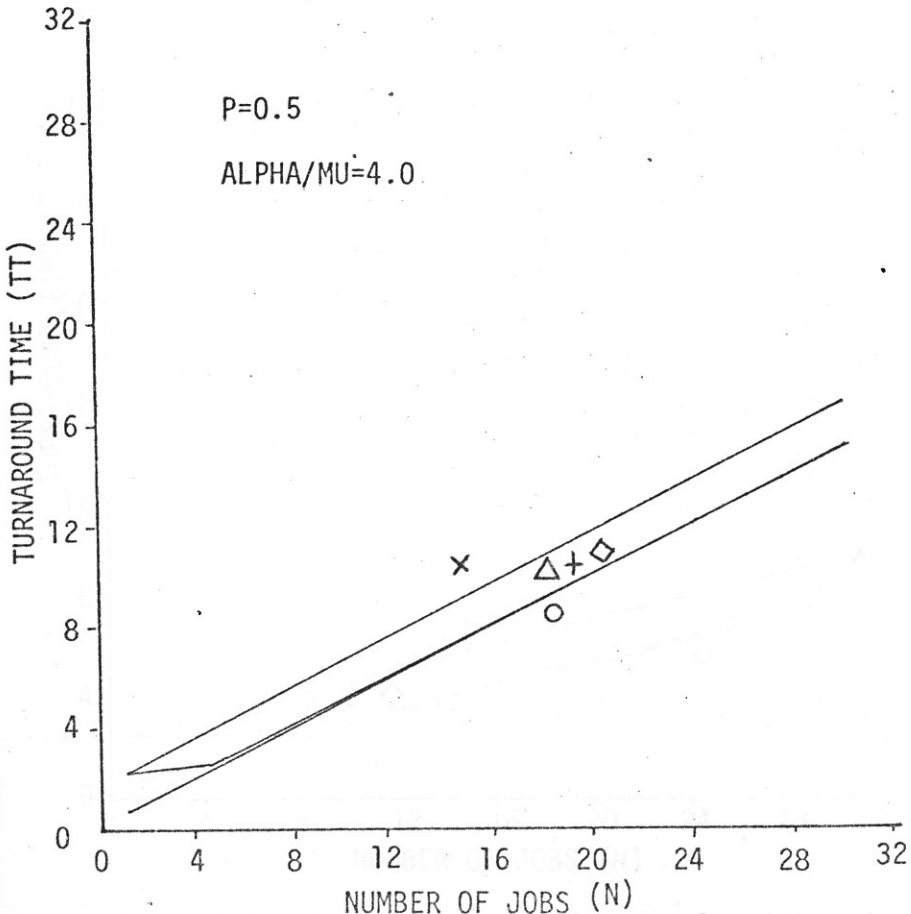
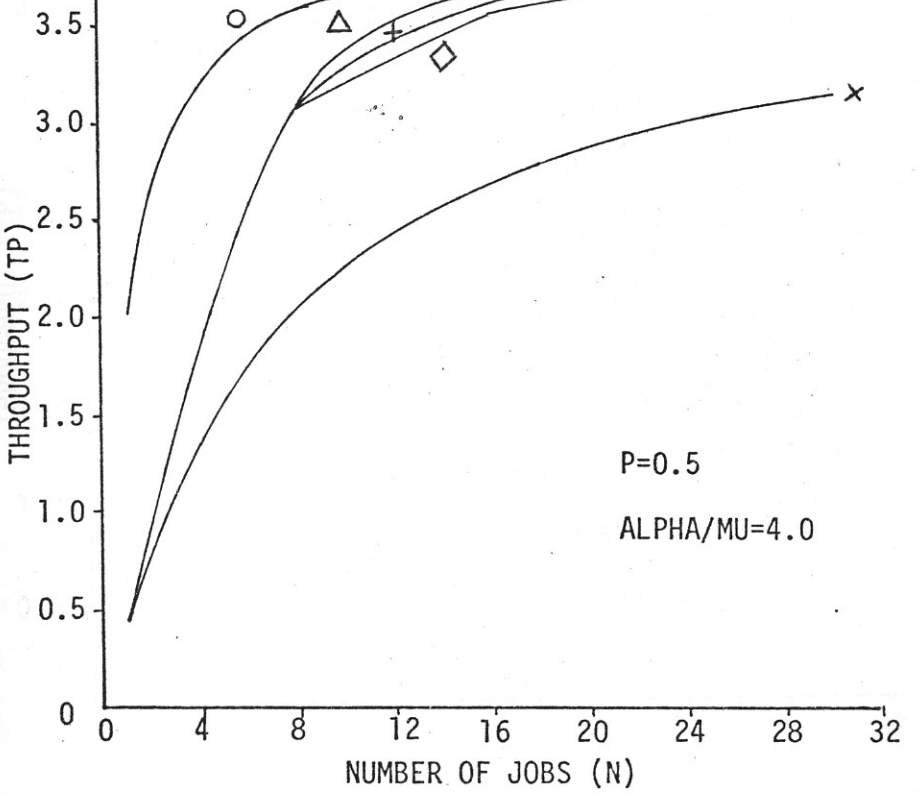


Figure 4-8: Comparisons of Five Models with m=4.



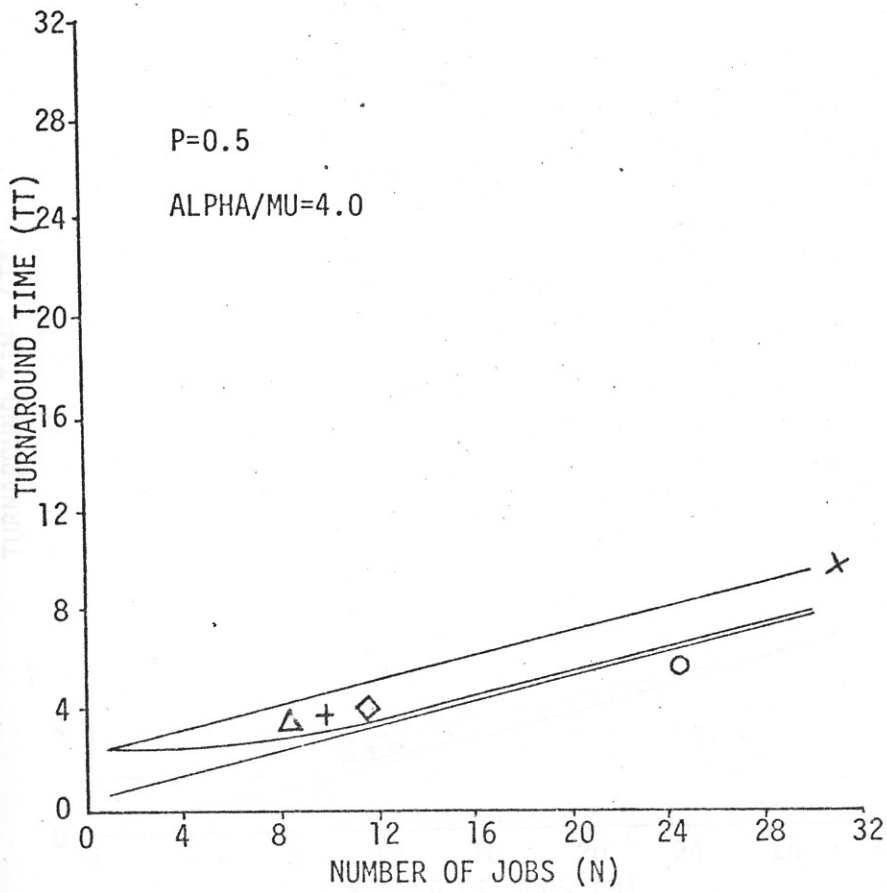
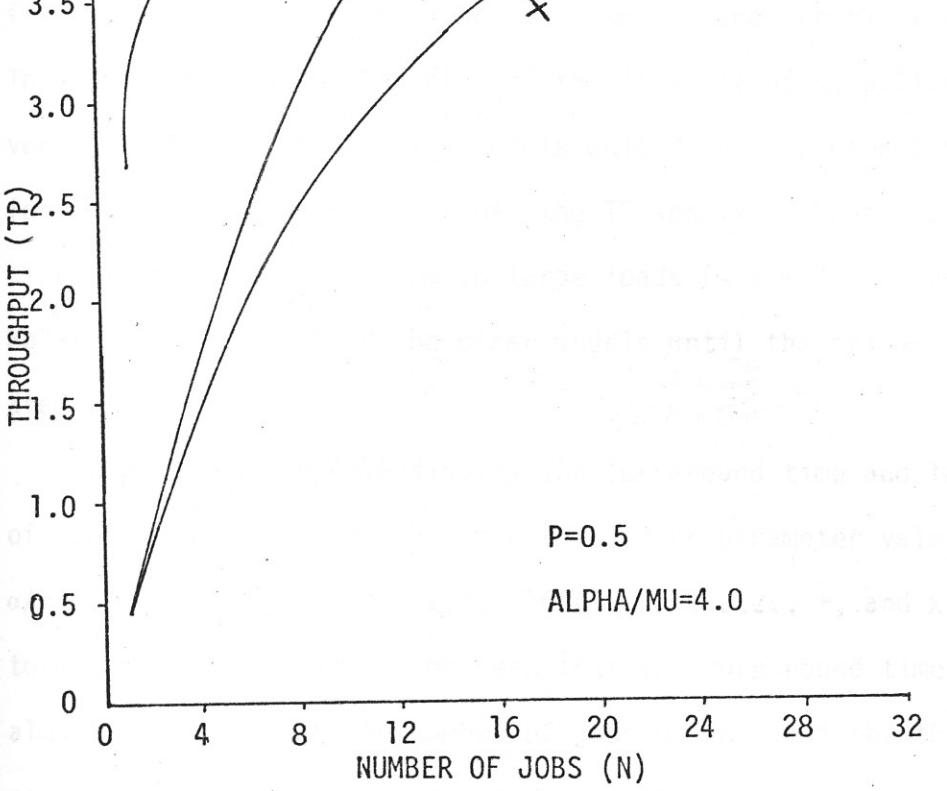


Figure 4-9: Comparisons of Five Models with m=8.



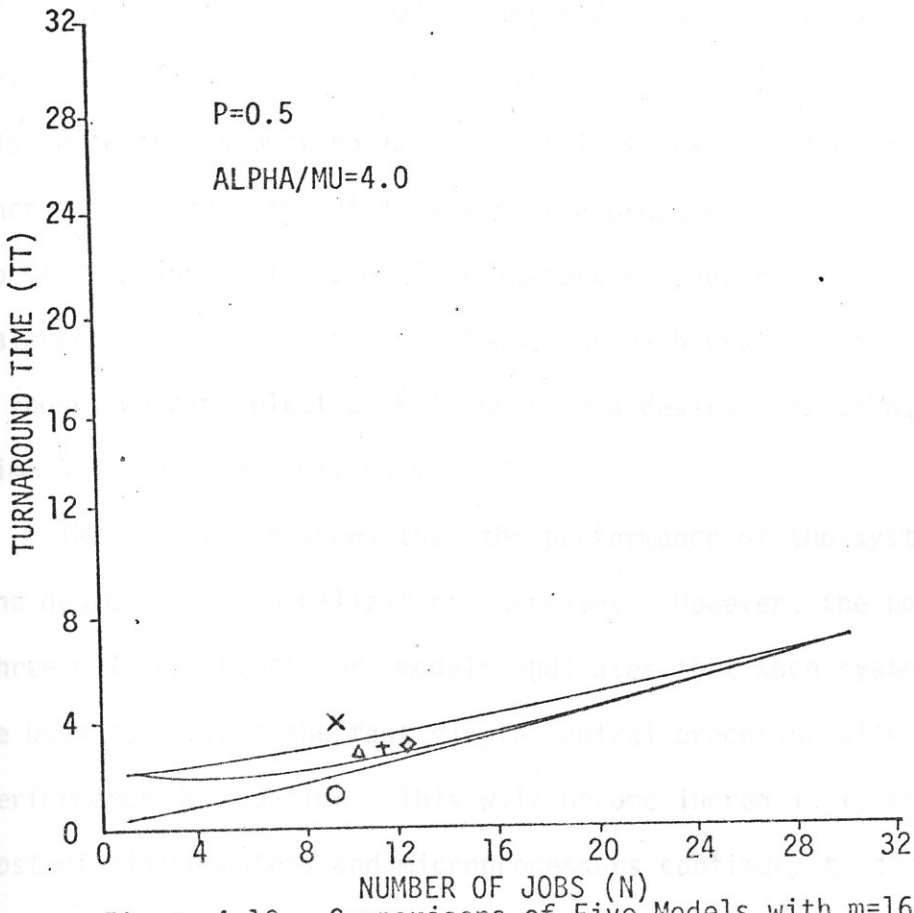
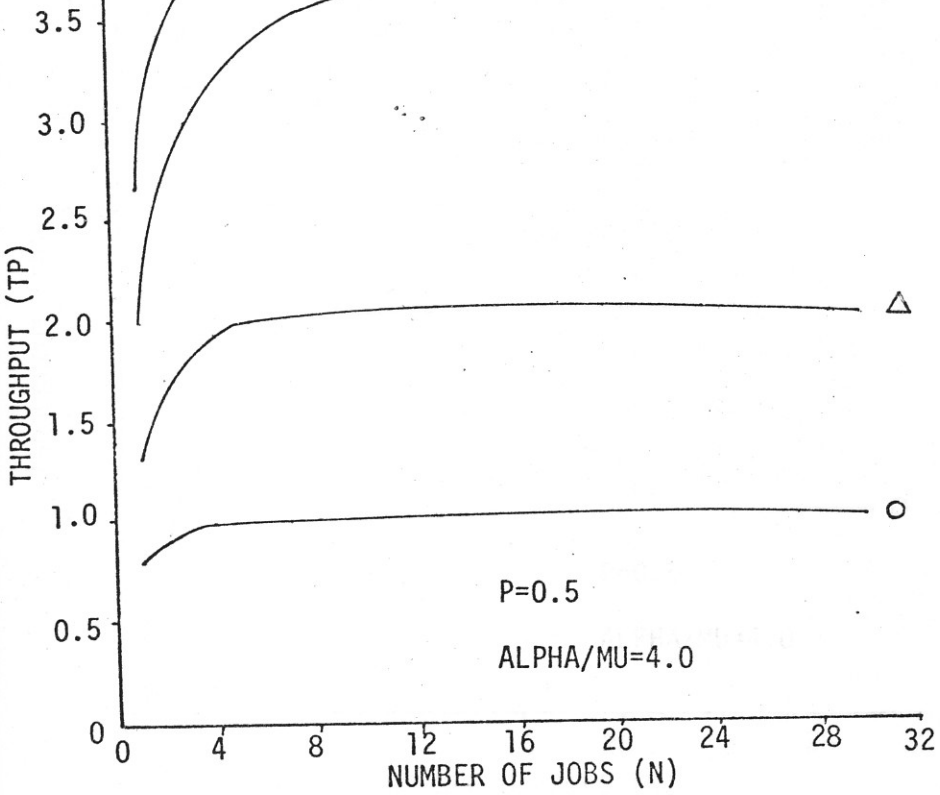


Figure 4-10: Comparisons of Five Models with m=16.

its lower bound. In fact, the three models are identical when $N \leq m$. This demonstrates the benefits of the load balancing policy of Model 3 versus Model 5. Of the five models only Model 3 does not have a closed form solution. In general, both the TT and TP of Model 2, 3, and 4 are close to Model 1 for medium to large loads ($N \geq m$). The performance of Model 5 lags far behind the other models until the system becomes saturated for large N.

Figure 4-11 to 4-15 display the turnaround time and throughput rate of Models 1, 2, 3, 4 and 5 respectively for parameter values $\mu = 1\ sec^{-1}$, $\alpha/\mu = 4$, $p = 0.5$, and $m = 2,4,8,16$ (Symbols $0, \Delta, +,$ and x correspond to $m = 2,4,8,16$). It can be seen that the turnaround time increases almost linearly with the number of jobs N, and that the throughput of each system is ultimately limited by either the maximum processing rate of service center 1 or service center 2. The structure of the five models reveals that the maximum throughput rate is $\min[(1-p)m\mu,(1-p)\alpha/p]$. We also note that almost no improvement in turnaround time can be gained by increasing m once the utilization of a processor in service center 1 equals to utilization of the processor in service center 2. This condition of balance exists when $\alpha = m\mu p$. Thus, for each system model with α, μ, and p known, we can select an N to achieve a desired throughput and turnaround time while also minimizing m.

The comparison shows that the performance of the system decreases as the degree of decentralization increases. However, the performance of the three multiple processor models indicates that such systems may sometimes be used to replace the fast single central processor without significant performance degradation. This will become increasingly attractive as the cost of minicomputers and microprocessors continues to decrease.



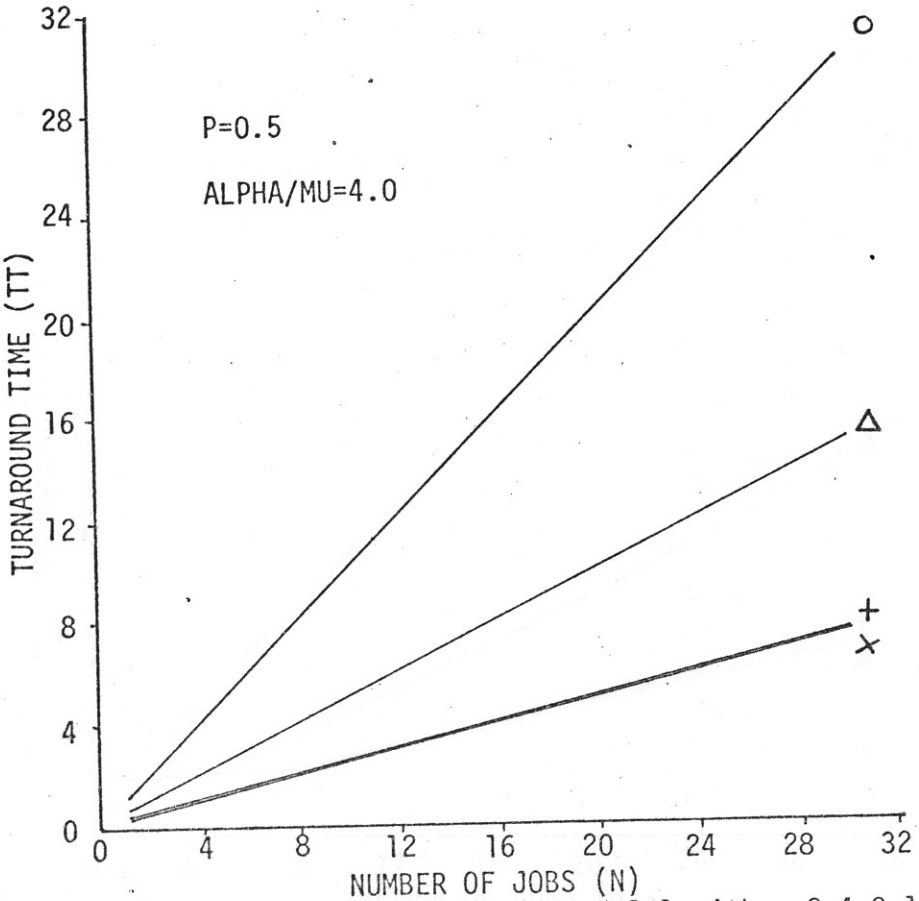
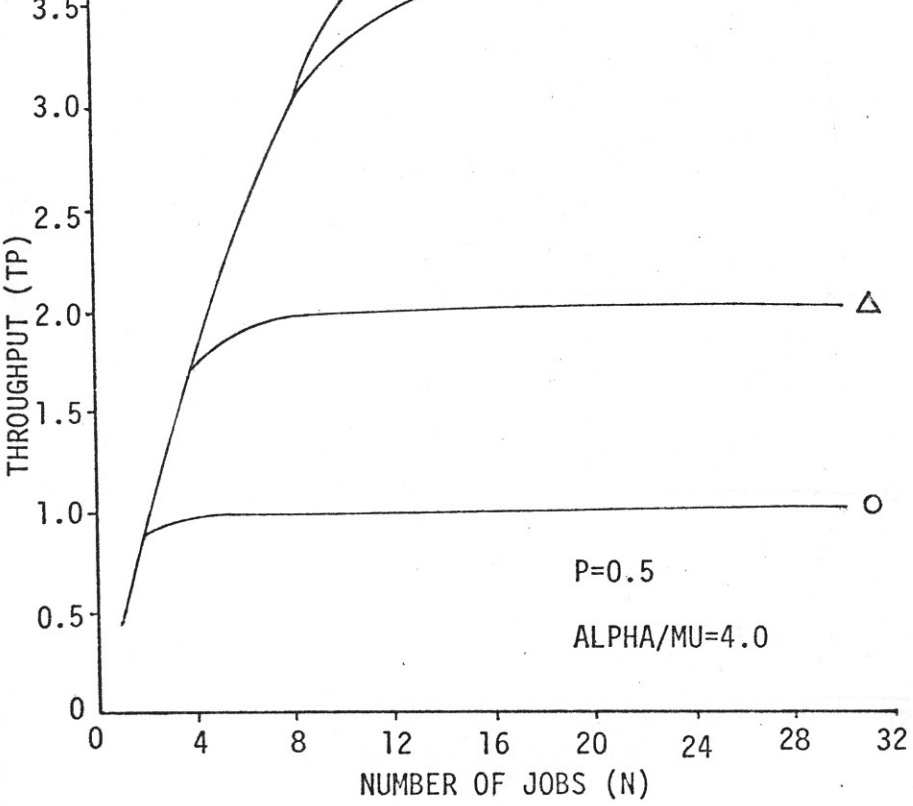


Figure 4-11: Performance of Model 1 with m=2,4,8,16.



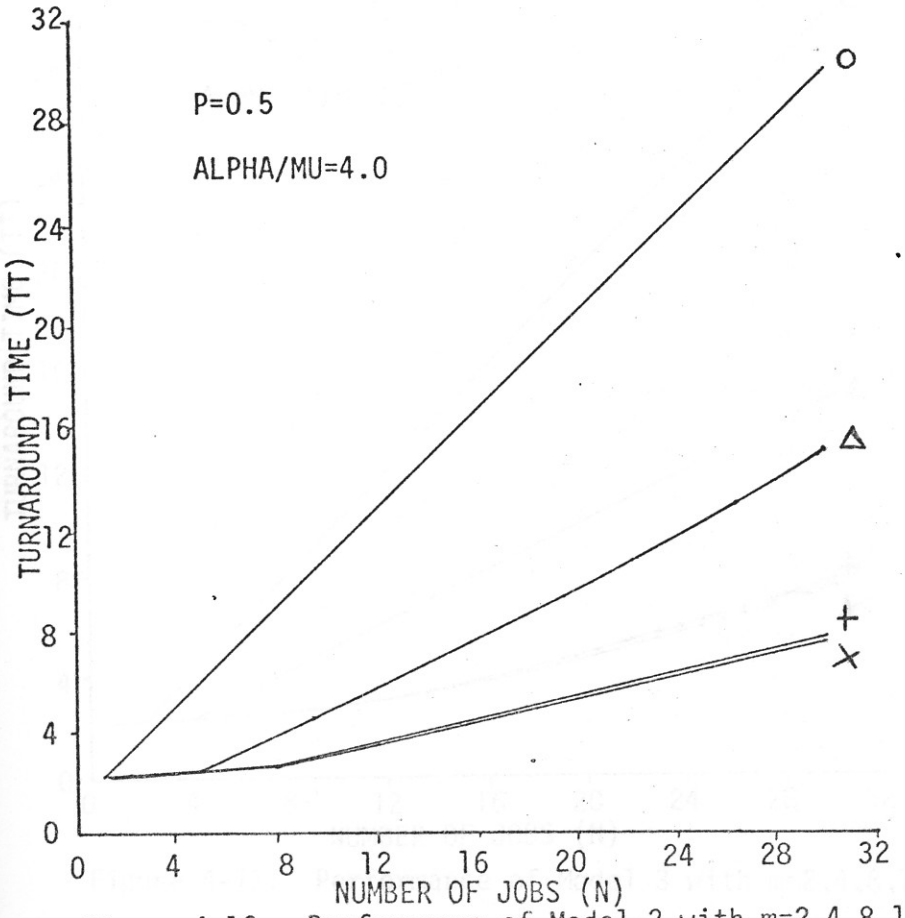
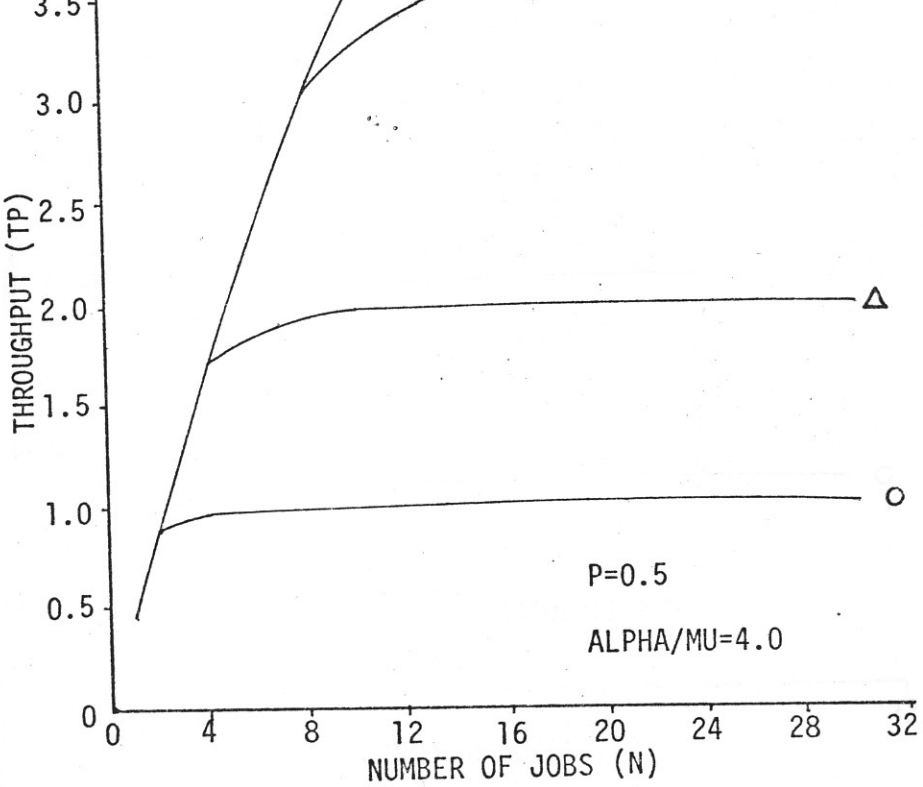


Figure 4-12: Performance of Model 2 with m=2,4,8,16.



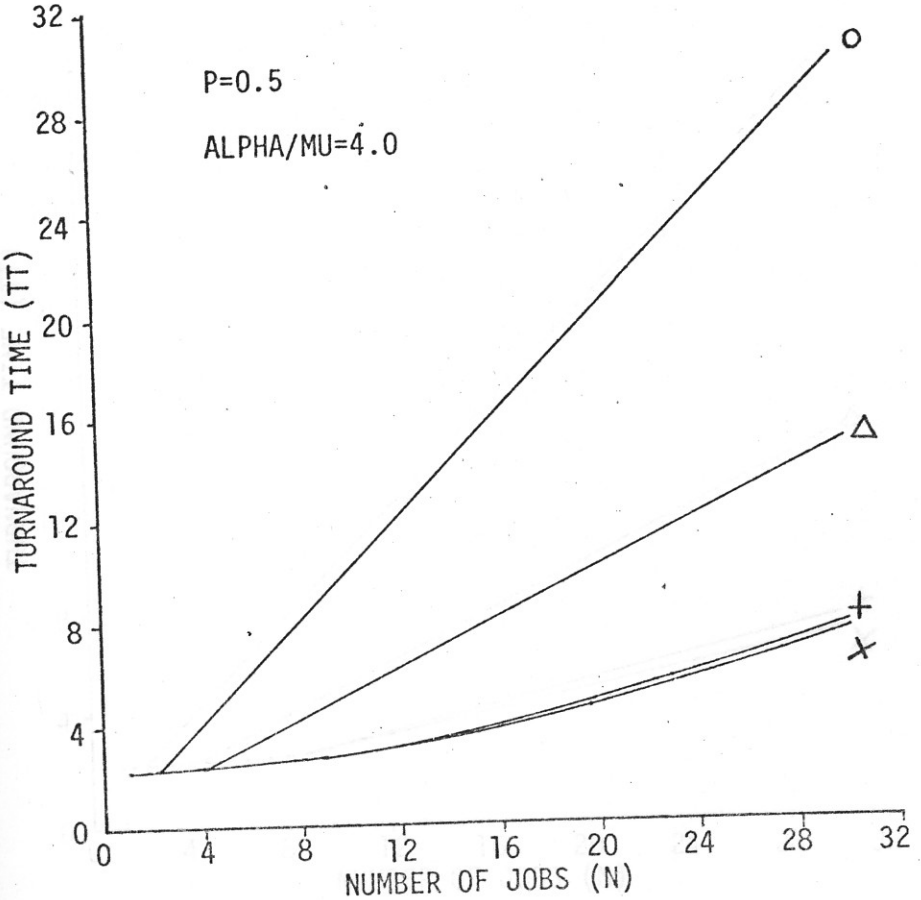
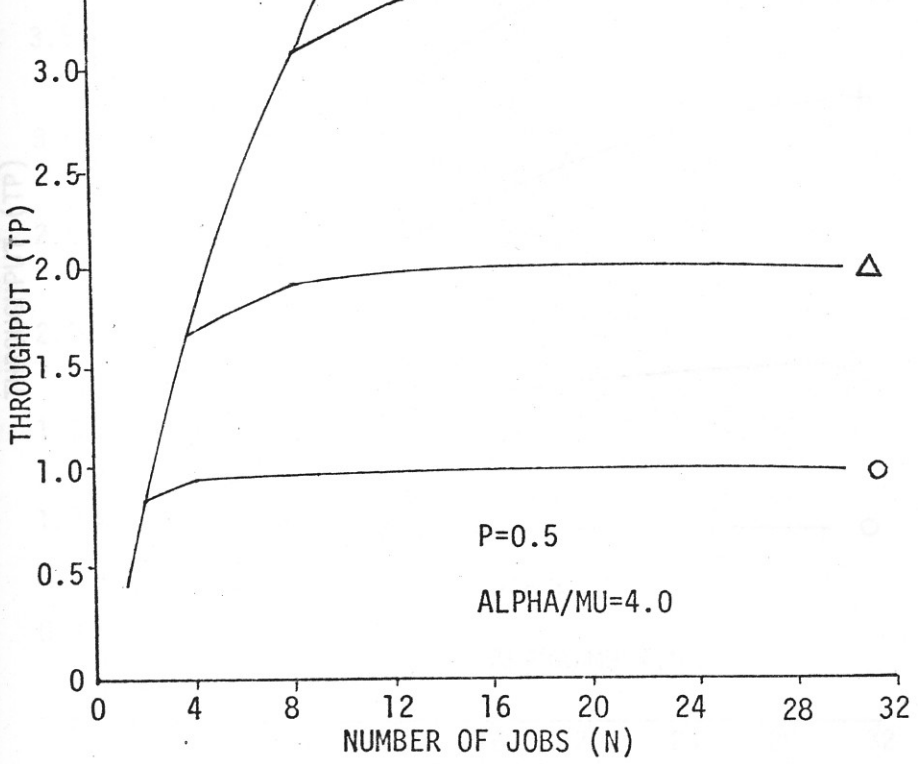


Figure 4-13: Performance of Model 3 with m=2,4,8,16.



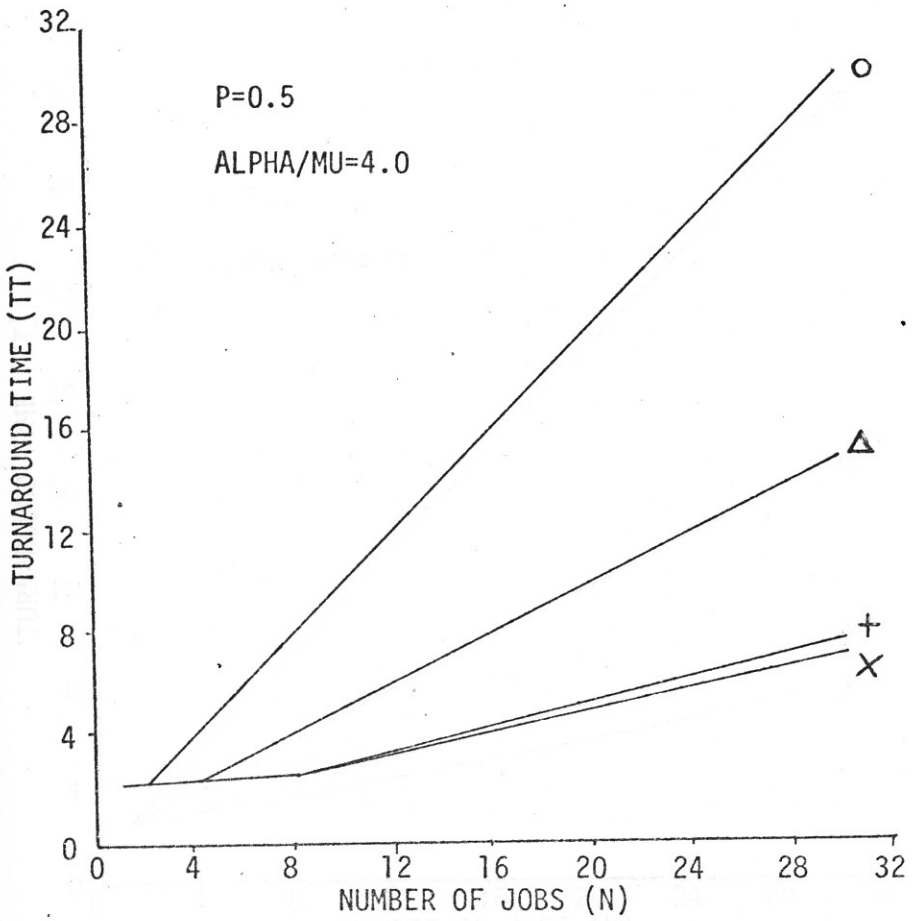
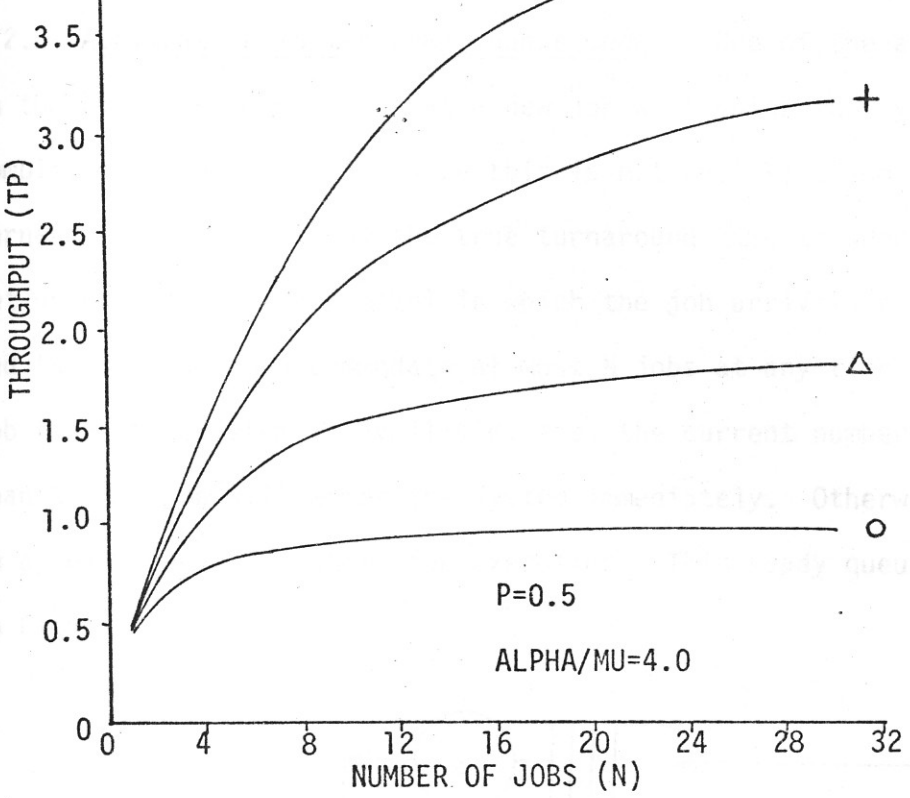


Figure 4-14: Performance of Model 4 with m=2,4,8,16.



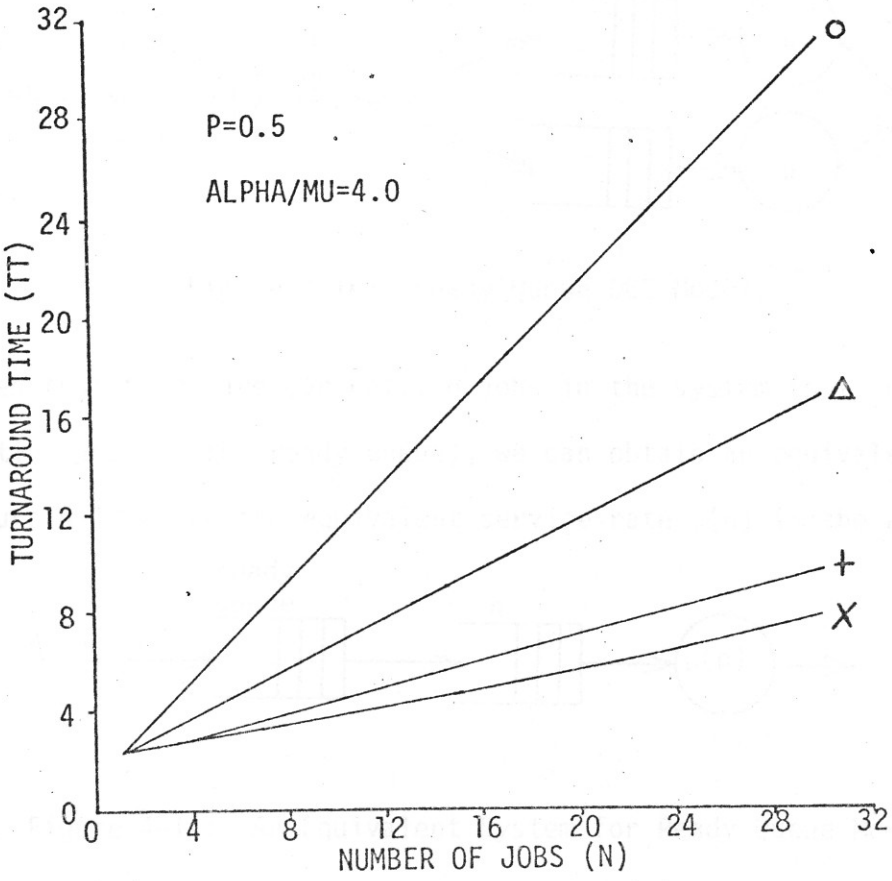


Figure 4-15: Performance of Model 5 with m=2,4,8,16.

4.2.8 Application to the ready queue model. One of the assumptions made in the previous models is that a new job will enter the system upon the completion of a job. Of course this is not realistic and the computed turnaround time TT is not the true turnaround time as seen by the user. Let us consider an open model in which the job arrival is a Poisson process and the system can accommodate at most N jobs at any time. If an arriving job finds the system is available, i.e. the current number of jobs is less than N, the job will enter the system immediately. Otherwise, it is put in a ready queue and waits for execution. This ready queue model is shown in Figure 4-16.

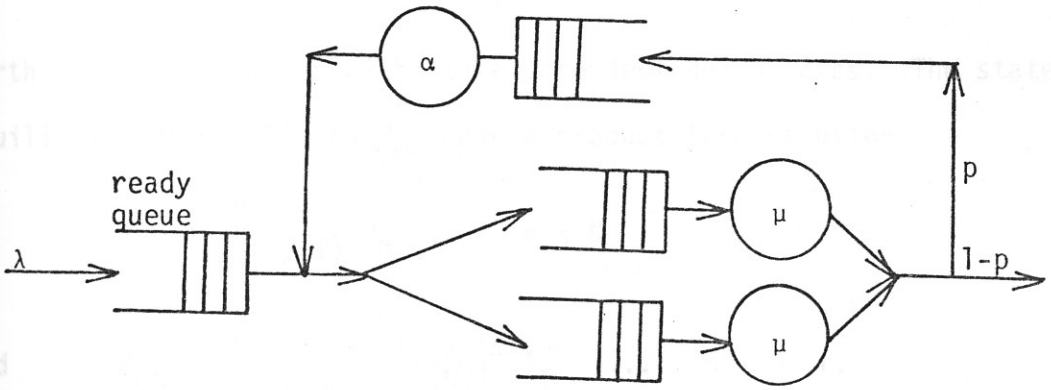


Figure 4-16: Ready Queue DCS Model.

Given that there are currently n jobs in the system (not including the waiting jobs in the ready queue), we can obtain an equivalent system as Figure 4-17 where the equivalent service rate μ(n) is the average through-

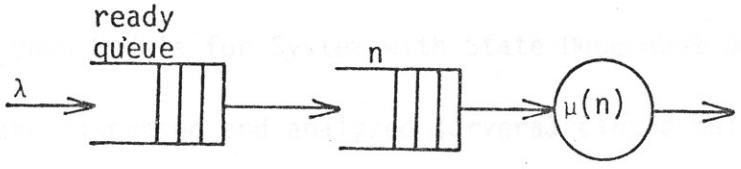


Figure 4-17: An Equivalent System for Ready Queue Model.

put calculated for the closed models with degree of multiprogramming DMP = n discussed in the previous sections. The state of the system can then be defined as a tuple (m,n) where m and n are the number of jobs in the ready queue and system queue respectively. The state transition diagram can be obtained easily as Figure 4-18. This state diagram is a simple

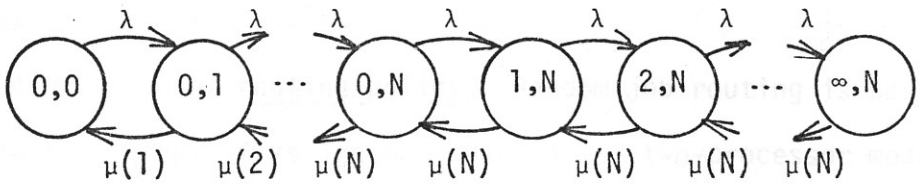


Figure 4-18: State Transition Diagram for Ready Queue Model.

birth and death process with state dependent death rates. The state equilibrium probabilities P_{mn} have a product from solution.

$$P_{On} = \prod_{i=1}^{n} \frac{\lambda}{\mu(i)} P_O \quad \ldots \quad n \leq N$$

and $$P_{mN} = \prod_{i=1}^{N} \frac{\lambda}{\mu(i)} \cdot \left[\frac{\lambda}{\mu(N)}\right]^m P_O \quad \ldots\ldots \quad m = 0, 1, \ldots \infty$$

where $$P_O = \left[\sum_{\text{all states}} P_{mn}\right]^{-1}.$$

The true turnaround time as seen by the user can be computed from the steady state probabilities directly.

4.3 Open Models for System with State Dependent Job Routing Policies

We have discussed and analyzed serveral closed multiple processor DCS models. We have shown that the job scheduling strategy in a DCS system can

Each circle in the diagram represents an exponential stage which generates an exponential inter-arrival time. The arrival process will have an inter-arrival time of $\frac{i}{2\lambda}$ with probability $(1-p_1)^{i-1}p_1$. To obtain the inter-arrival time distribution we can first find the s-transform of the staging diagram and then determine the inverse transform. Starting with the s-transform of an exponential stage, $\frac{2\lambda}{s+2\lambda}$, we can compute the overall s-transform A(s). That is,

$$A(s) = \sum_{i=1}^{\infty} (1-p_1)^{i-1} p_1 \left(\frac{2\lambda}{s+2\lambda}\right)^i$$

$$= \frac{2p_1\lambda}{s+2\lambda}\left[1 + (1-p_1)\frac{2\lambda}{s+2\lambda} + (1-p_1)^2\left(\frac{2\lambda}{s+2\lambda}\right)^2 + \ldots\right]$$

$$= \frac{2p_1\lambda}{s+2p_1\lambda} .$$

The inverse transform of A(s) is an exponential distribution with mean $2p_1\lambda$. Thus, we have proved that processor 1 has an independent Poisson arrival process with mean $2p_1\lambda$. Similarily, the arrival process for processor 2 is also Poisson with mean $2p_2\lambda$. The average turnaround time TT for the composite system is

$$TT = p_1 \times \text{average turnaround time in processor 1}$$
$$+ p_2 \times \text{average turnaround time in processor 2}$$
$$= \frac{p_1}{\mu-2p_1\lambda} + \frac{p_2}{\mu-2p_2\lambda} .$$

It is obvious that average turnaround time is minimized when $p_1 = p_2$, i.e. $p_1 = p_2 = \frac{1}{2}$.

4.3.2 Alternating job routing policy. We can do better in balancing the job load for the two-processor system by sending all even numbered jobs to processor 1 and all odd numbered jobs to processor 2. This is called the alternating strategy. The idea is similar to the round robin concept employed in many systems. For our particular model, the new inter-arrival time becomes the sum of two exponential inter-arrival times. This changes the exponential arrival distribution to an E_2(two stage Erlang) arrival distribution with mean λ for each processor. The system again is reduced to two independent $E_2/M/1$ systems. Standard solutions exist for $E_2/M/1$ [KLEI75]. The average turnaround time TT is given by

$$TT = \frac{\sigma}{\mu(1-\sigma)} + \frac{1}{\mu} = \frac{1}{\mu(1-\sigma)} ,$$

where σ is the unique root of $\sigma = A(s)\Big|_{s=\mu-\mu\sigma}$ in the range of $0 < \sigma < 1$ and $A(s) = (\frac{2\lambda}{s+2\lambda})^2$ is the s-transform of E_2 distribution. σ can be directly computed from the equation

$$\sigma = \frac{1}{2} + 2\frac{\lambda}{\mu} - \frac{1}{2}\sqrt{1 + 8\frac{\lambda}{\mu}}$$

In the case of general m-processor systems, the effect of the alternating policy is simply m $E_m/M/1$ systems. This leads to an interesting limiting case as m approaches infinity. Recall that for m processors the s-transform of the arrival process for each processor is $A(s) = (\frac{m\lambda}{s+m\lambda})^m$. As m approaches ∞ we get

$$\lim_{m\to\infty} A(s) = \lim_{m\to\infty} (\frac{m\lambda}{s+m\lambda})^m$$

$$= \frac{1}{\lim_{m\to\infty} (\frac{s+m\lambda}{m\lambda})^m}$$

$$= \frac{1}{\lim_{m\to\infty} (1 + \frac{s}{m\lambda})^m}$$

$$= \frac{1}{\lim_{m\to\infty} (1 + \frac{1}{\frac{m\lambda}{s}})^{\frac{m\lambda}{s} \cdot \frac{s}{\lambda}}}$$

$$= \frac{1}{e^{\frac{s}{\lambda}}}$$

$$= e^{-\frac{s}{\lambda}} .$$

The transform of $e^{-s/\lambda}$ is the impulse function $U_0(t - \frac{1}{\lambda})$. This indicates that we have a deterministic interarrival time of $\frac{1}{\lambda}$. Therefore $E_m/M/1$ becomes D/M/1 when $m \to \infty$. The solution for D/M/1 can be obtained by finding σ where $\sigma = e^{-(\mu-\mu\sigma)/\lambda}$.

4.3.3 Join-the-shorter-queue policy. We have discussed the random and the alternating job routing policies. A job dispatcher using the random strategy requires no knowledge of the system, but in order to use the alternating strategy he needs to remember which processor received the previous job. We now introduce a more sophisticated job scheduling scheme, join-the-shorter-queue policy. The job dispatching strategy depends on the current state of the system.

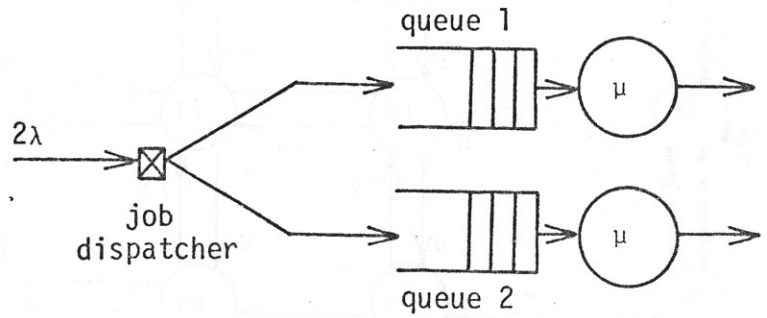


Figure 4-21: Join-the-Shorter-Queue Job Routing.

shorter queue. If both queues have an equal number of jobs (including empty), the arriving job is dispatched randomly to one of the two queues with equal probability. The model was also studied by Flatto [FLAT76] who used the generating function approach to solve for the equilibrium state probabilities. The result is a complicated infinite sum for the equilibrium state probabilities. In this section we use the recursive technique [HERZ75] described in Chapter III for the analysis of the model. The system can be described by the two dimensional state-transition-rate diagram in Figure 4-22. Since the system is an open model we have an infinite number of states. However, if the system is ergodic, the equilibrium state probabilities P_{ij}(Pr[i jobs in queue 1 and j jobs in queue 2]) will approach zero for large i and j. It is therefore reasonable to truncate the infinite state diagram by assuming that the maximum queue length is N.

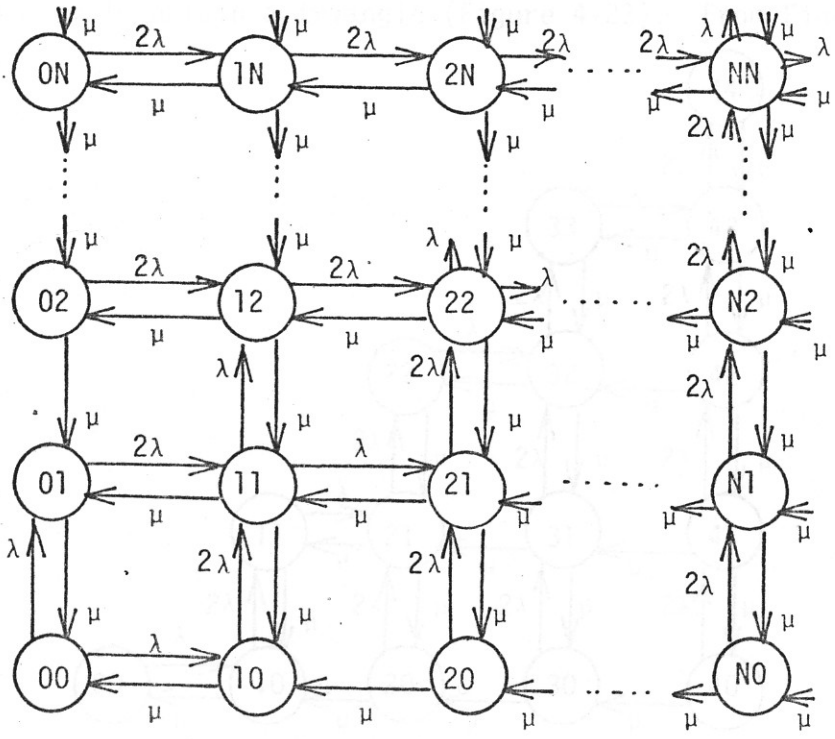


Figure 4-22: State-Transition-Rate Diagram.

This will introduce a negligible error if N is properly chosen. The total number of states is then $(N + 1)^2$. Let $P_{N,j} = X_j$, where the X's are unknown variables for j = 0, 1, ..., N. Using the GBE for $P_{N,j}$ we can express $P_{N-1,j}$ as a function of the X's. In turn all the P_{ij}'s, for i = N-1 down to 0, can be expressed in terms of the X's. This process reduces the unknowns from $(N + 1)^2$ to N + 1. Finally, we can use the GBE for P_{0j}, j = 0, 1, ..., N, to obtain a set of N independent linear homogeneous equations in the X's. Together with the conservation equation $\sum_{i=0}^{N} \sum_{j=0}^{N} P_{ij} = 1$, the set of N + 1 simultaneous equations can be solved for the X's.

Additional properties exhibited by the model enable us to further simplify the method. First, we recognize that the state diagrams (Figure 4-22) are symmetric about the diagonal elements (i,i) and therefore can be folded over to obtain a triangle (Figure 4-23). From Figure 4-23 we

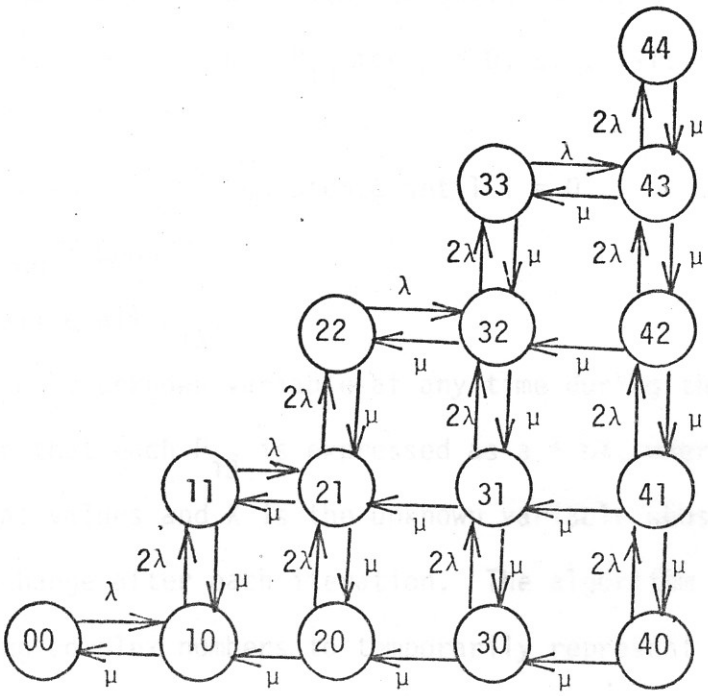


Figure 4-23: A Folded State Diagram for N = 4.

now notice that if state probabilities $P_{N,N}$ and $P_{N-1,N-1}$ were known, then $P_{N,j}$, for j = N-1 down to 0, could be calculated recursively by using the GBE for state (N,j+1). This property allows us to efficiently calculate all the state equilibrium probabilities by proceeding recursively from top to bottom and right to left in the state diagram. We may initially set $P_{N,N}$ to an arbitrary constant, since the P_{ij}'s will be computed iteratively and normalized at the end of the process. $P_{N-1,N-1}$ is initially unknown and represented by the variable X, but can be found by solving the GBE for state (N,0). The general method is described by the following algorithm and illustrated in APPENDIX A for the sample system of Figure 4-23.

1. Set i = N, CONST = 1.
2. Set $P_{i,i}$ = CONST, $P_{i-1,i-1}$ = X (an unknown variable).
3. For j = i down to 1, use GBE for state (i,j) to determine an expression for $p_{i,j-1}$ as a function of X.
4. Use the GBE for state (i,0) to solve for X.
5. With X known, compute P_{ij} for j = 0, ..., i-1.
6. Set i = i-1, CONST = X.
7. Repeat Step 2 through Step 6 until i = 0.
8. Set P_{00} = CONST.
9. Normalize all P_{ij}.

X is the only unknown variable at any time during the iterations. It can be seen that each P_{ij} is expressed as a + bX, where a and b are known numerical values and X is the unknown variable whose value is subjected to change after each iteration. The algorithm can be implemented by using complex numbers to temporarily represent the P_{ij}, with

real part and imaginary part of each complex number denoting a and b respectively. Since the P_{ij} are relative to one another, they must be normalized at the end to satisfy the conservation relation

$$\sum_{i=0}^{N} \sum_{j=0}^{N} P_{ij} = 1.$$

The algorithm applies similarly to closed queueing models with the join-the-shorter-queue load balancing policy. Figure 4-24 shows a closed

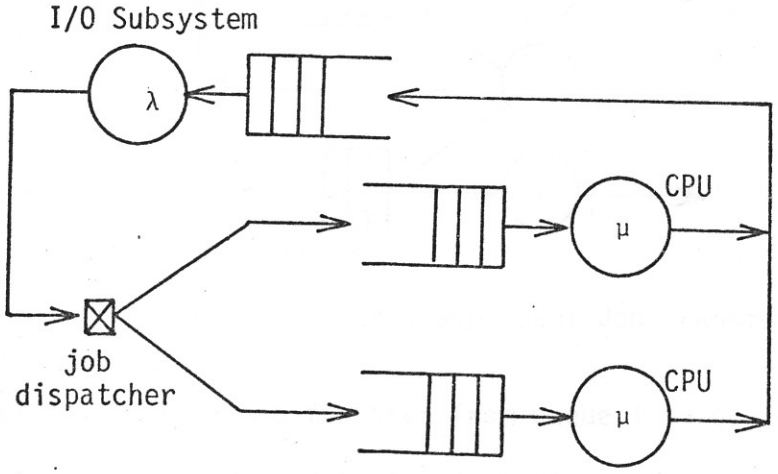


Figure 4-24: Closed Queueing Model with Join-the-Shorter-Queue Load Balancing Policy.

queueing model which can be interpreted as representing a distributed multiprogramming system with fixed degree of multiprogramming 2N, or a demand paging system in which the I/O subsystem represents the activities of the paging system [CHOW76]. The definition of a state and the state-transition-rate diagram are the same as in the open model except that the P_{ij} are by definition identically zero whenever $i > N$ or $j > N$.

4.3.4 Random arrival with channel transfer. Earlier in Chapter III we introduced the idea of balancing the load dynamically through the communication channels. This dynamic load balancing scheme is essential in a distributed system since job entries in a real system are normally initiated from distributed sites. Figure 4-25 illustrates a simple DCS model with state dependent job transfer policy. We assume that the communication

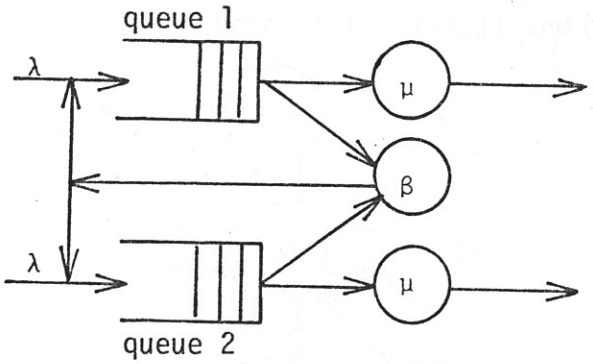


Figure 4-25: State Dependent Job Transfer.

channel initiates a job transfer from queue i to queue j whenever the number of jobs in queue i is two or more greater than the number of jobs in queue j. The channel can only service one job at a time and the job transfer is discontinued if the imbalance condition changes before the channel transfer is completed. Only one bi-directional communication channel is required in the model since the transfer to the shorter queue policy excludes the possibility of two opposite direction transfers simultaneously. The channel can be viewed as an input/output processor. If jobs waiting in queue i are interpreted as being resident in the memory system of central processor i, then the model ignores the effect of memory contention between the input/output and central processors. This is a reasonable approximation for systems with multi-port memory.

The transfer of jobs between queues is sometimes called jockeying.

The special case of instantaneous jockeying ($\beta = \infty$) between homogeneous or heterogeneous queues has been studied by [KOEN66, DISN76] and they have derived closed form solutions for P_{ij}. There is no effective solution method available for our model with $0 < \beta \neq \infty$. The recursive technique as outlined in the last section can not be applied here since the GBE for state (i,j) involves transition inputs not only from its upper and rightward states but also from both state (i-1,j) and (i,j-1) (see Figure 4-26).

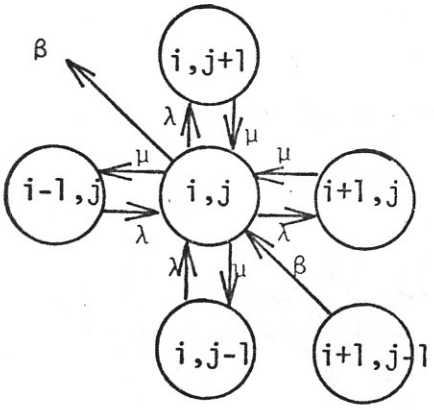


Figure 4-26: Transition of (i,j) to/from Neighboring States.

Neither state (i-1,j) or state (i,j-1) can be expressed only in terms of states (i,j), (i,j+1), (i+1,j), and (i+1,j-1). This makes it impossible to proceed with the iterations from right to left and from top to bottom as described in the recursive method. The analysis of the model is done through simulation. The results and the comparison with other models are shown in the last section of the chapter.

4.3.5 Join-the-shorter-queue policy with channel transfer. One may wish to balance the job load through both the job dispatcher and the communication channels. We now consider the ultimate case in which the join-the-shorter-queue policy is used for both job arrivals and job transfers. The model and the state-transition-rate diagram are shown in Figure 4-27

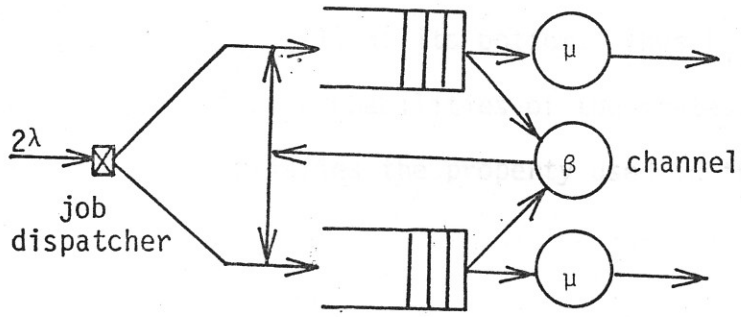


Figure 4-27: Join-the-Shorter-Queue with Channel Transfer.

and Figure 4-28, respectively.

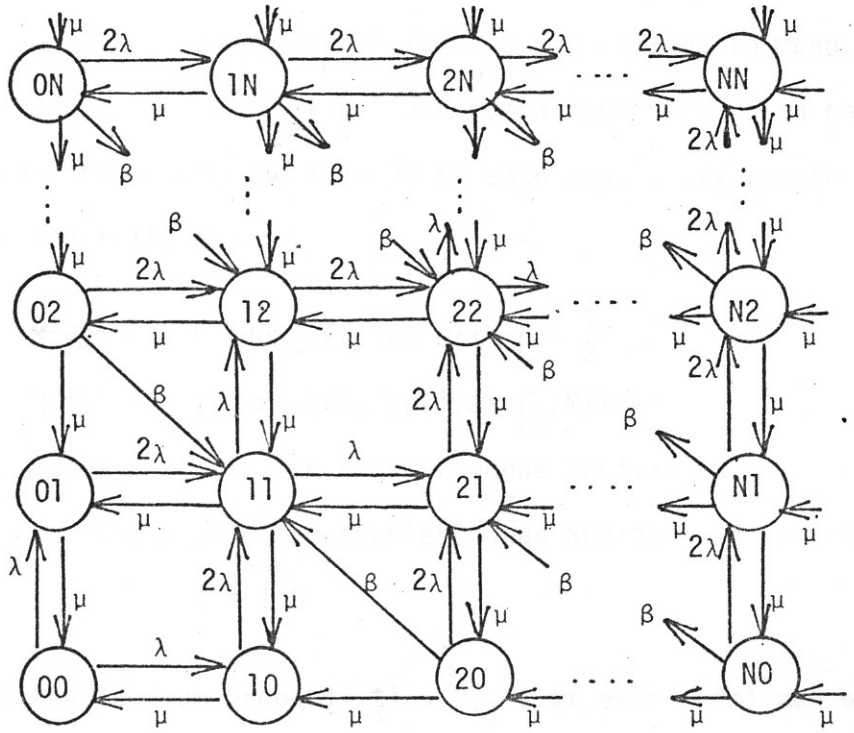


Figure 4-28: State-Transition-Rate Diagram for Figure 4-27.

A transfer from state (i-1,j+1) or (i+1,j-1) to state (i,j) initiated when $|i-j| > 1$. The recursive algorithm in section 4.3.3 can be similarily applied to Figure 4-28. The transition of β poses no additional difficulty since the

GBE of state (i,j) in the folded state diagram (Figure 4-23) still involves only one unknown state (i,j-1) at its bottom. Thus $P_{i,j-1}$ can be expressed as function of P_{ij} and the probabilities of the states above and to the right of state (i,j). This satisfies the property used in the algorithm (Section 4.3.3).

4.4 Comparison of Performance

Performance measures obtained by analyzing the above queueing models include the utilization of the processors (U), the average queue length (L), and the mean job turnaround time (TT). By definition, the utilization of the processors in each of the open models is $U \triangleq \lambda/\mu$ for any given λ and μ. The average queue length is directly related to mean response time by Little's result, $\lambda \times TT = L$. Our comparisons are based on the analysis of the mean job turnaround time TT of each model. We denote each of the models by the following

Model A - random arrival with $p = \frac{1}{2}$

Model B - alternating arrival $E_2/M/1$

Model C - join the shorter queue arrival

Model D - join the shorter queue arrival and channel transfer $\beta = 1.0$.

The state equilibrium probabilities P_{ij}'s of each model can be obtained as described in the previous section. The average queue length L can then be computed using $L \triangleq \sum_{i=1}^{N} \sum_{j=1}^{N} i\, P_{ij} \triangleq \sum_{i=1}^{N} \sum_{j=1}^{N} j\, P_{ij}$. The mean turnaround time TT is calculated directly from Little's law $TT = L/\lambda$.

We also introduce two limiting cases (Model E and Model F) of the above models for comparison purposes. Model E represents the M/M/2, the special case when $\beta = \infty$. Model F is a M/M/1 with same arrival rate 2λ but only one processor with twice processing speed (2μ). The standard solution [KLEI75] for turnaround time TT of Model E and Model F are given by

$$TT_E = \frac{\mu}{(\mu+\lambda)(\mu-\lambda)}$$

and $$TT_F = \frac{1}{2(\mu-\lambda)} .$$

Table 4-1 and Figure 4-29 show the comparison of the mean turnaround time TT for the six models (A, B, C, D, E, F) as a function of λ. In each

Model	Description of Model	λ	.1	.2	.3	.4	.5	.6	.7	.8	.9
A	random arrival M/M/1	TT_A	1.11	1.25	1.43	1.67	2.00	2.50	3.33	5.00	10.00
B	alternating arrival E_2/M/1	TT_B	1.03	1.10	1.22	1.38	1.62	1.98	2.60	3.84	7.59
C	join the shorter queue without jockeying	TT_C	1.02	1.06	1.14	1.26	1.42	1.68	2.10	2.95	5.47
D	join the shorter queue with jockeying, β=1.0	TT_D	1.01	1.05	1.12	1.23	1.39	1.63	2.05	2.88	5.35
E	M/M/2, $\beta=\infty$	TT_E	1.01	1.04	1.09	1.19	1.33	1.56	1.96	2.78	5.26
F	processor with twice the speed	TT_F	0.56	0.63	0.71	0.83	1.00	1.25	1.67	2.50	5.00

Table 4-1: Comparison of Mean Turnaround Time for Models A, B, C, D, E, F.

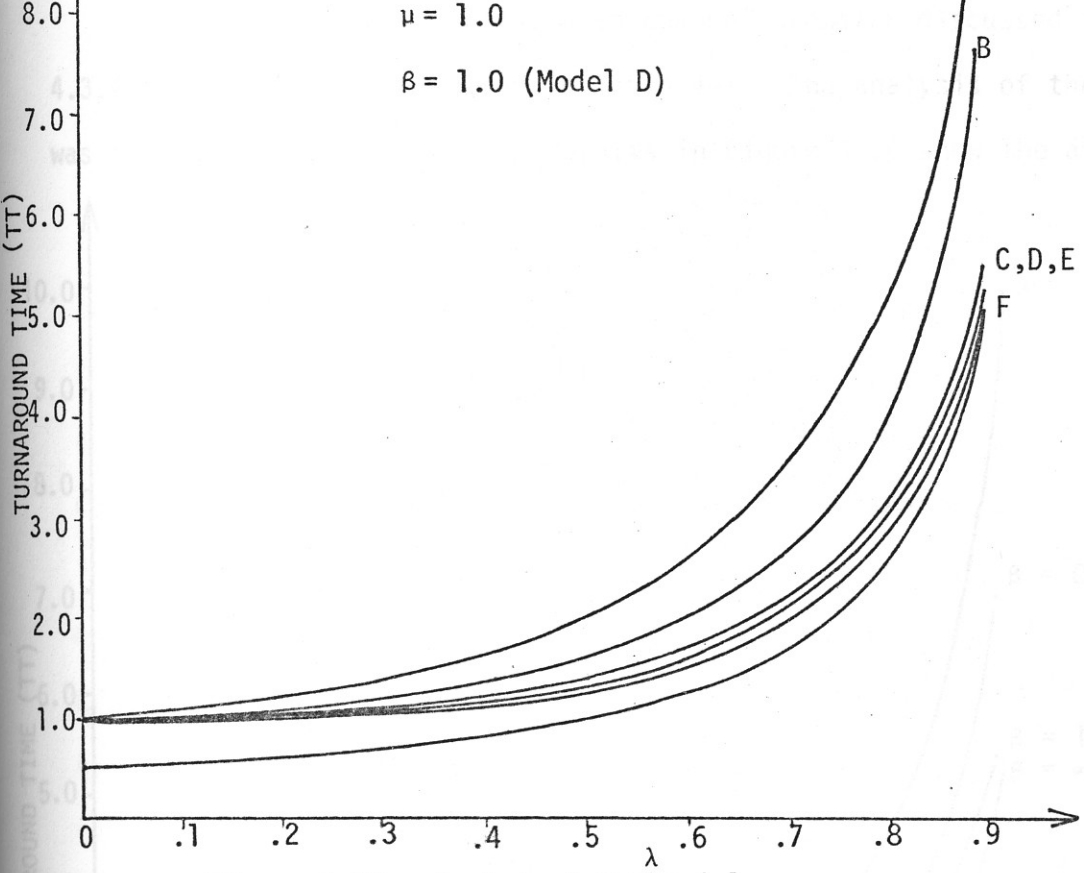


Figure 4-29: A plot of Table 4-1

case μ is set of 1 and λ varies from 0.1 to 0.9. The channel rate in Model D is β = 1.0. The results indicate that for each choice of parameter λ

$$TT_A \geq TT_B \geq TT_C \geq TT_D \geq TT_E \geq TT_F.$$

We also notice that under heavy load conditions the performance of the distributed models with load balancing (Models C, D, E) is significantly better than the system with random arrival policy. Furthermore, the performance rapidly approaches that of Model F, which is the best that can be achieved. The result that Model C closely approximates Model D indicates that the channel is almost redundant due to the load having been balanced by the job dispatcher.

The case of random arrival with channel transfer discussed in section 4.3.4 is not directly compared in Table 4-1. The analysis of the model was done with simulation. The results in Figure 4-30 show the average

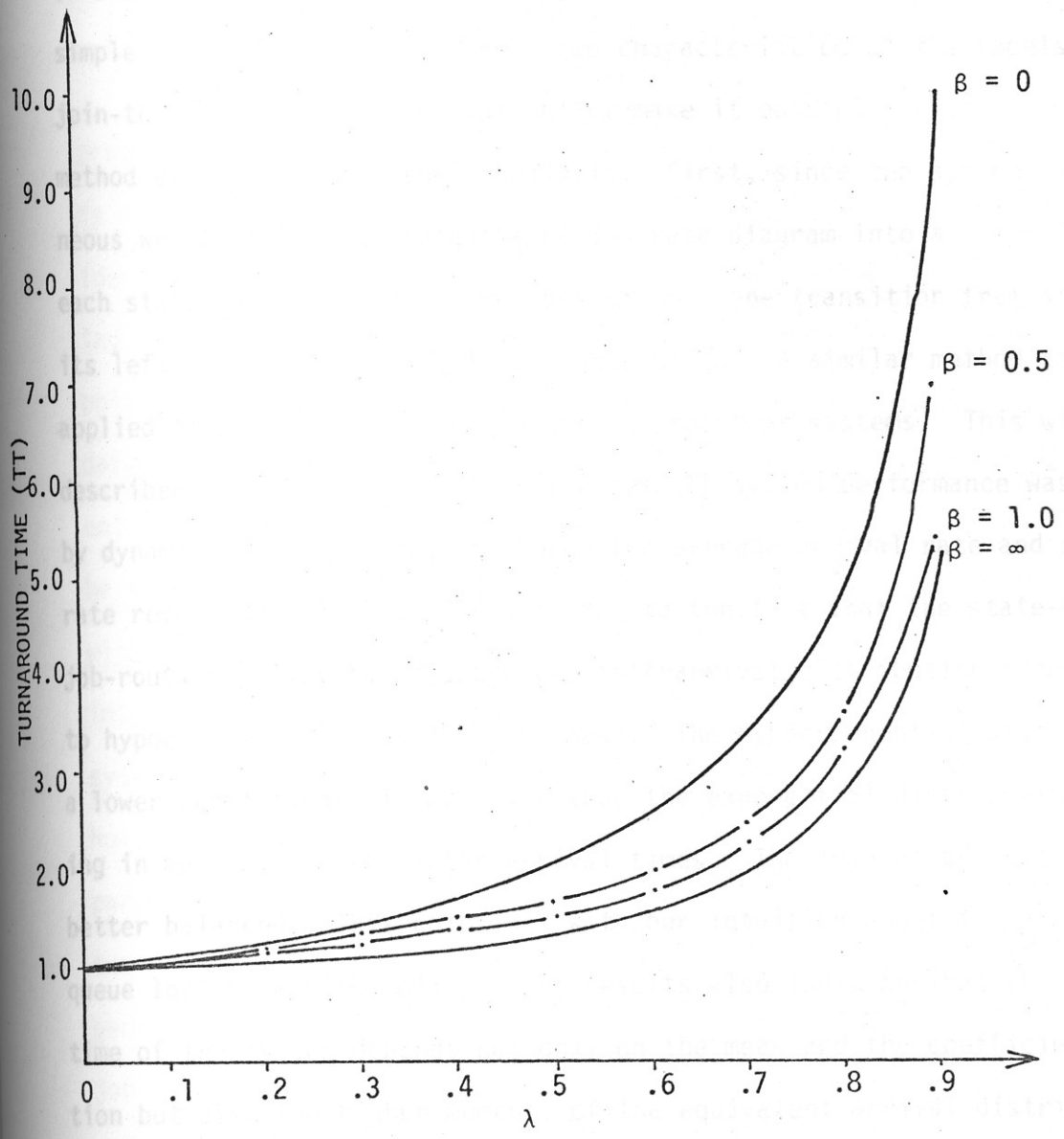


Figure 4-30: Comparison of Mean Turnaround Time TT for the Model with Random Arrival and Channel Transfer.

turnaround time TT as a function of λ and β. It indicates that a transfer channel with $\beta = 1.0$ achieves almost the same performance as the join-the-shorter-queue.

We have shown the impact of job routing strategies on homogeneous two-processor DCS. The results of our analysis suggest that the performance of distributed systems under heavy load can be improved significantly with simple load balancing policies. Two characteristics of the models with join-the-shorter-queue arrival policy make it possible to use the recursive method effectively for their analysis. First, since the systems are homogeneous we can fold the state-transition-rate diagram into a triangle. Second, each state in the state diagram has at most one transition from states to its left or one transition from states below. A similar method can be applied to analyze non-homogeneous two-processor systems. This will be described in the next chapter. The overall system performance was improved by dynamic load balancing, although the average arrival rate and service rate remained unchanged. This is due to the fact that the state-dependent job-routing policy has changed the interarrival distribution from exponential to hypoexponential with the same mean. The hypoexponential distribution has a lower coefficient of variation than the exponential distribution, resulting in more consistent inter-arrival times. The load of the system is thus better balanced. This coincides with our intuition about the join-the-shorter queue load balancing policy. The results also indicate that the turnaround time of the system depends not only on the mean and the coefficient of variation but also the higher moments of the equivalent arrival distribution. It would be an interesting problem to model the effect of the higher moments of arrival distributions on the system performance.

CHAPTER V

HETEROGENEOUS SYSTEM

We have studied various homogeneous distributed systems in the previous chapter. Although homogeneous systems are interesting, it is often the case that we have to deal with systems which involve non-identical processors. In this chapter we extend the idea of job routing to heterogeneous systems. Again, two classes of job routing policies are presented: non-deterministic and deterministic. The non-deterministic policies that we will investigate include state independent routing and state dependent routing. The Coxian staging method will be shown to be useful in describing the arrival processes associated with non-deterministic routing policies. The deterministic policies introduced in this chapter are the maximum ratio policy, the minimum system time policy, and the maximum throughput policy. We will show how these policies can be formally expressed as functions of the system parameters. The recursive solution technique used for the analysis of two-processor homogeneous systems is generalized for two-processor heterogeneous systems with special properties. The models with the deterministic job routing policies that we propose are shown to have such properties and thus can be efficiently analyzed. Comparisons of performance based on the solution of two-processor open models with various job routing strategies are then presented. Lower bound models for average system turnaround time are also introduced for comparison purposes. The analysis shows that the maximum throughput job routing strategy gives the best performance and it is conjectured to be optimal.

5.1 Models with Non-Deterministic Routing Policies

Consider the multiple processor heterogeneous systemin Figure 5-1. An arriving job is routed randomly to processor i with probability P_i.

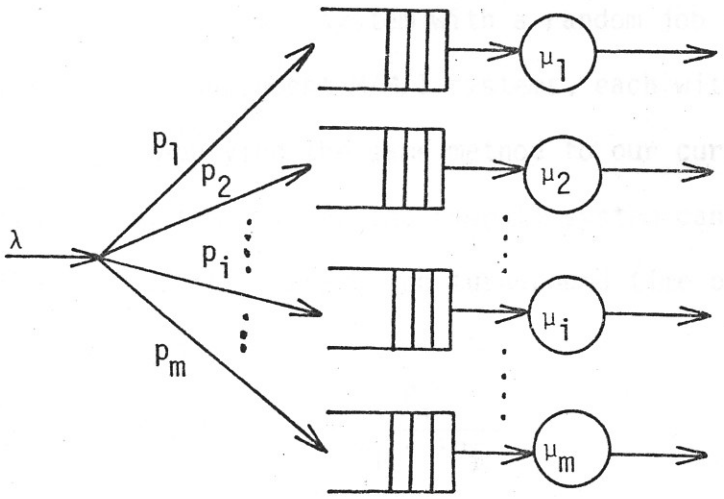


Figure 5-1: A Heterogeneous System with Random Job Routing.

This is called the random job routing policy. p_i can be chosen arbitrarily or as a function of system parameters. A random job routing policy is state independent if the branching probabilities $p_{i's}$ are fixed at all times. On the other hand we say the job routing strategy is state dependent if $p_{j's}$ are functions of the workload distribution of the system.

5.1.1 State independent job routing. The state independent job routing policy is the most commonly used arrival strategy in queueing network analysis. For open models with the exponential distribution assumption, the state equilibrium probabilities have a simple product form solution [JACK 63, GORD 67]. In the case of closed models, the

local balance property exists for systems with exponential service distributions or special priority rules and efficient solution techniques have been found [CHAN 72, BASK 75, BUZE 73].

Using the Coxian staging method in section 4.3.1, we showed how an m-processor homogeneous system with a random job routing policy could be analyzed as m independent M/M/1 systems, each with an equivalent job arrival rate of λp_i. Applying the same method to our current model, the average job trunaround time TT for the overall system can be expressed as the weighted sum of the average job turnaround time on the individual systems, which is

$$TT = \sum_{i=1}^{m} \frac{P_i}{\mu_i - \lambda p_i} \qquad (5\text{-}1)$$

The minimum value for TT under the following constraints can be computed analytically:

$$0 \leq p_i \leq 1;$$

$$\sum_{i=1}^{m} p_i = 1;$$

$$\lambda p_i < \mu_i.$$

The last constraint ensures that non of the queues will be saturated. We note that in this state independent job routing strategy one might foolishly choose a p_i such that $\lambda p_i \geq \mu_i$, even though λ is smaller than the total processing power $\sum_{j=1}^{m} \mu_j$. This leads us to consider reasonable values for the branching probabilities p_i's. It is natural to select p_i such that it is directly proportional to the processing speed of processor i, i.e. $p_i = \frac{\mu_i}{\sum_{j=1}^{m} \mu_j}$. Although this choice of p_i may not result in mini-

mum job turnaround time, it does guarantee that all queues will never be saturated as long as $\lambda < \sum_{j=1}^{m} \mu_j$.

This can be shown as follows:

$$\lambda p_1 = \lambda \frac{\mu_i}{\sum_{j=1}^{m} \mu_j} = \mu_i \frac{\lambda}{\sum_{j=1}^{m} \mu_j} < \mu_i .$$

The average turnaround time TT can then be expressed as

$$TT = \sum_{i=1}^{m} \frac{p_i}{\mu_i - \frac{\lambda \mu_i}{\sum_j \mu_j}}$$

$$= \sum_{i=1}^{m} \frac{\frac{\mu_i}{\sum_j \mu_j}}{\mu_i - \frac{\lambda \mu_i}{\sum_j \mu_j}}$$

$$= \frac{m}{\sum_j \mu_j - \lambda} .$$

Let P_{n_i} denote the marginal steady state probability of the ith M/M/1 system where n_i is the number of jobs in the ith queue. P_{n_i} can be expressed as

$$P_{n_i} = \left(\frac{\lambda \frac{\mu_i}{\sum_j \mu_j}}{\mu_i} \right)^{n_i} P_o$$

$$= \left(\frac{\lambda}{\sum_j \mu_j} \right)^{n_i} P_o .$$

It is interesting to notice that the global state equilibrium probabilities $P_{n_1 n_2 \ldots n_m}$ have the product form

$$P_{n_1 n_2 \ldots n_m} = \left(\frac{\lambda}{\sum_j \mu_j}\right)^{n_1} \left(\frac{\lambda}{\sum_j \mu_j}\right)^{n_2} \cdots \left(\frac{\lambda}{\sum_j \mu_j}\right)^{n_m} P_{oo\ldots o}$$

$$= \left(\frac{\lambda}{\sum_j \mu_j}\right)^{n_1+n_2+\ldots+n_m} P_{oo\ldots o} .$$

Using the standard formula for the average queue length L_i for each M/M/1 queue, the L_i are found to be identical and given by

$$L_i = \frac{\delta_i}{1-\delta_i}$$

$$= \frac{\lambda}{\sum_j \mu_j - \lambda} \quad \ldots \quad \text{where } \delta_i = \frac{\lambda}{\sum_j \mu_j} .$$

This state independent job routing policy with proportional job branching probabilities will be compared with other deterministic state dependent job routing strategies proposed in section 5.2.

5.1.2 State dependent job routing. The state of a system is defined as the workload distribution in the system. A job routing policy is state dependent if the branching probabilities p_i's are functions of the queue lengths. It is often desired to choose p_i such that it is inversely proportional to the queue length of the ith processor, i.e. $p_i = G \frac{1}{i+1}$ where G is the normalizing constant for all p_i. This policy is sometimes referred to as a discouraged arrival policy [KLEI75] or as the functional job routing [TOWS75]. It was shown [TOWS77] that the equilibrium state probabilities in such systems take a modified product form solution if the

functional branching probabilities p_i belong to a special class of linear functions. However, there exists no efficient algorithm to calculate the normalizing constant for the steady state probabilities.

In this section we will analyze a particular case of the discouraged arrival model. Our reasons for analyzing this model are twofold. First, we believe that the model reflects the actual implementation of some distributed systems. Second, we will show through the analysis that the Coxian staging method is useful in describing the arrival process in models with state dependent routing policies. Figure 5-2 is a special

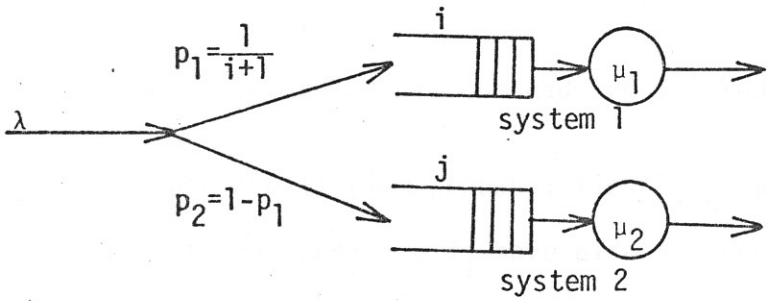


Figure 5-2: A Special Model with State Dependent Job Routing.

case of the models with non-deterministic state dependent job routing. The model consists of two systems. The branching probability p_1 to system 1 is set to $\frac{1}{i+1}$, where i is the number of jobs in queue 1. We can interpret the model as a system composed of one fast and one slow processor with mean processing rates μ_1 and μ_2 respectively. Furthermore, although μ_1 is larger than μ_2, processor 1 can accommodate fewer jobs since the size of its fast memory is limited. By setting $p_1 = \frac{1}{i+1}$, the rate of job arrivals to system 1 is greatly reduced as the queue length i increases. However, we want to send every job to system 1 if

its queue is empty. Processor 1 can be viewed as a master processor which is independent of processor 2, while processor 2 is regarded as a slave whose arrival rate is a function of i. The model can be decomposed into two separate systems as shown in Figure 5-3. System 1 is an independent single queue model with discouraged arrivals. Standard

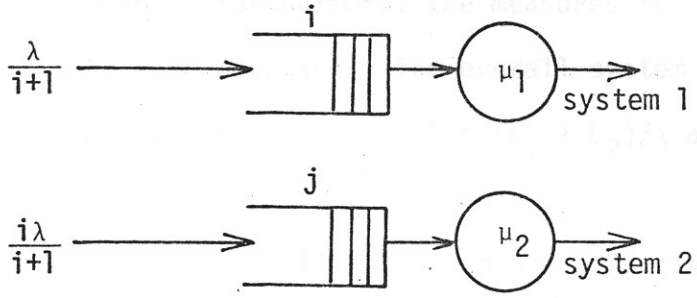


Figure 5-3: Separated Discouraged Arrival Model.

solution for the state equilibrium probabilities $P_{i's}$, average queue 1 length L_1, and mean turnaround time T_1 have already been derived [KLEI 75]. P_i is given by

$$P_i = \frac{(\frac{\lambda}{\mu_1})^i \, e^{-\frac{\lambda}{\mu_1}}}{i!}$$

Since P_i is poisson, the average queue length L_1 can be expressed as $L_1 = \sum_{i=0}^{\infty} i \, P_i = \lambda/\mu_1$. Let λ_1 denote the equivalent arrival rate to queue 1. λ_1 can be obtained by equating the arrival and departure rates as follows:

$$\lambda_1 = \mu_1(1-P_0)$$
$$= \mu_1(1-e^{-\frac{\lambda}{\mu_1}}). \qquad (5\text{-}3)$$

Using Little's law, the mean turnaround time T_1 for system 1 is given by

$$T_1 = \frac{L_1}{\lambda_1} = \frac{\lambda}{\mu_1^2(1-e^{-\frac{\lambda}{\mu_1}})} .$$

We will now show that similar results can be obtained by using the Coxian staging method. Furthermore, the measures for system 2 will be computed using the same approach. The overall system turnaround time TT will be calculated directly from $TT = (L_1 + L_2)/\lambda$ after L_1 and L_2 are determined.

Recall that in chapter III we showed how an arrival process can be described using a Coxian staging diagram. Figure 5-4 is the Coxian staging

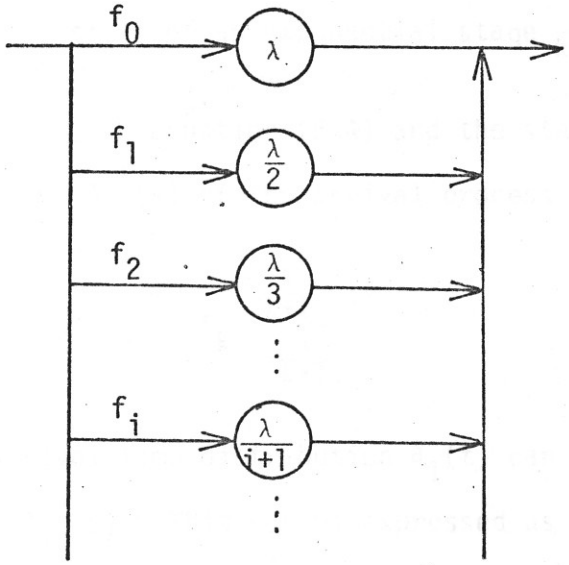


Figure 5-4. Coxian Staging Diagram for Arrival Process of System 1.

diagram representing the arrival process for system 1. f_i denotes the probability that the inter-arrival rate is $\frac{\lambda}{i+1}$. During a unit time interval the average number of arrivals is λ_1. Let us assume that among the arrivals, n_{1i} of them arrive at rate $\frac{\lambda}{i+1}$. Therefore, during this

unit time interval

$$f_i = \frac{n_{1i}}{\lambda_1}$$

$$= \frac{\lambda}{i+1} P_i / \sum_{k=0}^{\infty} \frac{\lambda}{k+1} P_k$$

$$= \frac{\lambda}{i+1} \cdot \frac{(\frac{\lambda}{\mu_1})^i e^{-\frac{\lambda}{\mu_1}}}{i\,!} / \mu_1 (1 - e^{-\frac{\lambda}{\mu_1}})$$

$$= \frac{e^{-\frac{\lambda}{\mu_1}}}{1-e^{-\frac{\lambda}{\mu_1}}} \frac{(\frac{\lambda}{\mu_1})^{i+1}}{(i+1)!} \qquad (5\text{-}4)$$

The s-transform of an exponential stage $\frac{\lambda}{k+1}$ is $\frac{\frac{\lambda}{k+1}}{s+\frac{\lambda}{k+1}}$. Given the probabilities f_i in equation (5-4) and the staging diagram in Figure (5-4), the s-transform $A_1^*(s)$ of the arrival process is

$$A_1^*(s) = \sum_{k=0}^{\infty} f_k \frac{\frac{\lambda}{k+1}}{s+\frac{\lambda}{k+1}} .$$

The inter-arrival time distribution $a_1(t)$ can be found by inverting the s-transform $A_1^*(s)$. This can be expressed as

$$a_1(t) = \sum_{k=0}^{\infty} f_k \frac{\lambda}{k+1} e^{-\frac{\lambda}{k+1} t}$$

$$= \frac{e^{-\frac{\lambda}{\mu_1}}}{1-e^{-\frac{\lambda}{\mu_1}}} \sum_{k=0}^{\infty} \frac{\lambda}{k+1} \frac{(\frac{\lambda}{\mu_1})^{k+1}}{(k+1)!} e^{-\frac{\lambda}{k+1} t}$$

The average inter-arrival time T_1 is the expected value E[t] of the inter-arrival time and becomes

$$T_1 = E[t] = \int_0^\infty t\, a_1(t)\, dt$$

$$= \frac{e^{-\frac{\lambda}{\mu_1}}}{1-e^{-\frac{\lambda}{\mu_1}}} \sum_{k=0}^{\infty} \frac{1}{\mu_1} \frac{(\frac{\lambda}{\mu_1})^k}{k!}$$

$$= \frac{1}{\mu_1(1-e^{-\frac{\lambda}{\mu_1}})}.$$

By taking the inverse of T_1 we get the equivalent mean arrival rate $\lambda_1 = \mu_1(1-e^{-\frac{\lambda}{\mu_1}})$, which is the same as equation (5-3).

The arrival distribution $a_2(t)$ for system 2 can be derived similarly. Figure 5-5 is the Coxian staging diagram. g_i is a function of P_i and denotes the probability that the inter-arrival rate is $\frac{i\lambda}{i+1}$. By analogy with system 1 we define $g_i = \frac{n_{2i}}{\lambda_2}$, where n_{2i} is the number of arrivals at

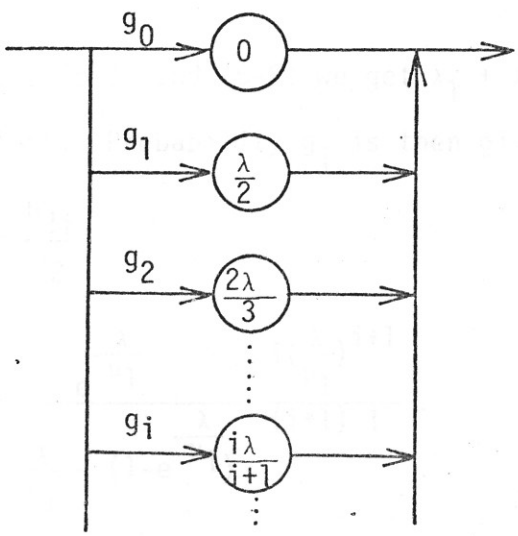


Figure 5-5: Coxian Staging Diagram for Arrival Process of System 2.

rate $\frac{i\lambda}{i+1}$ during a unit time interval. Given the state equilibrium probabilities P_i for system 1, n_{2i} becomes

$$n_{2i} = \frac{i\lambda}{i+1} P_i$$

$$= \frac{i\lambda}{i+1} \cdot \frac{(\frac{\lambda}{\mu_1})^i \, e^{-\frac{\lambda}{\mu_1}}}{i!} .$$

The equivalent mean arrival rate λ_2 to system 2 can be evaluated as follows:

$$\lambda_2 = \sum_{k=0}^{\infty} \frac{k\lambda}{k+1} P_k$$

$$= \sum_{k=0}^{\infty} \frac{k\lambda}{k+1} \frac{(\frac{\lambda}{\mu_1})^k}{k!} e^{-\frac{\lambda}{\mu_1}}$$

$$= \mu_1 \, e^{-\frac{\lambda}{\mu_1}} \int \sum_{k=0}^{\infty} \frac{k(\frac{\lambda}{\mu_1})^k}{k!} \, d(\frac{\lambda}{\mu_1})$$

$$= \lambda - \mu_1(1 - e^{-\frac{\lambda}{\mu_1}}) . \qquad (5\text{-}5)$$

Adding equations (5-3) and (5-5) we get $\lambda_1 + \lambda_2 = \lambda$. This confirms the above derivations. Probability g_i is then given by

$$g_i = \frac{n_{2i}}{\lambda_2}$$

$$= \frac{e^{-\frac{\lambda}{\mu_1}}}{\frac{\lambda}{\mu_1} - (1-e^{-\frac{\lambda}{\mu_1}})} \frac{i(\frac{\lambda}{\mu_1})^{i+1}}{(i+1)\,!} .$$

The s-transform $A_2^*(s)$ of the Coxian staging diagram in Figure 5-5 becomes

$$A_2^*(s) = \sum_{i=0}^{\infty} \frac{e^{-\frac{\lambda}{\mu_1}}}{\frac{\lambda}{\mu_1} - (1-e^{-\frac{\lambda}{\mu_1}})} \frac{i(\frac{\lambda}{\mu_1})^{i+1}}{(i+1)\,!} \frac{\frac{i\lambda}{i+1}}{s + \frac{i\lambda}{i+1}} .$$

Finally, we get the inter-arrival time distribution $a_2(t)$ for system 2 by inverting $A_2^*(s)$, which gives

$$a_2(t) = \frac{e^{-\frac{\lambda}{\mu_1}}}{\frac{\lambda}{\mu_1}(1-e^{-\frac{\lambda}{\mu_1}})} \sum_{i=0}^{\infty} \frac{i(\frac{\lambda}{\mu_1})^{i+1}}{(i+1)!} \frac{i\lambda}{i+1} e^{-\frac{i\lambda}{i+1}t} .$$

It can be shown that the mean inter-arrival time T_2 for system 2 is $1/\lambda_2$ as expected from (5-5).

$$T_2 = E[t] = \int_0^{\infty} t\, a_2(t)\, dt$$

$$= \frac{e^{-\frac{\lambda}{\mu_1}}}{\frac{\lambda}{\mu_1} - (1-e^{-\frac{\lambda}{\mu_1}})} \sum_{i=0}^{\infty} \frac{1}{\lambda} \frac{(\frac{\lambda}{\mu_1})^{i+1}}{i!} .$$

$$= \frac{1}{\lambda - \mu_1(1-e^{-\frac{\lambda}{\mu_1}})}$$

$$= \frac{1}{\lambda_2} .$$

By using the Coxian staging method we have shown that the discouraged arrival two-processor system can be reduced to two G/M/1 systems with known arrival process distributions $a_1(t)$ and $a_2(t)$. Although it may

be tedious, it is theoretically possible to analyze any G/M/1 model. The turnaround time for a G/M/1 system can be expressed as a function of σ, where σ is the root of equation $A^*(\mu-\mu\sigma) = \sigma$ [KLEI 75]. Alternately, the recursive solution technique used in Chapter IV for the M/G/1 models can be applied to these G/M/1 systems also. After the average queue length L_1 and L_2 are calculated for each G/M/1 system, the average turnaround time TT of the two-processor system is then given by

$$TT = \frac{L_1 + L_2}{\lambda} .$$

The performance of this model is not directly compared with the other routing policies since the restriction of faster processing rate and smaller memory size placed on one of the processors is not compatible with our other models. However, the above derivation demonstrates and important fact. Job routing strategies may create new arrival distributions. In many cases these new arrival distributions can be derived by using the Coxian staging method. Therefore, not only may the mean, the coefficient of variation, and other moments of the new distributions be computed, but the system can be decomposed into separate G/M/1 models.

5.2 Models with Deterministic Routing Policies

Consider the multiple processor heterogeneous system in Figure 5-6. If an arriving job finds the system in state $S = (n_1, n_2, \ldots, n_m)$, then the job will be dispatched (routed) to queue $q(S,C)$, where $q(S,C)$ is determined by a system criterion function C. We use the notation $C(n_1, n_2, \ldots, n_i+1, \ldots, n_m)$ to denote the value of the function when the job is sent to the ith queue. n_s represents the number of jobs in queue s. The job

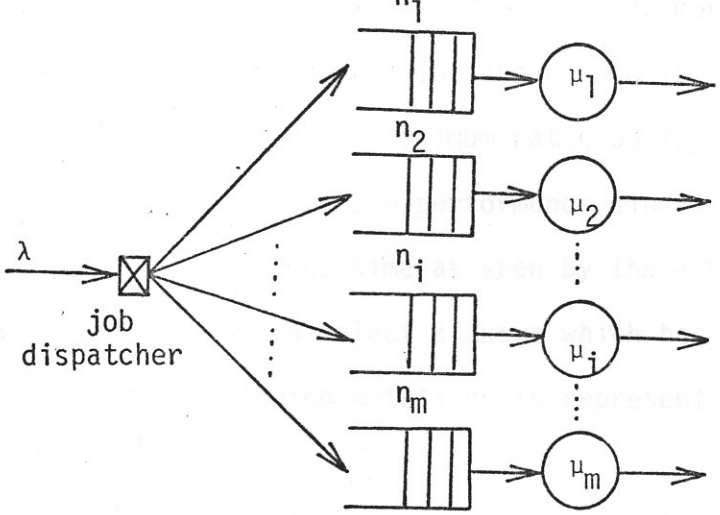


Figure 5-6: A Heterogeneous System with Deterministic Job Routing.

dispatching strategy is deterministic since an arriving job is always routed to a processor in accordance with the deterministic criterion function of the job dispatcher. We will investigate three state dependent routing policies for the above model: the maximum ratio policy, the minimum system time policy, and the maximum throughput policy. We will express the criterion function C in terms of the system parameters for each policy. It will be shown that two-processor heterogeneous systems with these routing policies can be solved using a generalized recursive solution technique to be described in section 5.3.

5.2.1 Maximum ratio policy. The policy can be stated as follows:

Assume the system is in state $S = (n_1, n_2, \ldots, n_m)$.

1) An arriving job is sent to queue i if the service rate to queue length ratio $\mu_i/(n_i+1) = \max \{\mu_k/(n_k+1) \mid k=1,2,\ldots,m\}$.

2) If the maximum $\mu_i/(n_i+1)$ is not unique, the job dispatcher selects from among the tied queues the one with maximum μ_i.

The choice of how to resolve ties when two or more queues have the same $\mu/(n+1)$ ratio was arbitrary. The above rule is standard for all policies whenever a tie exists. The maximum ratio policy is based on an individual user's view as how to improve performance since the inverse of the ratio is the average turnaround time as seen by the arriving job. It is natural for the arriving job to select a queue which has a maximum $\mu/(n+1)$ ratio. We will use the following notation to represent the maximum ration policy (POLICY I).

POLICY I: Maximum Ratio Policy

$$q(S,C) = \begin{cases} i & \text{if } \max\{C(n_1,n_2,\ldots n_k+1,\ldots,n_m) \mid k=1,2,\ldots,m\} = \dfrac{\mu_i}{n_i+1} \\ & \text{and } \dfrac{\mu_i}{n_i+1} \neq \dfrac{\mu_j}{n_j+1} \text{ for } i \neq j, \\ & \text{where } C(n_1,n_2,\ldots,n_k+1,\ldots,n_m) \triangleq \dfrac{\mu_k}{n_k+1}. \\ j & \text{otherwise, where } \max\{\mu_k \mid \dfrac{\mu_k}{n_k+1} = \dfrac{\mu_j}{n_j+1}\} = \mu_j. \end{cases}$$

Since the departure transitions in the state-transition-rate diagram are independent of the routing policy, it is convenient to simplify the diagram by only showing the arrival transitions. Figure 5-7 shows a grid which represents the simplified state diagram for a two-processor system with $\mu_1 = 2$ and $\mu_2 = 1$. Each intersection represents a state (i,j). The arrows indicate transitions from state (i,j) to (i',j') of rate λ due to an arrival. The departure transitions from state (i,j) to $(i,j-1)$ and $(i-1,j)$ of rates μ_1 and μ_2 are omitted. It will be shown in section 5.3 that the state diagram satisfies special properties and can be solved efficiently by using a generalized recursive method.

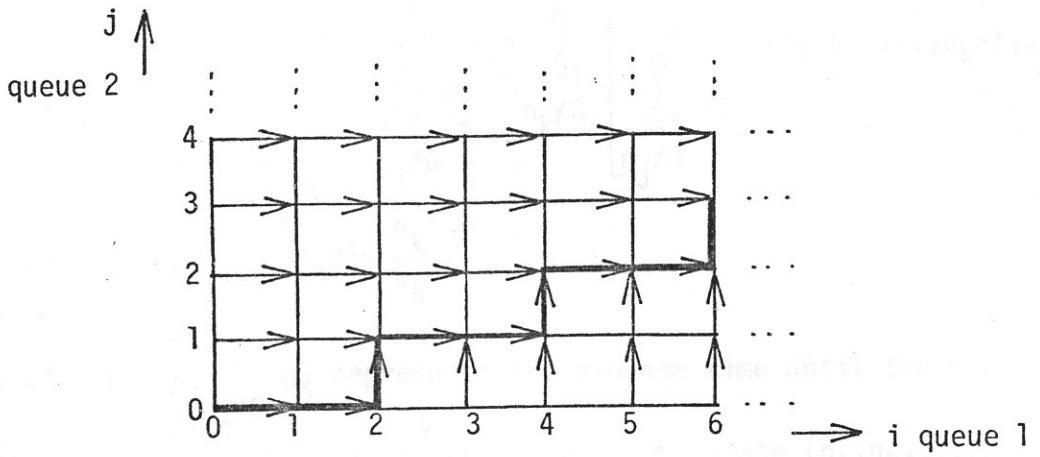


Figure 5-7: Simplified State Diagram for POLICY I when $\mu_1 = 2$ and $\mu_2 = 1$.

5.2.2 Minimum system time policy. The mimimum system time policy is motivated by two goals, the desire to find an optimal routing strategy and the need to model parallel processing. Our analysis shows that although the policy does improve system performance it is not an optimal job routing strategy. Given that the system is in state $S = (n_1, n_2, \ldots, n_m)$, the system time $T(n_1, n_2, \ldots, n_m)$ is defined as the expected time to complete every job already in the system. This definition can be expressed recursively as follows:

$$T(n_1, n_2, \ldots, n_m) = E[\text{time to service every job}]$$

$$= E[\text{time to service the next job}] +$$

$$E[\text{time to service the remaining jobs}]$$

$$= \frac{1}{\sum_{\substack{j=1 \\ n_j \neq 0}}^{m} \mu_j} + \sum_{\substack{k=1 \\ n_k \neq 0}}^{m} \left[\frac{\mu_k}{\sum_{\substack{j=1 \\ n_j \neq 0}}^{m} \mu_j} T(n_1, n_2, \ldots, n_k - 1, \ldots, n_m) \right],$$

$$T(0,0,\ldots,n_k,\ldots,0,0) = \frac{n_k}{\mu_k} .$$

The term $1/\sum_{j=1, n_j \neq 0}^{m} \mu_j$ represents the average time until the next job completes its service given that the system is in state $(n_1, n_2, \ldots, n_m)$. This follows from the assumption that each server has an exponential service distribution. The expression $\mu_k / \sum_{j=1, n_j \neq 0}^{m} \mu_j$ is the probability that the next departing job is from queue k. If a job departs from queue k, then the average remaining system time is $T(n_1, n_2, \ldots, n_k - 1, \ldots, n_m)$. Thus, the average system time can be defined recursively using the first equation. The second equation defines the boundary conditions for the recurrence relation. If we choose $T(n_1, n_2, \ldots, n_m)$ as our criterion function, then the mimimum system time policy (POLICY II) can be expressed formally as

POLICY II: Minimum System Time Policy

$$q(S,C) = \begin{cases} \text{i} & \text{if } \min \{C(n_1, n_2, \ldots, n_k + 1, \ldots, n_m) \mid k = 1, 2, \ldots, n_m\} = \\ & T(n_1, n_2, \ldots, n_i + 1, \ldots, n_m) \text{ and } T(n_1, n_2, \ldots, n_i + 1, n_m \neq \\ & T(n_1, n_2, \ldots, n_j + 1, \ldots, n_m) \text{ for } i \neq j, \text{ where } C(n_1, n_2, \ldots, \\ & n_k + 1, \ldots, n_m) = T(n_1, n_2, \ldots, n_k + 1, \ldots, n_m). \\ \text{j} & \text{otherwise, where } \max \{\mu_k \mid T(n_1, n_2, \ldots, n_k + 1, \ldots, n_m) = \\ & \qquad T(n_1, n_2, \ldots, n_j + 1, \ldots, n_m)\} = \mu_j. \end{cases}$$

For the special case of a two-processor system, we have

$$T(n_1,n_2) = \frac{1}{\mu_1 + \mu_2} + \frac{\mu_1}{\mu_1 + \mu_2} T(n_1-1,n_2) + \frac{\mu_2}{\mu_1 + \mu_2} T(n_1,n_2-1) ,$$

$$T(n_1,0) = \frac{n_1}{\mu_1} ,$$

and $$T(0,n_2) = \frac{n_2}{\mu_2} .$$

The simplified state diagram for this policy when $\mu_1 = 2$ and $\mu_2 = 1$ is given by Figure 5-8.

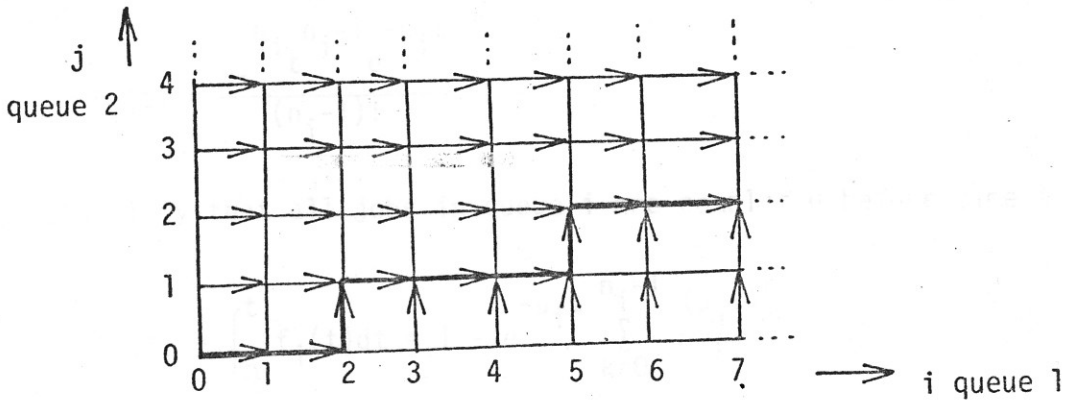


Figure 5-8: Simplified State Diagram for POLICY II when $\mu_1 = 2$ and $\mu_2 = 1$.

The recursive definition of the average system time $T(n_1,n_2,\ldots,n_m)$ can be expressed in an alternate way. If we consider the system time t as a random variable, the average system time can be computed from its distribution. We will show how this distribution can be obtained. Given that we have n_i jobs in queue i, the distribution of the time to complete all jobs is the same as the service time distribution for a single job to pass through n_i stages of servers. This is illustrated

by the diagrams in Figure 5-9.

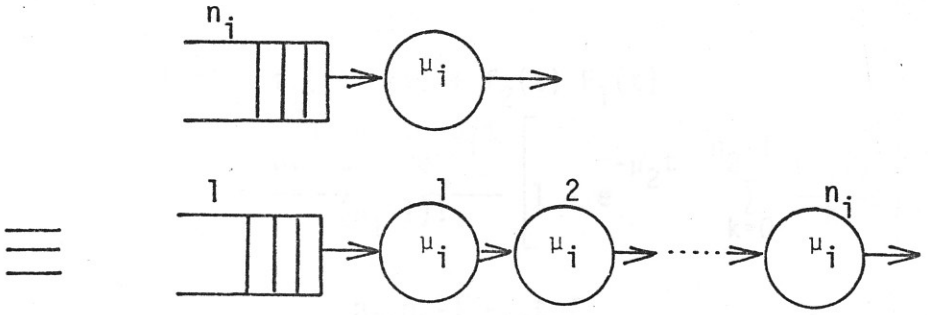


Figure 5-9: Systems with Equivalent System Time Distributions.

The service time distribution is a n_i stage Erlang distribution and is given by

$$f_i(t) = \frac{\mu_i^{n_i} t^{n_i-1} e^{-\mu_i t}}{(n_i-1)!} .$$

The probability that all jobs in queue i are completed before time t is

$$F_i(t) = \int_0^t f_i(t)dt = 1 - e^{-\mu_i t} \sum_{k=0}^{n_i-1} \frac{(\mu_i t)^k}{k!} .$$

The overall system time distribution can be expressed as

$$f(t) = Pr[\text{last completion occurs at time } t]$$

$$= \sum_{i=1}^{m} Pr[\text{last completion occurs at time } t \mid \text{last completion is on processor } i] \; Pr[\text{last completion is on processor } i]$$

$$= \sum_{i=1}^{m} \{f_i(t) \prod_{\substack{j=1 \\ j\neq i}}^{m} F_j(t)\}$$

for the two-processor case

$$f(t) = f_1(t)\,F(t) + f_2(t)\,F_1(t)$$

$$= \frac{\mu_1^{n_1} t^{n_i-1} e^{-\mu_1 t}}{(n_1-1)!}\left[1 - e^{-\mu_2 t}\sum_{k=0}^{n_2-1}\frac{(\mu_1 t)^k}{k!}\right]$$

$$+ \frac{\mu_2^{n_2} t^{n_2-1} e^{-\mu_2 t}}{(n_2-1)!}\left[1 - e^{-\mu_1 t}\sum_{k=1}^{n_1-1}\frac{(\mu_1 t)^k}{1!}\right].$$

The average system time can be found using the above distribution. This becomes

$$T(n_1,n_2) = \frac{n_1}{\mu_1} - \sum_{k=0}^{n_2-1}\frac{\mu_1^{n_1}\,\mu_2^{k}}{(n_1-1)!k!}\,\frac{(n_1+k)!}{(\mu_1+\mu_2)^{n_1+k+1}}$$

$$+ \frac{n_2}{\mu_2} - \sum_{k=0}^{n_1-1}\frac{\mu_2^{n_2}\,\mu_1^{k}}{(n_2-1)!k!}\,\frac{(n_2+k)!}{(\mu_1+\mu_2)^{n_2+k+1}}.$$

Although this expression for $T(n_1,n_2)$ is cumbersome, it yields the same result as the recursive definition of $T(n_1,n_2)$ presented earlier.

An interesting related application of the minimum system time policy is the modeling of parallel processing systems. Figure 5-10 shows a simple parallel processing model. Jobs in the ready queue can be partitioned into independent tasks to be executed separately and in parallel on different processors. A new job is executed only if all tasks of the previous job are completed by the multiprocessor server. The tasks of a job are scheduled in such a way that the average service time of the job is minimized. This scheduling policy is similar to our minimum system

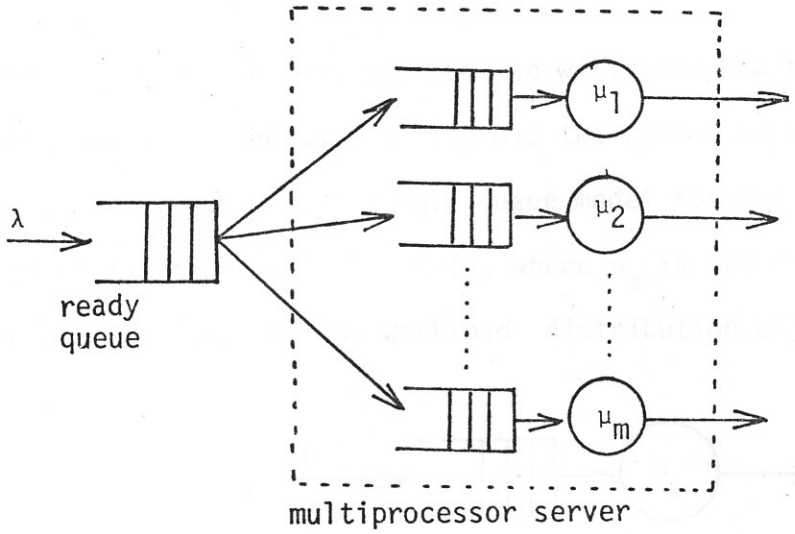


Figure 5-10: Parallel Processing Model.

time strategy. The system time distribution becomes the service distribution of the equivalent system in Figure 5-11. The equivalent mean

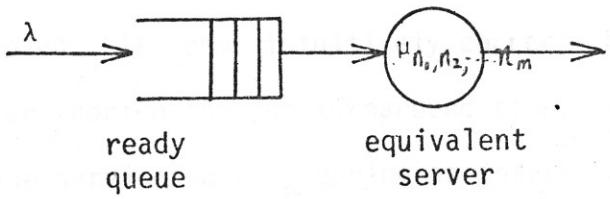


Figure 5-11: Equivalent Parallel Processing Model.

service rate is the inverse of the average minimum system time, where the minimization is done over all possible assignments of the tasks, i.e., $\mu_{n_1,n_2,\ldots,n_m} = 1/\min.\{T(n_1,n_2,\ldots,n_m) \mid \text{all possible assignments}\}$. The reduced system is a simple M/M/1 model with job dependent service rate.

5.2.3. Maximum throughput policy. The two routing policies discussed above are functions of the system queue lengths and processor service rates only. The job dispatcher may also have information available on the job

arrival rate. In this section, we will show how this additional information can be used to improve the performance of the system. Consider an equivalant single queue model for the multiple processor system as shown in Figure 5-12, where μ_e is the equivalent service rate. $\mu_e$depends on the workload distribution among the queues

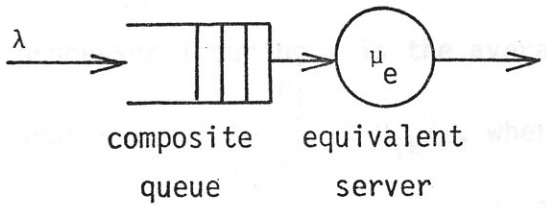


Figure 5-12: Equivalent Single Queue Model.

although this distribution is not explicityly shown in the composite queue. The function of the job dispatcher is to update the workload distribution after an arrival so that the minimum average turnaround time can be achieved. It seems intuitively correct that if we can increase μ_e we can shorten the job turnaround time. We define the equivalent average service rate μ_e during an inter-arrival time as the average system throughput during this interval since the workload distribution can be updated by the dispatcher only after a job arrival. The maximum throughput job routing policy is a strategy that assigns an arrival to a queue such that the expected throughput of the system during the next inter-arrival period is maximized.

The throughput of the system is the sum of the individual throughputs of each processor in the system. The average throughput $TP(n_i)$ of processor i during an inter-arrival period can be derived as follows.

Let Q_{ik} = Pr[k jobs leave processor i during an inter-arrival period of length τ]

Since the service distribution of processor i is exponential, Q_{ik} is Poisson and

$$Q_{ik} = \frac{(\mu_i \tau)^k \, e^{-\mu_i \tau}}{k!}, \qquad 0 \le k \le n_i .$$

The throughput of processor i during τ is the average number of jobs that are serviced and is equal to $\sum_{k=1}^{n_i} k \, Q_{ik}/\tau$, where n_i is the number of jobs in queue i. Because of the Poisson arrival assumption, the probability that the inter-arrival time is τ is given by $f(\tau) = \lambda e^{-\lambda\tau}$. Thus, we can express the average throughput of processor i as

$$TP(n_i) = \int_0^\infty \frac{\sum_{k=1}^{n_i} k \, Q_{ik}}{\tau} f(\tau) d\tau$$

$$= \int_0^\infty \frac{\sum_{k=1}^{n_i} k \, \frac{(\mu_i \tau)^k \, e^{-\mu\tau}}{k!}}{\tau} \cdot \lambda \, e^{-\lambda\tau} d\tau$$

$$= \int_0^\infty \frac{\lambda e^{-\lambda\tau}}{\tau} \left[\sum_{k=1}^{n_i} k \, \frac{(\mu_i \tau)^k \, e^{-\mu_i \tau}}{k!} \right] d\tau$$

$$= \sum_{k=1}^{n_i} \lambda \left(\frac{\mu_i}{\lambda+\mu_i}\right)^k \int_0^\infty e^{-(\lambda+\mu_i)\tau} \cdot \frac{[(\lambda+\mu_i)\tau]^{k-1}}{(k-1)!} d(\lambda+\mu_i)\tau$$

$$= \sum_{k=1}^{n_i} \lambda \left(\frac{\mu_i}{\lambda+\mu_i}\right)^k \left[-e^{-(\lambda+\mu)\tau} \sum_{\ell=0}^{k-1} \frac{[(\lambda+\mu_i)\tau]^\ell}{\ell!} \right]_0^\infty$$

$$= \sum_{k=1}^{n_i} \left(\frac{\mu_i}{\lambda+\mu_i}\right)^k .$$

Given that the system is in state $(n_1,n_2,\ldots,n_m)$, the overall average throughput $TP(n_1,n_2,\ldots,n_m)$ during the next inter-arrival period is the sum of all the individual $TP(n_i)$. This can be expressed as

$$TP(n_1,n_2,\ldots,n_m) = \sum_{i=1}^{m} \sum_{k=1}^{n_i} \left(\frac{\mu_i}{\lambda+\mu_i}\right)^k$$

We now express more formally the maximum throughput routing policy (POLICY III) as we did for the other two policies.

POLICY III: Maximum Throughput Policy

$$q(S,C) = \begin{cases} i & \text{if } \max\{C(n_1,n_2,\ldots,n_k+1,\ldots,n_m) \mid k=1,2,\ldots,m\} = \\ & TP(n_1,n_2,\ldots,n_i+1,\ldots,n_m) \text{ and } TP(n_1,n_2,\ldots,n_i+1,\ldots,n_m) \neq \\ & TP(n_1,n_2,\ldots,n_j+1,\ldots,n_m) \text{ for } i \neq j, \text{ where } C(n_1,n_2,\ldots, \\ & n_k+1,\ldots,n_m) = TP(n_1,n_2,\ldots,n_k+1,\ldots,n_m) \\ j & \text{otherwise, where } \max\{\mu_k \mid TP(n_1,n_2,\ldots,n_k+1,\ldots,n_m) = \\ & \qquad TP(n_1,n_2,\ldots,n_j+1,\ldots,n_m)\} = \mu_j . \end{cases}$$

Figure 5-13 shows the simplified state diagram for a two-processor system. The arrival transitions which are indicated by arrows, are a function of λ, μ_1, μ_2, n_1, and n_2. The major difference between POLICY III and the other two policies (POLICY I and POLICY II) is that it depends on λ.

5.3 Recursive Solution Technique for Two-Processor Systems

We have shown in Chapter IV that the recursive solution method is

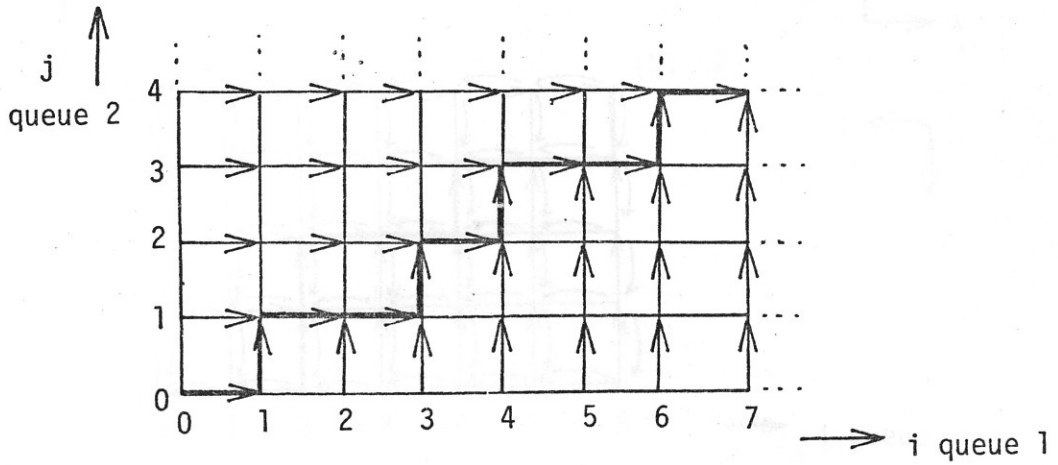


Figure 5-13: State Diagram for POLICY III when $\lambda = 1.5$, $\mu_1 = 2$, and $\mu_2 = 1$.

an efficient numerical technique for the analysis of two-processor homogeneous systems with a join-the-shortest-queue arrival policy. This method can be extended to some classes of heterogeneous systems. In this section we will develop a recursive technique for solving two-processor heterogeneous systems with special properties. The heterogeneous models with deterministic state dependent routing policies discussed in sections 5.2.1 through 5.2.3 will be shown to satisfy these properties. The efficient analysis of such models thus becomes possible.

Consider the state-transition-rate diagram in Figure 5-14 which represents a two-processor heterogeneous system with arrival rate λ and service rates μ_1 and μ_2. The state diagram is not symmetric because of the arrival job routing policy and $\mu_1 \neq \mu_2$. The job dispatching strategy is deterministic, i.e. for each state (i,j) the transition with arrival rate λ always leads to one next state, (i+1,j) or (i,j+1). The job

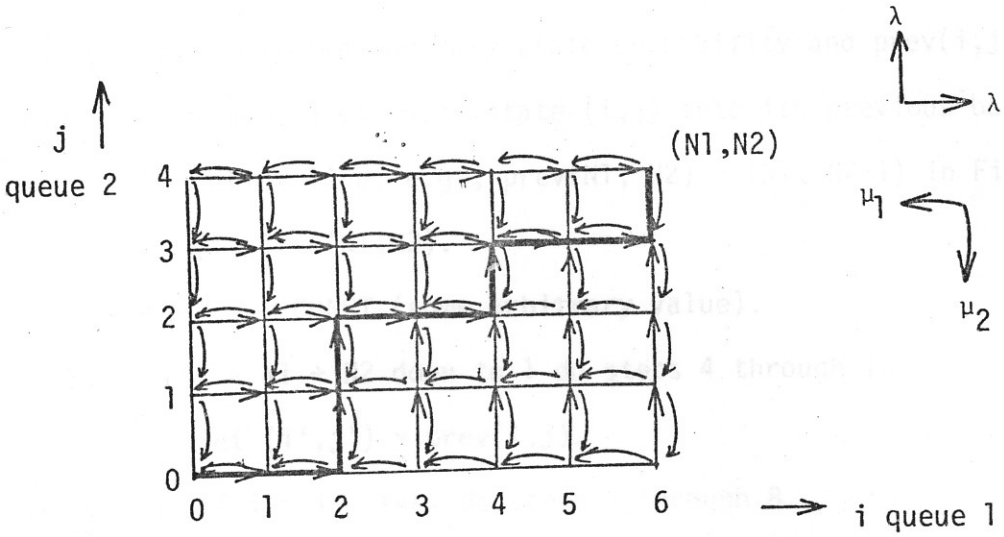


Figure 5-14: Arbitrary Two-Processor Heterogeneous System.

dispatcher assigns the next arriving job to one of the two queues depending upon a predefined deterministic criterion. A well defined job routing policy partitions the state diagram into three classes of states. Figure 5-14 is an example of one such partition. The chain of dark transitions with rate λ will be called the policy line of the system. The states connected by the chain are called balanced states. The states above and the states below the chain are upper unbalanced states and lower unbalanced states respectively.

If the equilibrium state probabilities P_{ij} of the system exist, we can truncate the infinite state diagram to an (N1+1, N2+1) rectangle, with the policy line ending at state (N1, N2). This will introduce negligible error if the system is unsaturated and N1 and N2 are large enough, since P_{ij} will rapidly approach zero for i>>0 and j>>0.

The generalized recursive solution method for two-processor heterogeneous systems is described by the following algorithm and illustrated by a simple example in Appendix B. Let X be a variable

representing an unknown boundary state probability and prev(i,j) a function which maps a balanced state (i,j) into its previous balanced state on the policy line; e.g., prev(N1, N2) = (N1, N2-1) in Figure 5-14.

1. Set (i,j) = (N1,N2).
2. Set P_{ij} = CONST (some arbitrary value).
3. For K = N1 + N2 down to 1 do steps 4 through 15.
4. Set (i',j') = prev(i,j).
5. If i = i' then do steps 6 through 8.
6. Set P_{0j} = X.
7. For I = 1 to i - 1 use the GBE of state (I-1,j) to find an expression for $P_{I,j}$ as a function of X.
8. Use the GBE of state (i-1,j) to solve for X.
9. If j=j' then do steps 10 through 12.
10. Set P_{i0} = X.
11. For J=1 to j-1 use the GBE of state (i,J-1) to find and expression for $P_{i,J}$ as a function of X.
12. Use the GBE of state (i,j-1) to solve for X.
13. Use the GBE of state (i,j) to solve for $P_{i',j'}$.
14. Set (i,j) = (i',j').
15. Continue.
16. Normalize all P_{ij}.

The important characteristics of the state-transition-rate diagram in Figure 5-14 that enable us to efficiently compute the state probabilities are that each upper unbalanced state (i,j) transits right to state (i+1,j) upon an arrival, while each lower unbalanced state (i,j) goes up to state (i,j+1). This means that each upper unbalanced state has an

excess of jobs in queue 2 and therefore jobs are dispatched to queue 1 whenever possible. The complement is true for the lower unbalanced states.

For two-processor systems it is more convenient to use a relational operator R to replace the max and min functions used previously in the definitions of job routing policies. Let C(i+1,j) and C(i,j+1) be the values of the criterion function after a job is dispatched to queue 1 and queue 2 respectively, and assume $\mu_1 > \mu_2$. The routing strategy of max {C(i+1,j), C(i,j+1)}, with ties broken in favor of the faster processor, can then be described as

$$q(S,C) = \begin{cases} 1. \quad \text{i.e., } (i,j) \rightarrow (i+1,j), \text{ if } C(i+1,j) \; R \; C(i,j+1) = \text{true} \\ 2. \quad \text{i.e., } (i,j) \rightarrow (i,j+1), \text{ otherwise, where R is } \geq. \end{cases}$$

The special properties of a job routing policy that enable the above recursive solution technique to be used for the two-processor heterogeneous systems will now be stated formally.

Property I: The job routing policy is deterministic, i.e., if the system is in state (i,j) just prior to an arrival, it moves to state (i+1,j) if C(i+1,j) R C(i,j+1) is true. otherwise, the system goes from state (i,j) to state (i,j+1).

Definition: The <u>policy line</u> of a routing strategy is the chain of arrival transitions starting at state (0,0) given that no departure occurs.

Property II: The policy line partitions the states of the state-transition-rate diagram into two regions. For each state (i,j) in the upper unbalanced region

$$C(i+1,j+n)\ R\ C(i,j+n+1) = \text{true} \quad \forall\ n \geq 0,$$

while for each state (i,j) in the lower unbalanced region

$$C(i+n,j+1)\ R\ C(i+n+1,j) = \text{true} \quad \forall\ n \geq 0.$$

The policy line is by definition a continuous chain. Theorem I below shows that any system satisfying Property I and Property II can be solved using the recursive solution method.

Theorem I: Property I and Property II are sufficient conditions for using the recursive solution method.

Proof: It follows from property I that the system has a policy line which is a continuous chain of states running from state (0,0) to the last boundary state (N1,N2). Therefore for each column i in the state diagram, there exists a j such that the transition from state (i,j) to state (i+1,j) lies on the policy line. This implies that

$$C(i+1,j)\ R\ C(i,j+1) = \text{true}.$$

Using this relation and property II, we know that

$$C(i+1,j+n)\ R\ C(i,j+n+1) = \text{true} \quad \forall\ n \quad 0.$$

This indicates that every arrival transition above the policy line points from left to right. Therefore, the GBE for an upper unbalanced state (i,j) in the recursive algorithm can be expressed as

$$(\lambda+\mu_1+\mu_2)P_{ij} = \lambda P_{i-1,j} + \mu_1 P_{i+1,j} + \mu_2 P_{i,j+1}.$$

Since the recursive algorithm proceeds with the computations from top to bottom and from left to right for the upper unbalanced states, we can express $P_{i+1,j}$ as

$$P_{i+1,j} = \frac{1}{\mu_1}\left[(\lambda+\mu_1+\mu_2)P_{ij} - \lambda P_{i-1,j} - \mu_2 P_{i,j+1}\right],$$

which involves only the known states to the left and above state (i+1,j). A similar analysis applies to the lower unbalanced states. This shows that the recursive solution technique is applicable.

Q.E.D.

We now show that the maximum ratio, minimum system time, and maximum throughput policies for two-processor heterogeneous systems satisfy the above properties. Property I follows from the definition of each policy. Property II will be established by the next three theorems.

Theorem II: The maximum ratio policy satisfies Property II.

Proof: Let R be $\geq$ and $\mu_1 > \mu_2$. By definition of the policy, we have

$$C(i+1,j) = \frac{\mu_1}{i+1} \quad \text{and} \quad C(i,j+1) = \frac{\mu_2}{j+1}$$

If $C(i+1,j) \geq C(i,j+1)$ then $\frac{\mu_1}{i+1} \geq \frac{\mu_2}{j+1}$.

Therefore, $C(i+1,j+n) - C(i,j+n+1) = \frac{\mu_1}{i+1} - \frac{\mu_2}{j+n+1} \geq 0, \quad \forall\, n \geq 0.$

Likewise, if $C(i,j+1) > C(i+1,j)$ then $\frac{\mu_2}{j+1} > \frac{\mu_1}{i+1}$.

Therefore, $C(i+n,j+1) - C(i+n+1,j) = \frac{\mu_2}{j+1} - \frac{\mu_1}{i+n+1} > 0, \quad \forall\, n \geq 0.$

Q.E.D.

Theorem III: The minimum system time policy satisfies Property II.

Proof: Let R be $\leq$ and $\mu_1 > \mu_2$. By definition of the policy we have

$C(i+1,j) = T(i+1,j)$ and $C(i,j+1) = T(i,j+1)$.

We will first show that

if $T(i+1,j) \leq T(i,j+1)$

then $T(i+1,j+n) \leq T(i,j+n+1), \quad \forall\, n \geq 0$

i.e., all the arrival transitions above the policy line go from left to right as shown in Figure 5-15.

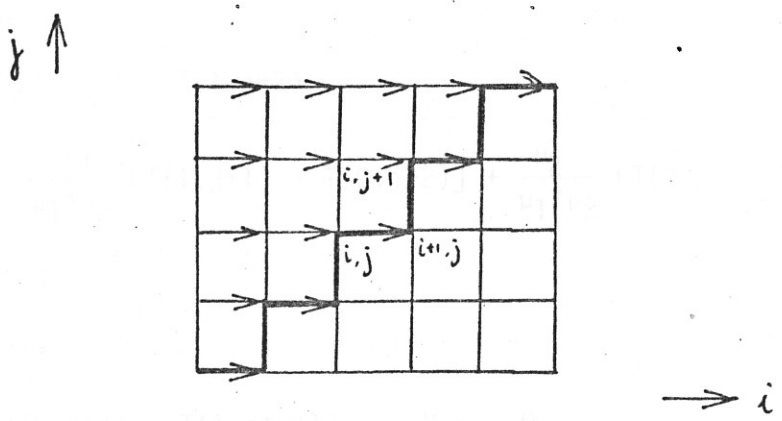


Figure 5-15: Arrival Transition Diagram for Upper Unbalanced States.

We may proceed inductively with the proof as follows.

For $i = 0$ there exists a j such that $T(1,j) \leq T(0,j+1)$. This is due to the transition on the policy line for column $i = 0$. It then follows that

$$T(1,j+1) - T(0,j+2)$$

$$= \frac{1}{\mu_1+\mu_2} [1 + \mu_1 T(0,j+1) + \mu_2 T(1,j)] - T(0,j+2)$$

$$= \frac{1}{\mu_1+\mu_2} [1 + \mu_1 T(0,j+1) + \mu_2 T(0,j+1) - \Delta] - T(0,j+2)$$

$$\ldots\ldots \Delta = \text{positive}$$

$$= \frac{1}{\mu_1+\mu_2} [1 + \mu_1 T(0,j+1) + \mu_2 T(0,j+1) - \Delta] - T(0,j+1) - T(0,1)$$

$$= - \frac{\Delta}{\mu_1+\mu_2} + (\frac{1}{\mu_1+\mu_2} - \frac{1}{\mu_2}) \leq 0.$$

By repeating this process, we can show that

$$T(1,j+n) \leq T(0,j+n+1), \quad \forall\ n \geq 0.$$

Similarly, for $i = 1$ there exists a j such that $T(2,j) \leq T(1,j+1)$. Therefore,

$$T(2,j+1) - T(1,j+2)$$

$$= \frac{\mu_1}{\mu_1+\mu_2} [T(1,j+1) - T(0,j+2)] + \frac{\mu_2}{\mu_1+\mu_2} [T(2,j) - T(1,j+1)]$$

$$\leq 0.$$

In general,

$$T(2,j+n) \leq T(1,j+n+1), \quad \forall\ n \geq 0.$$

Assume $i = k$ and $T(k+1,j+n) \leq T(k,j+n+1), \quad \forall\ n \geq 0.$

For $i = k+1$ we have $T(k+2,j) \leq T(k+1,j+1)$ for some j.

Therefore, $T(k+2,j+1) - T(k+1,j+2)$

$$= \frac{\mu_1}{\mu_1+\mu_2}\,[T(k+1,j+1) - T(k,j+2)] + \frac{\mu_2}{\mu_1+\mu_2}\,[T(k+2,j) - T(k+1,j+1)]$$

$$\leq 0.$$

The general case becomes $T(k+2,j+n) \leq T(k+1,j+n+1) \quad \forall\ n > 0.$

We have now proved by induction that if $T(i+1,j) \leq T(i,j+1)$ then $T(i+1,j+n) \leq T(i,j+n+1)$. The other half of the proof, that if $T(i,j+1) \leq T(i+1,j)$ then $T(i+n,j+1) \leq T(i+n+1,j)$, can be shown analogously. Thus, the minimum system time policy satisfies Property II.

Q.E.D.

Theorem IV: The maximum throughput policy satisfies Property II.

Proof: Let R be $\geq$ and $\mu_1 > \mu_2$. By the definition of the policy we have

$$C(i+1,j) = \sum_{k=1}^{i+1} \lambda\left(\frac{\mu_1}{\lambda+\mu_1}\right)^k + \sum_{k=1}^{j} \lambda\left(\frac{\mu_2}{\lambda+\mu_2}\right)^k$$

$$C(i,j+1) = \sum_{k=1}^{i} \lambda\left(\frac{\mu_1}{\lambda+\mu_1}\right)^k + \sum_{k=1}^{j+1} \lambda\left(\frac{\mu_2}{\lambda+\mu_2}\right)^k.$$

$C(i+1,j) \geq C(i,j+1)$ implies $\left(\frac{\mu_1}{\lambda+\mu_1}\right)^{i+1} - \left(\frac{\mu_2}{\lambda+\mu_2}\right)^{j+1} \geq 0.$

Therefore, $C(i+1,j+n) - C(i,j+n+1)$

$$= \lambda\left(\frac{\mu_1}{\lambda+\mu_1}\right)^{i+1} - \left(\frac{\mu_2}{\lambda+\mu_2}\right)^{j+n+1}$$

$$\geq 0, \text{ since } \left(\frac{\mu_2}{\lambda+\mu_2}\right)^{j+n+1} < \left(\frac{\mu_2}{\lambda+\mu_2}\right)^{j+1}$$

Similarly, if $C(i,j+1) \geq C(i+1,j)$

then $\left(\frac{\mu_2}{\lambda+\mu_2}\right)^{j+1} - \left(\frac{\mu_1}{\lambda+\mu_1}\right)^{i+1} \geq 0:$

Therefore, $C(i+n,j+1) - C(i+n+1,j)$

$$= \lambda\left(\frac{\mu_2}{\lambda+\mu_2}\right)^{j+1} - \left(\frac{\mu_1}{\lambda+\mu_1}\right)^{i+n+1}$$

$$\geq 0.$$

Q.E.D.

One special case of the general state-transition-rate diagram for two-processor heterogeneous systems is when the policy line partitions the state diagram into a set of upper unbalanced states and an empty set of lower unbalanced states as in Figure 5-16. This system has an

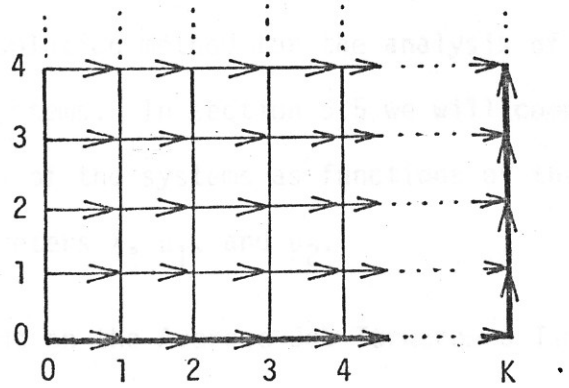


Figure 5-16: State-Transition-Rate Diagram for the Limited Storage Model.

interesting interpretation. The state diagram represents the limited storage two-processor system shown in Figure 5-17. The model is like the discouraged arrival model except that the job routing is deterministic and processor 1 can accept no more than K jobs. The job dispatcher sends all arriving jobs to queue 1 as long as the number of jobs in queue 1 is less than K. A job is dispatched to queue 2 only if queue 1 is full. This limitation occurs commonly in real systems. The system

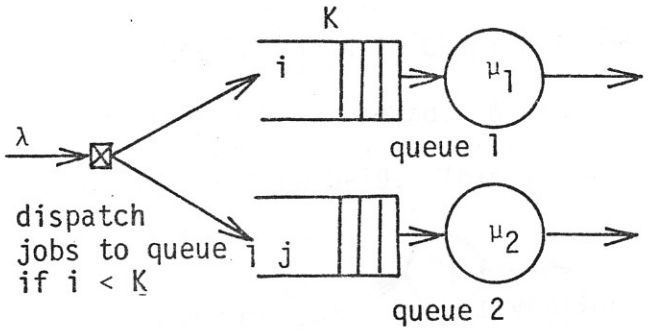


Figure 5-17: Limited Storage Model.

can be analyzed using the recursive technique.

We have shown that each of our deterministic routing policies satisfies Property I and II. It is thus possible to use the generalized recursive solution method for the analysis of these two-processor heterogeneous systems. In section 5.5 we will compare the average job turnaround times of the systems as functions of the routing policy and the system parameters λ, μ_1, and μ_2.

5.4 Lower Bounds on the Average Job Turnaround Time

The average job turnaround time of the single fast processor model in Figure 5-18 can serve as a lower bound for the two-processor hetero-

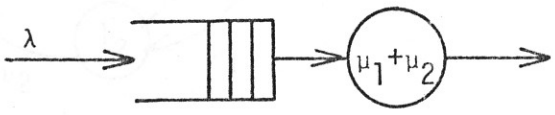


Figure 5-18: Single Fast Processor Model.

geneous models discussed in this chapter. The average turnaround time TT is given by

$$TT = \frac{1}{\mu_1+\mu_2-\lambda} .$$

For the homogeneous models in Chapter IV we used a standard homogeneous M/M/2 model as a lower bound. A similar heterogeneous M/M/2 model would look like Figure 5-19. This is not a standard M/M/2 system

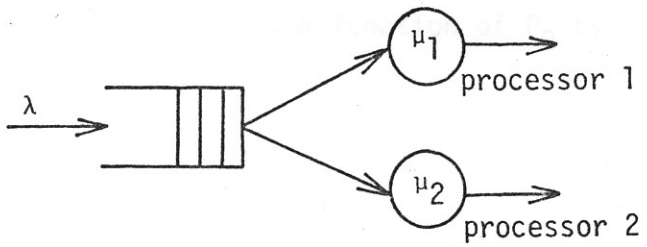


Figure 5-19: Heterogeneous M/M/2 Model.

since an arriving job can choose either processor 1 or processor 2 when the queue is empty. In order to minimize the average turnaround time, we make the assumption that an arrival always selects the fastest processor if the system is empty. If $\mu_1 > \mu_2$ we have the state-transition-rate diagram of Figure 5-20. P_i represents the equilibrium probability

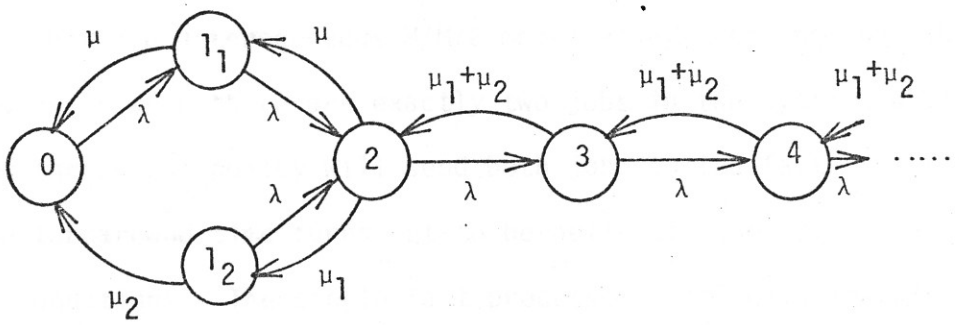


Figure 5-20: State-Transition-Rate Diagram for the Heterogeneous M/M/2 System.

that i jobs are in the queue. P_{1_j} denotes the joint probability that there is exactly one job in the system and it is being served by

processor j. It follows from Figure 5-20 that P_i can be expressed as

$$P_i = \left(\frac{\lambda}{\mu_1+\mu_2}\right)^{i-2} P_2,$$

where P_2 can be obtained as a function of P_0 by solving the boundary equations

$$\lambda P_0 = \mu_1 P_{1_1} + \mu_2 P_{1_2}$$

$$(\lambda+\mu_1)P_{1_1} = \mu_2 P_2$$

$$(\lambda+\mu_2)P_{1_2} = \lambda P_0 + \mu_1 P_2 .$$

Using the conservation relation $\sum_{\text{all } i} P_i = 1$ and the above equations, all P_i can be found. Our analysis shows, however, that the turnaround time of this model is not a lower bound on the other models. For example, under very light load conditions with $\mu_1 >> \mu_2$, the maximum throughput policy performs better than this heterogeneous M/M/2 model. This is due to the fact that the heterogeneous M/M/2 model always assigns one job to each processor when there are exactly two jobs in the system, while the maximum throughput policy will send both jobs to the fast processor. The average turnaround time turns out to be better in the latter case under some conditions. The single fast processor model will therefore be used to compute a lower bound on the turnaround time for comparison with the heterogeneous systems.

5.5 Comparison of Performance

The average job turnaround times (TT) of the proposed two-processor heterogeneous models are compared in this section. We label these models as follows:

Model A: state independent routing with proportional branching probabilities,

Model B: minimum system time policy,

Model C: maximum ratio policy,

Model D: maximum throughput policy,

Model E: single fast processor.

In section 5.1.1 we showed that the average turnaround time of Model A can be expressed in closed form as $2/(\mu_1+\mu_2-\lambda)$. Models B, C, and D are solved by using the recursive solution technique. The computed state equilibrium probabilities (P_{ij}'s) are then used to evaluate the average job turnaround time. Model E serves as a lower bound on each model. The model is a standard M/M/1, and the average turnaround time is given by

$$TT = \frac{1}{\mu_1+\mu_2-\lambda} .$$

It is interesting to compare the effect of the different state dependent job routing strategies on the state-transition-rate diagram. Figure 5-21 compares the policy lines of Models B, C, and D when $\mu_1 = 4$, $\mu_2 = 1$, and $\lambda = 2.5$. The policy lines of Models B and C are functions of μ_1 and μ_2 only, while that of Model D also depends on λ. Each policy line favors processor 1 since $\mu_1 > \mu_2$ and the job dispatcher naturally

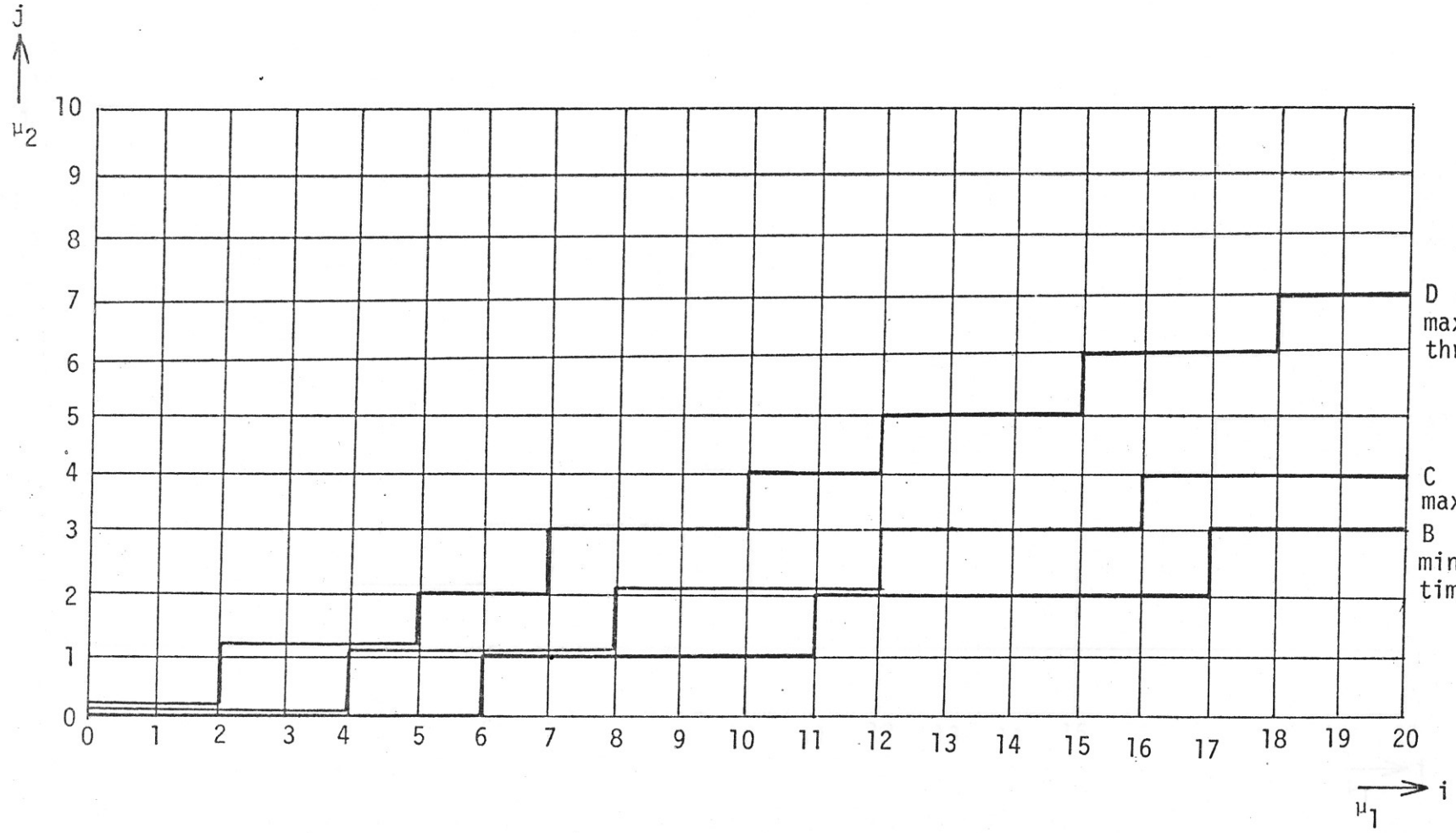


Figure 5-21: Comparison of Policy Lines for Models B, C and D when $\mu_1 = 4$, $\mu_2 = 1$, $\lambda = 2.5$.

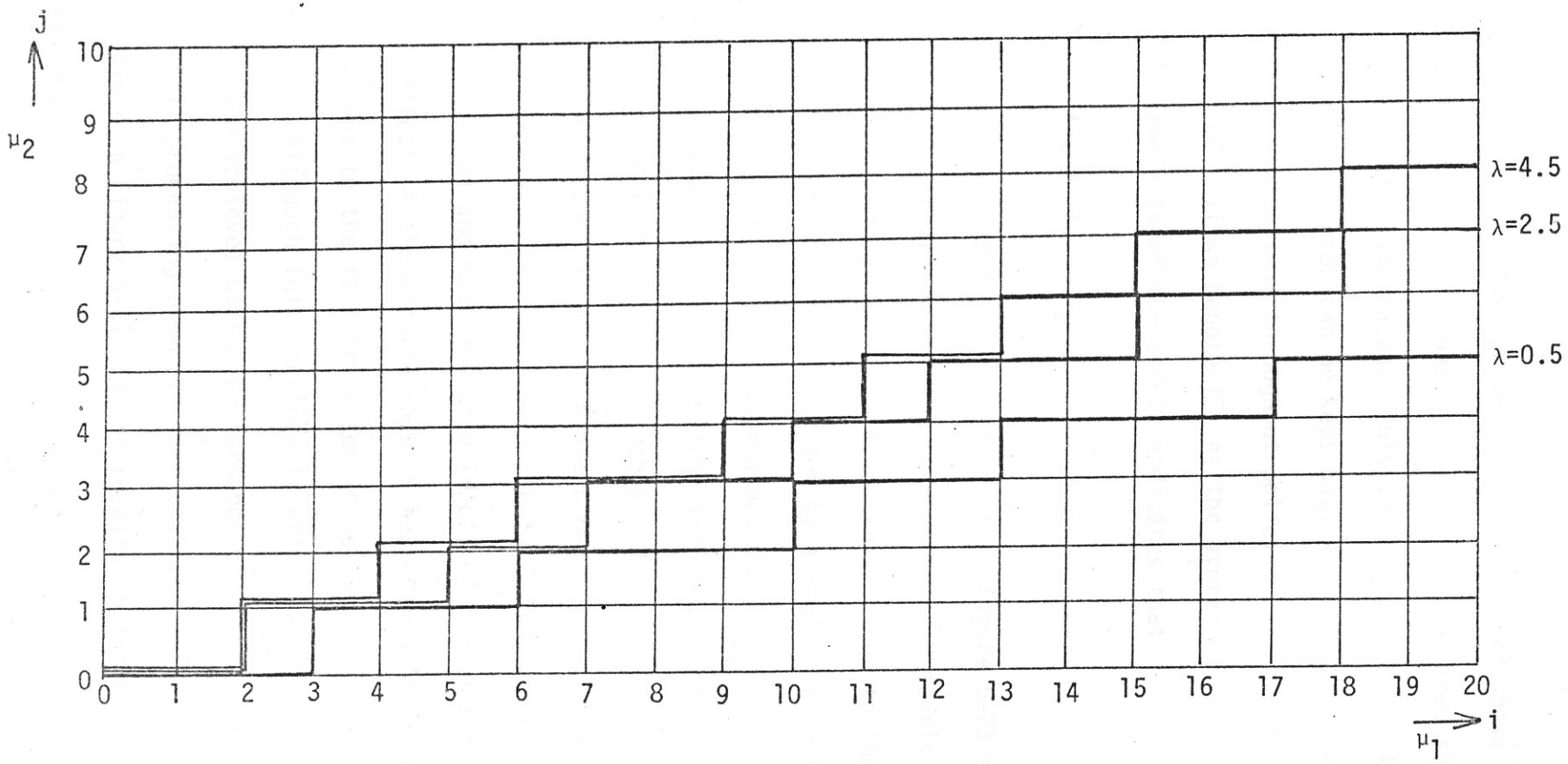


Figure 5-22: Policy Lines of Model D for λ = 0.5, 2.5, 4.5.

schedules more jobs on the fast processor. Figure 5-22 shows the policy lines of Model D for several values of λ. The policy lines shift slightly towards processor 2 as λ increases. This indicates that for larger arrival rates both processor can be kept busy most of the time.

As expected, the maximum throughput policy achieves the best average job turnaround time. If we denote TT_k as the average turnaround time of model K, the analysis of the models indicates that

$$TT_A \geq TT_B \geq TT_C \geq TT_D \geq TT_E ,$$

for each choice of λ. Table 5-1 and its plot in Figure 5-23 shows the results of the same comparison. Three data points for models B,C, and D are not shown in the table when $\lambda = 4.5$. This is due to the difficulty of accurately approximating an open system by a small finite state diagram when the $\lambda/(\mu_1 + \mu_2)$ ratio approaches one.

Several conclusions can be drawn from the comparison. Under light load conditions, models B, C, and D closely approximate the lower bound Model E. The performance of Models B and C degrade more than that of Model D as λ increases. This is due to the fact that the minimum system time (Model B) and maximum ratio (Model C) policies fail to consider the effect of the arrival rate. These two policies tend to schedule more jobs to the fast processor while leaving the slower processor idle. Although this strategy is effective for low arrival rates, the system achieves better performance under heavy load conditions when the overall processing power is more utilized. The comparison of the policy lines in Figure 5-21 and the results in Table 5-1 explain this phenomenon.

Model	Description	λ	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5
A	Proportional routing	TT_A	.444	.500	.571	.667	.800	1.000	1.333	2.000	4.000
B	minimum system time policy	TT_B	.286	.333	.399	.491	.622	.806	1.070	1.477	--
C	maximum ratio policy	TT_C	.286	.332	.390	.464	.559	.687	.875	1.175	--
D	maximum throughput policy	TT_D	.285	.330	.385	.446	.522	.627	.786	1.050	--
E	single fast processor	TT_E	.222	.250	.286	.333	.400	.500	.667	1.000	2.000

Table 5-1: Comparison of TT's for Models A, B, C, D, E.

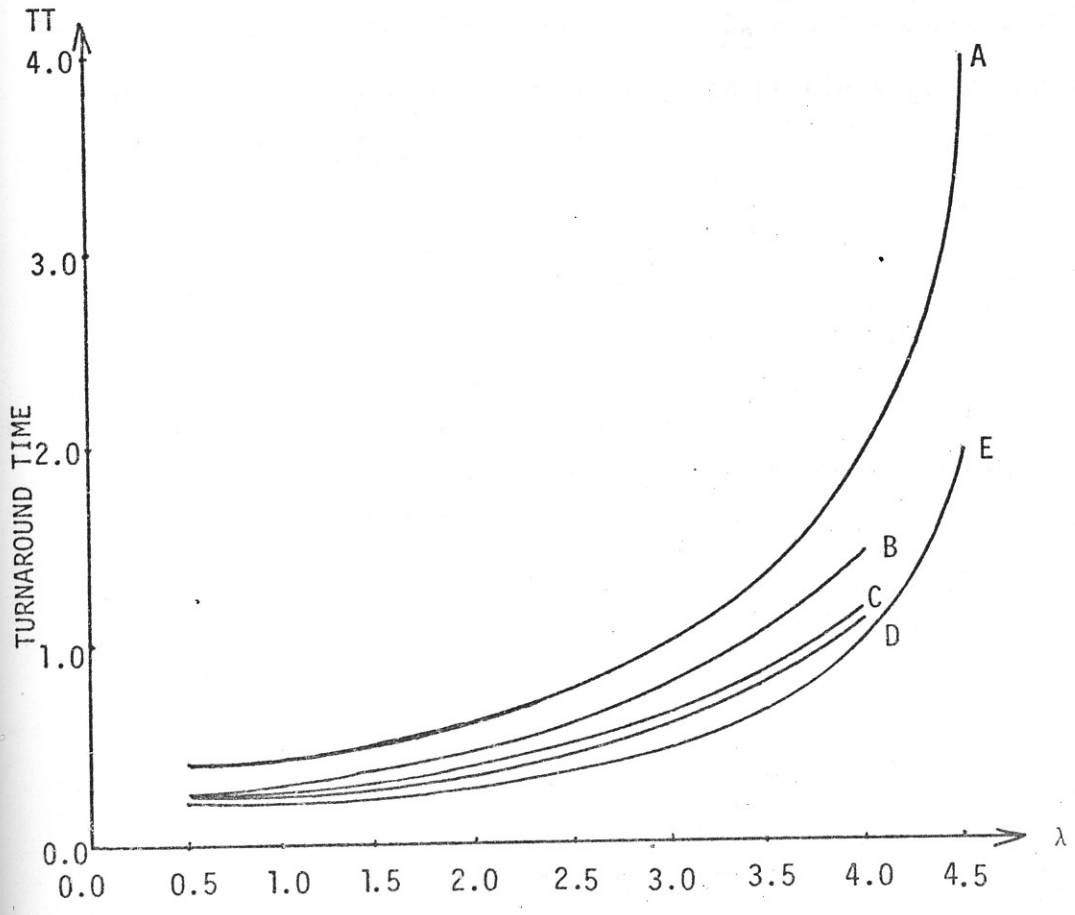


Figure 5-23: Comparison of TT's for Models A,B,C,D,E.

Model D gives the best performance and is extremely close to Model E, since the maximum throughput policy is adaptive to all system information available to the job dispatcher. The performance of Model C is superior to Model B since it attempts to minimizing the individual job turnaround time while the routing strategy in Model B (minimum system time policy) is not directly related to the minimization of the average job turnaround time.

We have shown that state dependent job routing policies can be used to improve the system performance significantly. Among all the job diapatching policies analyzed, the maximum throughput policy is the most attractive one. We conjecture that it is the optimal strategy for the job dispatcher given that the available information consists of the system parameters λ, μ_1, μ_2 and the current state (i,j) of the system. One of our future research problems is to formally verify this conjecture.

CHAPTER VI

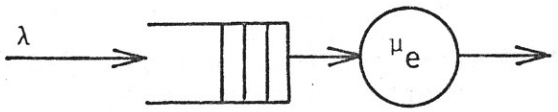
RELIABILITY RELATED PERFORMANCE EVALUATION

One of the most important design considerations of a distributed computer system is to provide high reliability. The reliability of computer systems has long been an active research area [BOUR 71, MATH 71, RENN 73, SPRA 74]. The research is usually concentrated on the direct modeling of the reliability of systems and components. In this chapter we present a different view of system reliability. The study is motivated by the recent work of Sauer and Chandy [SAUE 76] in which the problem of performance evaluation of systems that may fail is addressed. We are interested in the performance measures of distributed computer systems. Usually these measures are the average throughput rate (overall system criteria) and the average job turnaround time (overall user criteria). We will investigate how the reliability of system components effects overall system performance by embedding the question of reliability into other performance measures.

Consider a repairable computer system. Either a hardware or software system failure may occur during the execution of a job. The repair of the system is immediately begun after each failure and the terminated job is restarted after the repair is finished. Under this assumption, we will show that the effect of failures and repairs on system performance can be taken into account by defining a reduced equivalent service rate. We call this approach reliability related performance evaluation, and in the following sections we will use it to analyze the average job turnaround time and the average system throughput of an open and a closed system.

6.1 Average Job Turnaround Time of An Open Model

In section 5.1.2 we demonstrated that a job routing policy can create a new job arrival time distribution and how this distribution can be obtained by using the Coxian staging method. The same approach will be used to derive an equivalent service time distribution that takes system failures and repairs into consideration. Figure 6-1



$\mu_e \triangleq$ equivalent service rate of processor with failures and repair

Figure 6-1: Single Server Model with Failures.

shows a simple single server model with possible system failures and repair. The arrival, service, failure, and repair inter-event time distributions are assumed exponential with mean rates λ, μ, α, γ respectively. In particular,

$$\text{arrival time distribution } a(t) = \lambda e^{-\lambda t},$$

$$\text{service time distribution } f(t) = \mu e^{-\mu t},$$

$$\text{failure time distribution } g(t) = \alpha e^{-\alpha t},$$

$$\text{and repair time distribution } h(t) = \gamma e^{-\gamma t}.$$

Taking the possibility of failure and repair into account, the equivalent service time of a job can be expressed by the Coxian staging diagram in Figure 6-2. Each circle denotes either a service or repair stage with mean time $1/\mu$ and $1/\gamma$ respectively. P_f denotes the probability that a failure occurs before the completion of a job. $P_s = 1 - P_f$ is the

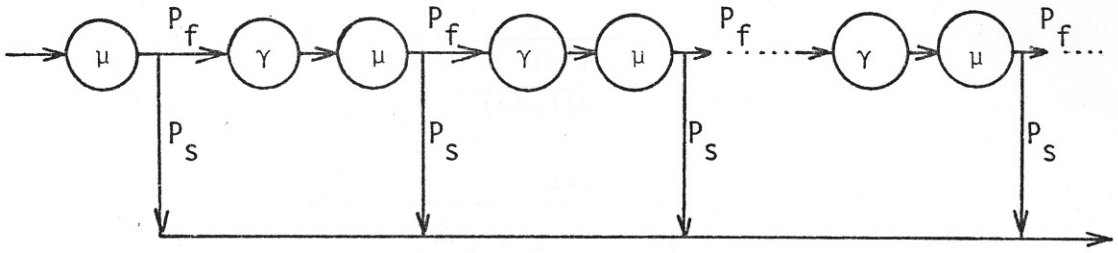


Figure 6-2: Coxian Staging Diagram of the Equivalent Service Distribution.

probability that a job is completed safely during its service time interval. The probability that a job passes through the system with n failures (and n restarts) is $P_f^n \times P_s$. P_s can be evaluated as follows:

$$P_s = \Pr[\text{a job is safely completed}]$$

$$= \int_0^\infty \Pr[\text{service time} = t] \cdot \Pr[\text{no failure during time } t]\, dt$$

$$= \int_0^\infty \mu e^{-\mu t} \cdot \left[1 - \int_0^t \alpha e^{-\alpha\tau}\, d\tau\right] dt$$

$$= \frac{\mu}{\alpha+\mu}.$$

P_f is then given by

$$P_f = 1 - P_s = \frac{\mu}{\alpha+\mu}.$$

With P_s and P_f known, the s-transform $B^*(s)$ of the equivalent service distribution represented by the Coxian staging diagram can be expressed as

$$B^*(s) = P_s \frac{\mu}{s+\mu} + P_f P_s \left(\frac{\mu}{s+\mu}\right)^2\left(\frac{\gamma}{s+\gamma}\right) + P_f^2 P_s \left(\frac{\mu}{s+\mu}\right)^3\left(\frac{\gamma}{s+\gamma}\right)^2 + \ldots$$

$$= P_s \frac{\mu}{s+\mu} \frac{1}{1 - \frac{\gamma\mu P_f}{(s+\gamma)(s+\mu)}}$$

$$= \frac{\mu^2}{\alpha+\mu} \frac{s+\gamma}{s^2 + (\mu+\gamma)s + \frac{\gamma\mu^2}{\alpha+\mu}} .$$

Since $B^*(s)$ can be expended and expressed as the sum of two first order terms, i.e., $B^*(s) = \frac{\mu^2}{\alpha+\mu}\left(\frac{C}{s+A} + \frac{D}{s+B}\right)$,

the inverse transform of $B^*(s)$ is a hyperexponential distribution with two parallel exponential stages (H_2). The equivalent mean service rate μ_e of the new service distribution is given by

$$\mu_e = -\frac{1}{\left.\frac{\partial B^*(s)}{\partial s}\right|_{s=0}}$$

$$= \frac{\gamma\mu^2}{\mu\alpha+\gamma\alpha+\gamma\mu} . \qquad (6\text{-}1)$$

Two limiting cases of $B^*(s)$ are of interest.

Case 1). Perfect server: $\alpha = 0$.

$$B^*(s) = \frac{\mu^2}{\mu} \frac{s+\gamma}{(s+\mu)(s+\gamma)} = \frac{\mu}{s+\mu}.$$

The service time distribution is exponential with mean service time $1/\mu$.

Case 2). Immediate repair: $\gamma = \infty$.

$$B^*(s) = \frac{\mu^2}{\alpha+\mu} \frac{s/\gamma + 1}{s^2/\gamma + \mu s/\gamma + s + \frac{\mu^2}{\alpha+\mu}} = \frac{\mu^2}{\alpha+\mu} / \left(s + \frac{\mu^2}{\alpha+\mu}\right).$$

The service time distribution is exponential with mean

service time $(\alpha+\mu)/\mu^2$.

We have shown that the equivalent service time distribution of a system with possible failures can be derived by using the Coxian staging method. The system becomes an $M/H_2/1$ model, where H_2 represent the hyperexponential service distribution with two parallel exponential stages. The new service distribution has a lower mean service rate and a higher coefficient of variation. The average job turnaround time can be analyzed using the Pollaczek-Khinchin formula [KLEI 75] and expressed as a function of the new mean service rate and coefficient of variation.

An interesting question to ask is : How does the degradation of the average turnaround time in an m-processor system with failures and repair compare with that of a single processor system of equaivalent total processing rate? Figure 6-3 shows such an m-processor system with failures and repair. Assume that the processors in both systems have the same failure and repair distributions and, furthermore,

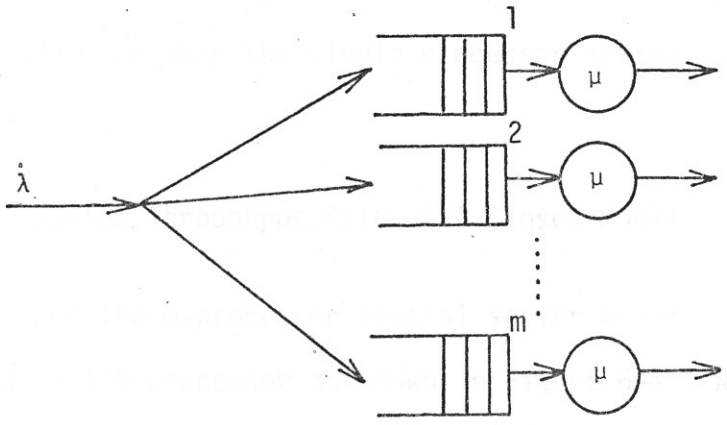


Figure 6-3: A m-processor System with Failures and Repair.

simultaneous repairs are allowed in the multiple processor system. These assumptions are pessimistic and may be unjustified; however, it provides an interesting comparison. Under these assumptions, the m-processor system degrades gracefully under failures and the probability of total system failure is much less than that of the single processor system as defined later in section 6.2. The equivalent service rate μ_s of the single fast processor can be obtained by substituting $m\mu$ for μ in equation (6-1) and is given by

$$\mu_s = \frac{\gamma(m\mu)^2}{m\mu\alpha + \gamma\alpha + m\mu\gamma} .$$

The equivalent service rate μ_m of the m-processor composite system is m times the equivalent processing rate of each individual processor, and can be expressed as

$$\mu_m = m \frac{\gamma\mu^2}{\mu\alpha + \gamma\alpha + \gamma\mu} .$$

It can be seen that $\mu_s > \mu_m$ for $m > 1$. This indicates that although the multiple processor system has higher availability, its performance will degrade more than the single processor system as the failure rate increases.

6.2 System Throughput Rate of A Closed Model

Consider the m-processor central server model consisting of m CPU's and an I/O processor as shown in Figure 6-4. We are interested

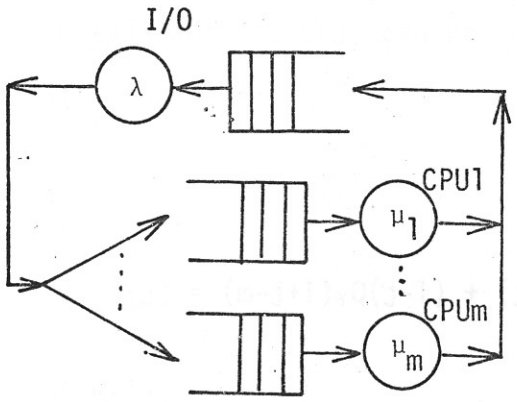


Figure 6-4: m-Processor Central Server Model.

in finding the average throughput of the system when the m CPU's are allowed to fail. We can define the average system throughput TP as

$$TP = \sum_{j=1}^{m} TP(j) \times Q(j),$$

where TP(j) is the average throughput given that exactly j processors are available and Q(j) is the probability that j processors are available. The TP(j) for a closed multiple processor system can be computed as discussed in Chapter IV and Q(j) can be derived from the state diagram in Figure 6-5. State j indicates that j CPU's are currently available,

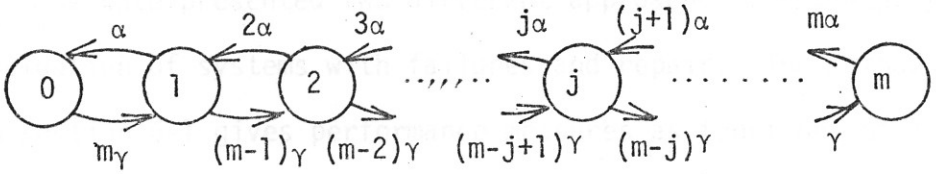


Figure 6-5: State Diagram for Failure and Repair.

and α and γ are the mean failure and repair rates respectively. The state diagram assumes that failed CPU's are simultaneously repaired.

The steady state availability Q(j) can be obtained by solving the equilibrium state equations:

$$m\gamma Q(0) = \alpha Q(1),$$

$$(j\alpha + (m-j)\gamma)Q(j) = (m-j+1)\gamma Q(j-1) + (j+1)\alpha Q(j+1), \quad 0 < j < m,$$

$$m\alpha Q(m) = \gamma Q(m-1),$$

where $\sum_{j=0}^{m} Q(j) = 1$. Q(j) has a closed form expression of

$$Q(j) = \frac{m!}{j!(m-j)!} \left(\frac{\gamma}{\alpha}\right)^j Q(0)$$

The average throughput of a single processor system with failures and repair can be found by setting m = 1 in equation (6-2).

We also can analyze these system by finding equivalent CPU service distributions as illustrated in the previous section. However, this approach gives us m H_2 servers in the queueing network with turns out to be tedious to analyze.

6.3 Discussion of the Approaches

We have presented two different approaches for the performance evaluation of systems with failures and repair. The method discussed in section 6-1 gives performance measures as functions of the equivalent service distributions which in turn are functions of the failure and repair distributions. The second approach in section 6-2 provides the performance measures as functions of the system availability, which dependes on the failure and repair rates. There is a basic difference between the two models. The former method may schedule a job on a

processor even if the processor has already failed, since the failure of the processor is invisible to the user job, while the latter method computes the throughput using only the available processors. If we are to find the equivalent service distribution of each processor in an m-processor system, the assumption of simultaneous repairs makes the analysis tractable. In the second approach the availability Q(j) can be computed as a function of the number of repairers. The assumption of the number of repairers is crucial in the performance analysis of multiple processor systems.

We have shown that the consideration of system failures and repairs can be embedded into other performance measures. The particular example in section 6.1 indicates that we may obtain an equivalent service distribution with lower mean service rate and higher coefficient of variation. An interesting study by Sauer and Chandy [SAUE 76] compares the system throughputs of a dual processor and a single processor system as a function of the coefficients of variation of the processors. They computed the system throughputs of closed central server models with single and dual CPU's for various coefficients of variation of the CPU service distribution. Their results indicate that the dual processor system achieves better system throughput than the single processor system when the coefficients of variation of the CPU's are high. This result is useful for our comparison of the performance of single and multiple processor systems with failure assumptions. Since the failure and repair processes will increase the coefficient of variation of the server, the results of the comparison of throughput will favor the multiple processor system as the failure rate increases. This conclusion seems

to contradict our analysis of the open model in section 6.1; however, it is true under certain conditions. In the central server model, the overall system throughput is directly proportional to the utilization of the I/O. Since the system is closed, there are a finite number of jobs in the system. If the coefficient of variation is high, the long jobs will tie up the CPU in the single processor system and therefore idle the I/O. In the multiple processor case, the short jobs have a better chance of being serviced when some CPU's are busy serving long jobs. This increases the utilization of the I/O. Their results also indicate that the advantage of the multiple processor system will disappear when the degree of multiprogramming is high and the I/O to CPU speed ratio is extremely low and when the I/O to CPU speed ratio is high. Under the first condition, the I/O will become a bottleneck. The utilization of the I/O then becomes less sensitive to the variation of job departures from the CPU's. In the case of high I/O to CPU speed ratio, the I/O will be mostly idle and again the utilization of the I/O is insensitive to the coefficient of variation of the CPU's. This reasoning explains the results shown by Sauer and Chandy. In general, for closed systems the average throughput of a multiple processor model can be greater than that of the single processor model if the service distributions of the CPU's have high coefficients of variation.

In multiple processor systems with failures and repairs, a state dependent job routing strategy would play an important role in improving the system performance. A job routing policy that takes failures and repairs into account would be adaptive to changes in the system. Our brief study in this chapter represents an initial investigation of the

relationship between system reliability and performance. It is expected to be a fruitful research area.

CHAPTER VII

CONCLUSIONS

The main contributions of this research were in three different areas:

1. the presentation of a more general approach to the problem of load balancing in a distributed computer system;
2. the development and analysis of queueing models for distributed computer systems; and
3. the generalization of some solution techniques for efficient analysis of the proposed models.

First, we presented the concept of dynamic load balancing in distributed computer systems. State dependent job routing policies for homogeneous and heterogeneous systems were introduced and we demonstrated that system performance can be improved significantly by balancing the workload at the job arrival stage and/or through communication channels.

Queueing models with features that reflect desireable properties of real distributed computer systems were proposed. These models were evaluated on the basis of overall system throughput rate and average job turnaround time. The conclusions drawn from the comparisons can be fed back to influence the implementation of such systems. We have shown that many existing analytical techniques can be simplified or generalized for the analysis of multiple processor computer systems. In the analysis of large homogeneous systems we were only interested in the overall system performance. Since all processors are identical

the state definition could be simplified and the complexity of the evaluation of steady state probabilities and their normalizing constant drastically reduced. We also illustrated that the Coxian staging method is useful in obtaining the job arrival and service distributions when the effects of job routing policies and system reliability are taken into consideration. The recursive solution technique was generalized for the analysis of two-processor distributed systems with job routing policies that satisfy certain properties.

Some state dependent job routing policies were suggested and formally defined. Among these, the maximum throughput policy is of particular interest. We conjecture that this job routing strategy is optimal under the condition that the system parameters and current state are the only information available to the job dispatcher. Our comparisons show that the maximum throughput policy does achieve the best overall system performance among those policies studied.

Directions for future research can be divided into two categories: short term and the long term. Topics of immediate short term interest include the proof of the conjecture that the maximum throughput policy is optimal and the generalization of the recursive solution method for systems with more than two processors. More ambitious long term research projects should focus on extending the work to larger multiple-processor systems, reliability related performance evaluation, and the development of approximation methods for distributed computer systems. The models presented in the dissertation consist of only simple distributed systems. In a larger network, load balancing strategies for

several stages of queues may be desirable. The topic of reliability related performance evaluation is an important new research area. Difficulties with the exact analysis of complex queueing networks lead us to consider the potential of approximation methods. A good approximation method can facilitate the efficient evaluation of the system and may also provide more direct insight into the system behavior.

APPENDIX A

Solution steps for the sample system in Figure 4-23.

1. Set $P_{44} = 1$.

2. Set $P_{33} = X$.

3. Use the GBE for state (4,4) to determine P_{43}.

$$\mu P_{44} = 2\lambda P_{43}$$

$$P_{43} = \frac{\mu}{2\lambda}$$

4. Use the GBE for state (4,3) to determine P_{42}.

$$(2\lambda + 2\mu)P_{43} = \mu P_{44} + \lambda P_{33} + 2\lambda P_{42}$$

$$P_{42} = \frac{1}{2\lambda}\left[(2\lambda + 2\mu)\ \frac{\mu}{2\lambda} - \mu - \lambda x\right]$$

$$= \frac{1}{2\lambda}\left[\frac{\mu^2}{\lambda} - \lambda x\right]$$

5. Use the GBE for state (4,2) to determine P_{41}.

$$(2\lambda + 2\mu)P_{42} = \mu P_{43} + 2\lambda P_{41}$$

$$P_{41} = \frac{1}{2\lambda}\left[(2\lambda + 2\mu)\ \frac{1}{2\lambda}\left(\frac{\mu^2}{\lambda} - \lambda x\right) - \frac{\mu^2}{2\lambda}\right]$$

$$= \frac{1}{4\lambda^2}\left[(2\lambda + 2\mu)\left(\frac{\mu^2}{\lambda} - \lambda x\right) - \mu^2\right]$$

$$= \frac{1}{4\lambda^2}\left[\mu^2\left(1 + \frac{2\mu}{\lambda}\right) - \lambda(2\lambda + 2\mu)x\right]$$

6. Use the GBE for state (4,1) to determine P_{40}.

$$(2\lambda + 2\mu)P_{41} = \mu P_{42} + 2\lambda P_{40}$$

$$P_{40} = \frac{1}{2\lambda}\left[(2\lambda+2\mu)\ \frac{1}{4\lambda^2}\left[(2\lambda+2\mu)\left(\frac{\mu^2}{\lambda} - \lambda x\right) - \mu^2\right] - \frac{\mu}{2\lambda}\left(\frac{\mu^2}{2\lambda} - \lambda x\right)\right]$$

$$= \frac{1}{8\lambda^3}[(2\lambda+2\mu)^2(\frac{\mu^2}{\lambda} - \lambda x) - \mu^2(2\lambda+2\mu) - 2\lambda\mu(\frac{\mu^2}{\lambda} - \lambda x)]$$

$$= \frac{1}{8\lambda^3}[(2\lambda+2\mu)^2 \frac{\mu^2}{\lambda} - (2\lambda+2\mu)\mu^2 - 2\mu^3 - (2\lambda+2\mu)^2 \lambda x + 2\lambda^2\mu x].$$

7. Use the GBE for state (4,0) to solve for X.

$$(2\lambda + \mu)P_{40} = \mu P_{41}$$

$$\frac{(2\lambda+\mu)}{8\lambda^3}[(2\lambda+2\mu)^2 \frac{\mu^2}{\lambda} - (2\lambda+2\mu)\mu^2 - 2\mu^3 - (2\lambda+2\mu)^2 \lambda x + 2\lambda^2 \mu x]$$

$$= \frac{\mu}{4\lambda^2}[\mu^2(1 + \frac{2\mu}{\lambda}) - \lambda(2\lambda + 2\mu)x]$$

$$X = \frac{(2\lambda+\mu)[(2\lambda+2\mu)^2 \frac{\mu^2}{\lambda} - (2\lambda+2\mu)\mu^2 - 2\mu^3] - 2\mu^3(\lambda+2\mu)}{(2\lambda+\mu)[\lambda(2\lambda+2\mu)^2 - 2\lambda^2\mu] - 2\lambda^2\mu(2\lambda+2\mu)}$$

8. Substitute the value of X into equations in Steps 2, 4, 5, 6 to obtain numerical values for P_{33}, P_{42}, P_{41}, and P_{40}.

9. Since P_{33} and the state equilibrium probabilities in column 4 are now known, the same process may be repeated for column 3 to obtain P_{22} and all P_{3j}'s. This is repeated in turn for columns 2 and 1.

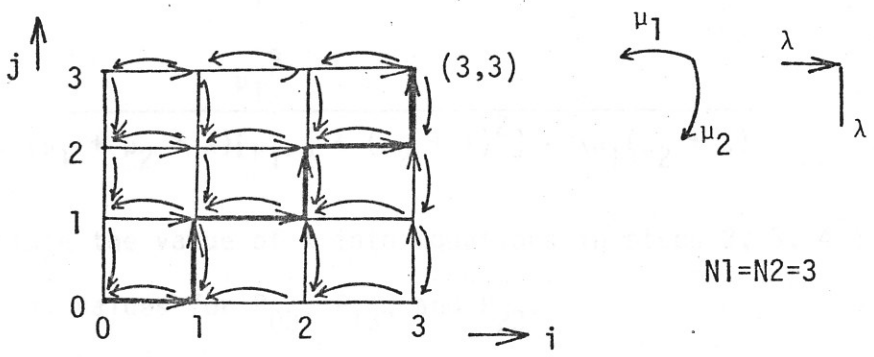
10. Use the relation $P_{ji} = P_{ij}$ to find the state probabilities for the other half of the state diagram.

11. Normalize all P_{ij} by setting

$$P_{ij} \longleftarrow \frac{P_{ij}}{\sum_{k=0}^{N} \sum_{l=0}^{N} P_{kl}}$$

A P P E N D I X B

Solution steps for a sample example using the algorithm in section 5.3.



1. Set $P_{33} = 1$
2. Set $P_{03} = X$
3. Use the GBE for state (0,3) to determine P_{13}

$$(\mu_2 + \lambda)P_{03} = \mu_1 P_{13}$$

$$P_{13} = \frac{1}{\mu_1}(\mu_2 + \lambda)X$$

4. Use the GBE for state (1,3) to determine P_{23}

$$(\mu_1 + \mu_2 + \lambda)P_{13} = \lambda P_{03} + \mu_1 P_{23}$$

$$P_{23} = \frac{1}{\mu_1}\left[(\mu_1 + \mu_2 + \lambda)\frac{\mu_2+\lambda}{\mu_1}X - \lambda X\right]$$

$$= \frac{\mu_1\mu_2 + (\mu_2+\lambda)^2}{\mu_1^2}X$$

5. Use the GBE for state (2,3) to solve for X

$$(\mu_1 + \mu_2 + \lambda)\, P_{23} = \lambda P_{13} + \mu_1\, P_{33}$$

$$\frac{(\mu_1 + \mu_2 + \lambda)[\mu_1\mu_2 + (\mu_2 + \lambda)^2]}{\mu_1^2} X = \frac{\lambda}{\mu_1} (\mu_2 + \lambda)\, X + \mu_1$$

$$X = \frac{\mu_1^3}{(\mu_1 + \mu_2 + \lambda)[\mu_1\mu_2 + (\mu_2 + \lambda)^2] - \lambda\mu_1(\mu_2 + \lambda)}$$

6. Substitute the value of X into equations in steps 2, 3, 4 to obtain numerical values for P_{03}, P_{13}, and P_{23}.

7. Use the GBE for state (3,3) to determine P_{32}

$$(\mu_1 + \mu_2)\, P_{33} = \lambda P_{23} + \lambda\, P_{32}$$

$$P_{32} = \frac{1}{\lambda} [\mu_1 + \mu_2 - \lambda\, P_{23}]$$

8. P_{32} is now known since we have the numerical value for P_{23}.

9. Set $P_{30} = X$

10. Use the GBE for state (3,0) to determine P_{31}

$$(\mu_1 + \lambda)\, P_{30} = \mu_2\, P_{31}$$

$$P_{31} = \frac{1}{\mu_2} (\mu_1 + \lambda)\, X$$

11. Use the GBE of state (3,1) to solve for X

$$(\mu_1 + \mu_2 + \lambda)\, P_{31} = \lambda P_{30} + \mu_2\, P_{32}$$

$$(\mu_1 + \mu_2 + \lambda) \frac{\mu_1 + \lambda}{\mu_2} X = \lambda x + \mu_2\, P_{32}$$

$$X = \frac{2^2 P_{32}}{(\mu_1 + \mu_2 + \lambda)(\mu_1 + \lambda) - \lambda\mu_2}$$

$$= \frac{\mu_2^2 P_{32}}{\mu_1\mu_2 + (\mu_1 + \lambda)^2}$$

12. Substitute the value of X into equations in steps 9, 10, to obtain numerical values for P_{30} and P_{31}.

13. Use the GBE for state (3,2) to determine P_{22}

$$(\mu_1 + \mu_2 + \lambda)\, P_{32} = \lambda\, P_{22} + \mu_2\, P_{33} + \lambda\, P_{31}$$

$$P_{22} = \frac{1}{\lambda}\left[(\mu_1 + \mu_2 + \lambda)\, P_{32} - \mu_2\, P_{33} - \lambda\, P_{31}\right]$$

14. We now have the numerical value for P_{22} since P_{33}, P_{32}, and P_{31} are known.

15. We have reduced the 4 x 4 state diagram to 3 x 3 with P_{22} known. The same process may be repeated to solve all other state probabilities.

16. Normalizing all P_{ij} by setting

$$P_{ij} \longleftarrow \frac{P_{ij}}{\sum_{k=0}^{3} \sum_{\ell=0}^{3} P_{k\ell}} .$$

BIBLIOGRAPHY

[ADIR 72] Adiri, I., "Queueing Models for Multiprogrammed Computers," Proceedings of the International Symposium on Computer Communication Networks and Teletraffic, Polytech Press, Brooklyn, N.Y.,1972, pp. 441-448.

[ANDE 75] Anderson, G.A., and E.D.Jensen, "Computer Interconnection Structures: Taxonomy, Characteristics, and Examples," Computing Surveys, vol.7, No.4, December 1975, pp. 197-213.

[BABA 75] Babad, J.M., "A Generalized Multi-Entrance Time-Sharing Priority Queue," J.ACM, Vol. 22, No. 2, April 1975, pp. 231-247.

[BAER 73] Baer, J.L., "A Survey of Some Theoretical Aspects of Multiprocessing," Computing Surveys, Vol.5, No.1, March 1973, pp. 31-80.

[BARN 68] Barnes, G.H., R.M. Brown, M. Kato, D. Kuck, D. Slotnick, R. Stokes, "The ILLIAC IV Computer," IEEE Trans. Computers, Vol. C-17, No. 8, August 1968, pp. 746-757.

[BASK 75] Baskett, F., K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," J.ACM, Vol. 22, No. 2, April 1975 pp. 248-260.

[BOUR 71] Bouricius, W.G. and et al., "Reliability Modeling for Fsult-Tolerant Computers," IEEE Trans. on Computers, Vol. C-20, No. 11, November 1971, pp. 1306-1311.

[BOWD 73] Bowdon, E.K.,Sr., S.A. Mamrak, and F.R. Salz, "Simulation - A Tool for Performance Evaluation in Network Computers," Proc. AFIPS 1973 Natl. Comp. Conf., Vol. 42, pp. 121-131.

[BUZE 73] Buzen, J.P., "Computational Algorithms for Closed Queueing Networks with Exponential Server," C.ACM, Vol. 16, No. 9, September 1973, pp. 527-531.

[BUZE 76] Buzen, J., "Operational Analysis : The Key to the New Generation of Performance Prediction Tools," COMPCON 76 Fall, Washington D.C., September 1976, pp. 166-171.

[CHAN 72] Chandy, K.M., "The Analysis and Solutions for General Queueing Networks," Proc. Sixth Annual Princeton Conf., 1972, pp. 224-228.

[CHAN 75A] Chandy, K.M., U. Herzog, and L. Woo, "Parametric Analysis of Queueing Networks," I.B.M. J. Res. Develop., Vol. 19, No. 1, January 1975, pp. 36-42.

[CHAN 75B] Chandy, K.M., U. Herzog, and L. Woo, "Approximate Analysis of General Queueing Networks," I.B.M. J. Res. Develop., Vol. 19, No. 1, January 1975, pp. 43-49.

[CHOW 76] Chow, Y., and W.H. Kohler, "Analysis and Comparison of Several Queueing Models for Multiprocessor Multiprogramming Systems," Technical Report ECE-CS-76-2, Dept. of Electrical and Computer Engineering, University of Massachusetts, Amherst, July 1976.

[COFF 68] Coffman, E.G., Jr., and L. Kleinrock, "Feedback Queueing Models for Time-Sharing Systems," J.ACM, Vol. 15, No. 4, October 1968, pp. 549-576.

[COFF 73] Coffman, E.G., Jr., and P.J. Denning, Operating System Theory, Prentice Hall, Englewood Cliffs, N.J., 1973.

[COLE 73] Coleman, M.L., "ACCENT - A Corporate Computer Network," Proc. AFIPS 1973 Natl. Comp. Conf., Vol. 42, pp. 133-140.

[COSE 75] Cosell, B., P. Johnson, J. Malman, R. Schantz, J. Sussman, R. Thomas, and D. Walden, "A Non-trivial Example of Computer Resource Sharing," Submitted to 5th Symp. on O.S. Principles.

[COX 55] Cox, D.R., "A Use of Complex Probabilities in the Theory of Stochastic Processes," Proceedings Cambridge Philosophical Society, 1955, pp. 313-319.

[COX 61] Cox, D.R., and W.L. Smith, Queues, Methuen, London, 1961.

[ENSL 76] Enslow, P.H., "What Does 'Distributed Processing' Mean?" Brown Univ. Work Shop on Distributed Processing, August 1976.

[FARB 73] Farber, D.J., et al., "The Distributed Computer System," Proc. 7th Annual IEEE Computer Society International Conference, Feb. 1973, pp. 31-34.

[FELL 68] Feller, W., An Introduction to Probability Theory and Its Applications, Volume 1, John Wiley & Sons, Inc, N.Y. 1968.

[FLAT 76] Flatto, L., and H. McKean, "Two Parallel Queues with Equal Serving Rates," IBM Research, RC 5916, March 1976.

[FRAN 72] Frank, H., R.E. Kahn, and L. Kleinrock, "Computer Communication Network Design: Experience with Theory and Practice," Networks, Vol. 2, 1972, pp. 135-166.

[FULL 73] Fuller, S.H., and D.P. Siewiorek, "Some Observations on Semi-conductor Technology and the Architecture of Large Digital Modules," Computer, Vol. 6, No. 10, October 1973, pp. 15-21.

[FULL 75] Fuller, S.H., and F. Baskett, "An Analysis of Drum Storage Units," J.ACM, Vol. 22, No. 1, January 1975 pp. 83-105.

[GELE 75] Gelenbe, E., "On Approximate Computer System Models," J. ACM Vol. 22, No. 2, April 1975, pp. 261-269.

[GORD 67] Gordon, W.J., and G.F. Newell, "Closed Queueing systems with Exponential Servers," Operation Research, Vol. 15, 1967, pp. 254-265.

[HEAR 73] Heart, F.E., S.M. Ornstein, W.R. Crowther, and W.B. Barker, "A New minicomputer/multiprocessor for the ARPA NETWORK," Proc. AFIPS 1973 Natl. Comp. Conf., Vol. 42, pp. 529-537.

[HERZ 75] Herzog, U., L. Woo, and K.M. Chandy, "Solution of Queueing Problems by a Recursive Technique," IBM J. Res. Develop., Vol. 19, No. 3, May 1975, pp. 295-300.

[HEWL 74] Hewlett-Packard, "HP Distributed Systems Technical Data," Sales Literature, September 1974.

[HOFR 75] Hofri, M. and M. Yadin, "A Processor in Series with Demand Interrupting Device--A Stochastic Model," J.ACM, Vol. 22, No. 2, April 1975, pp. 270-290.

[JACK 57] Jackson, J.R., "Networks of Waiting Lines," Operations Research, Vol. 5, No. 4, July-August 1957, pp. 518-521.

[JACK 63] Jackson, J.R., "Jobshop-Like Queueing Systems," Management Science, Vol. 10, No. 1, October 1963, pp. 131-142.

[KAHN 72] Kahn, R., "Resource-Sharing Computer Communications Networks," Proc. of the IEEE, Vol. 60, No. 11, Nov. 1972 pp. 1397-1407.

[KING 62] Kingman, J.F.C, "On Queues in Heavy Traffic," Journal of the Royal Statistical Society, Series B, 24, 1962, pp. 383-392.

[KLEI 69] Kleinrock, L., "Models for Computer Networks," Proc. IEEE International Conference on Communications, 1969.

[KLEI 70A] Kleinrock, L., "A Continuum of Time-Sharing Scheduling Algorithms," Proc. AFIPS 1970 Spr. Joint Comp. Conf., Vol. 36, pp. 453-458.

[KLEI 70B] Kleinrock, L., "Analytic and Simulation Methods in Computer Network Design," Proc. AFIPS 1970 Spr. Joint. Comp. Conf., Vol. 36, pp. 569-579.

[KLEI 75] Kleinrock, L., Queueing Systems, Vol. I: Theory, Wiley Interscience, New York, 1975.

[KLEI 76] Kleinrock. L., Queueing Systems, Volume II: Computer Applications, Wiley Interscience, New York, 1976.

[KOBA 73] Kobayaski, H., "Applications of the Diffusion Approximation to Queueing Networks: Part II," Proc. Seventh Annual Princeton Conf., March 1973, pp. 448-454.

[KOEN 66] Koenigsberg, E., "On Jockeying in Queues," Management Science, Vol. 12, No. 5, January 1966, pp. 412-436.

[KOHL 75] Kohler, W.H., "Queueing Models for Distributed Computer Systems," Research Proposal to NSF, University of Massachusetts, September 1975.

[LAVE 75] Lavenberg, S.S., and D.R. Slutz, "Introduction to Regenerative Simulation," IBM Journal of Research and Development, Vol. 19, No. 5, September 1975, pp. 458-462.

[MANN 76] Manning, E.G. and Peebles, R.W., "A Homogeneous Network for Data Sharing - Communications," Computer Communications Network Group, Report E-12, University of Waterloo, July 1976.

[MATH 71] Mathur F.P., "On Reliability Modelling and Analysis of Ultra-reliable Fault-tolerant Digital Systems," IEEE Trans. Computers, Vol. C-20, No. 11, Nov. 1971.

[McCR 73] McCredie, John W., "Analytic Models as Aids in Multiprocessor Design," Proc. Seventh Annual Princeton Conf., March 1973, pp. 186-191.

[McDO 70] MacDougall, M.H., "Computer System Simulation: An Introduction," Computing Surveys, Vol. 2, No. 3, September 1970, pp. 191-209.

[McGR 75] McGregor, P.V., and R.R. Boorstyn, "Optimal Load Sharing in a Computer Network," Proc. 1975 International Conference on Communications, Vol. III, June 1975, pp. 41:14-19.

[McKI 69] McKinney, J.M., "A Survey of Analytical Time-Sharing Models," Computing Surveys, Vol. 1, No. 2, June 1969, pp. 105-116.

[MODU 75] Modular Computer Systems, Inc., "MAXNET - The Software Operating System for Distributed Computing Networks," Sales Literature, June 1975.

[MUNT 74] Muntz, R.R., and J.W. Wong, "Asymtotic Properties of Closed Queueing Network Models," 8th. Annual Princeton Conf. on Information Science and Systems, 1974, pp. 348-352.

[NAKA 71] Nakamura, G., "A Feedback Queueing Model for and Interactive System," FJCC, Vol. 39, 1971, pp. 57-64.

[REIS 75] Reiser, M., and H. Kobayaski, "Queueing Networks with Multiple Closed Chains: Theory and Computational Algorithms," I.B.M. J. Res. Develop., Vol. 19, No. 3, May 1975, pp. 283-294.

[RENN 73] Rennels, D.A. and A. Avizienis, "RMS: A Reliability Modeling System for Self-Repairing Computers," Dig. of Pap. 3rd IEEE Int. Symp. on Fault-Tolerant Comput., 1973, pp. 131-135.

[SAAT 61] Saaty, T.L., Elements of Queueing Theory, McGraw-Hill, New York, 1961.

[SAUE 75] Sauer, C.H., and K.M. Chandy, "Approximate Analysis of Central Server Models," IBM Journal of Research and Development, Vol. 19, No. 3, May 1975, pp. 301-313.

[SAUE 76] Sauer, C.H., and K.M. Chandy, "Parametric Modeling of Multi-miniprocessor Systems," IBM Research Report, RC-5978, May 1976.

[SENC 73] Sencer, M.A., and C.L. Sheng, "An Analysis of Multi-programmed Time-Sharing Computer Systems," Proc. AFIPS 1973 Natl. Comp. Conf., Vol. 42, pp. 87-91.

[SPRA 74] Spragins, J.D., "Reliability Models for Computer Communication Systems," Conf. Res., 7th Annual Asilomar Conf. on Circuits, Syst. and Comput., 1974, pp. 302-307.

[STAB 74] Stabler, G.M., "A Ststem for Interconnected Processing," Ph.D. Dissertation, Division of Applied Mathematics, Brown University, October 1974.

[STON 75] Stone, H.S., and W.H. Kohler, "Laboratory Equipment for Graduate Research in Computer Engineering," Equipment Proposal to NSF, March 1975.

[STON 77] Stone, H.S., "Multiprocessor Scheduling with the aid of Network Flow Algorithms," IEEE Transactions on Software Engineering, Vol. SE-3, No. 1, January 1977, pp. 85-93.

[TAKA 62] Takas, L., Introduction to Theory of Queues, Oxford Univ. Press, New York, 1962.

[THOM 73] Thomas, R.H., "A Resource Sharing Executive for the ARPANET," Proc. AFIPS 1973 Natl. Comp. Conf., Vol. 42 pp. 155-163.

[TOWS 75] Towsley, D.F., "Loacal Balance Models of Computer Systems," Ph.D. Dissertation, University of Texas at Austin, Dec. 1975.

[TOWS 77] Towsley, D.F., "Queueing Network Models with State Dependent Routing," Submitted to JACM, June 1977.

[VAND 74] Van Dam, A., "Computer Graphics and Its Applications," Final Report, NSF Grnat GT-28401x, Brown Univ., May 1974.

[WALL 66] Wallace, V.L., R.S. Rosenburg, "Markovian Models and Numerical Analysis of Computer System Behavior," SJCC, Vol. 28, 1966, pp. 141-148.

[WULF 72] Wulf, W.A., and C.G. Bell, "C.mmp - A Multi-mini processor," Proc. AFIPS 1972 Fall Joint Comp. Conf., Vol. 41, Part II, pp. 765-777.