COMPUTER SYSTEM MODELS WITH PASSIVE RESOURCES

by

Thomas Walter Keller

May 1976 TR-57

This report constituted the author's Doctoral Dissertation in Computer Sciences at The University of Texas at Austin and was supported in part by National Science Foundation Grants GJ-1084 and DCR74-13302.

DEPARTMENT OF COMPUTER SCIENCES
THE UNIVERSITY OF TEXAS AT AUSTIN

TABLE OF CONTENTS

CHAPTER PAGE

CHAPTER PAGE

I. Introduction

Performance modeling is playing an increasing role in shaping the development of modern computer systems. The growing complexity of these systems places their thorough understanding beyond the grasp of intuition. Those charged with designing complex systems to meet performance criteria or else reconfiguring existing systems to maximize performance must turn to more abstract system conceptualizations from which they can predict the essential performance properties of the system. Computer system modeling is evolving to meet this need for increasing generalization and abstraction. Queueing network models, in particular, are well suited to the high level of abstraction of workload and system component characterization necessary for the system designer to maintain an intuitive grasp on his work. These analytical models offer surprising accuracy and robustness considering their simplicity of structure and parameterization.

This work is concerned with extending the capability of queueing network models of computer systems and broadening our understanding of their robustness. We also consider workload characterization and program behavior, both important components in accurate parameterization of these models. Throughout this research we seek to provide the analyst with practical tools for more accurate models of computer systems. These tools may take the form of either model solution techniques or greater understanding of the modeling process.

One class of queueing network models (local balance [B1, C1, C5, G1, J1]) has received particular attention because it is solved by a simple analytical technique. The application of this solution technique is limited to local balance models (which must exclude explicit representation of memories, channels, and input/output processors). The first area of research we consider makes a step toward removing these restrictions by developing models with explicit representation of input/output processors. Accurate and inexpensive approximate solutions to these models are formulated. The techniques can be applied to more general queueing network models.

A second area of research is the investigation of patterning in workload characterization, specifically file references and resource requests. Accurate workload characterization and knowledge of the effects of approximate workload representations are as essential to effective system modeling as the representation of devices or components. The consequences of patterning in system file references to memory management strategies for a library file system are investigated. One strength of this research is in its reliance upon empirical reference strings in determining the validity of models and the performance of different memory management strategies.

In a third investigation the patterning of resource requests in queueing network models is considered. Both repetitive resource holding time sequences and request routing

sequences are constructed. The importance of these deterministic sequences is that they can be used to model transient effects in computer systems. We prove the strongly counter-intuitive fact that for local balance models the effect of patterning (in both holding time and routing) upon the performance measures of the model is nil. A study of non-local balance queueing network models leads us to conclude that the impact of patterning upon performance measures is minimal. This result allows the analyst to place greater confidence in the robustness of simple workload characterizations.

1.1 Related Work

Since a survey of previous work relating to the research in each chapter is presented by chapter, this section is deliberately compact. The great interest in queueing network models has resulted in a large and growing body of literature. Perhaps the most widely used class is that for which Jackson [J1] and Gordon and Newell [G1] demonstrated the existence of closed, product form solutions. Chandy [C1] discovered the underlying basis of the solution to be the property of "local balance". Baskett, Muntz, Chandy and Palacios [B1] extended this class of models to include customer classes. Chandy, Howard and Towsley more fully explored the properties of this class in [C5]. Buzen [B8] developed efficient algorithms for product form solutions for "typeless" networks in his study of system bottlenecks. Chandy, Herzog and Woo [C3] developed a technique for reducing

the computational requirements of these solutions by the coalescing of subnetworks into a single queue—a technique used in this work.

A broader class of queueing network models can be characterized by a matrix representation of a system of linear equations. Wallace and Rosenberg [W1] formulated efficient solution methods for this class. Recently Herzog, Woo and Chandy [H1] and Sauer [S1] have developed efficient recursive techniques which take advantage of regularities in the characteristic matrix. The limited range of applicability of models for which efficient exact solutions exist has prompted the growth of approximate solution methods. In [M2] Muntz presents a survey of these methods. Of particular relevance to our research are the decomposition techniques of Courtois [C10,C11] and Avi-Itzhak and Heyman [A4] because of their applicability to passive resource problems. Florkowski [F1] obtained an iterative method which explicitly considers active and passive resources. Browne, et al. [B7] apply decomposition and other techniques in a structured manner to obtain approximate solutions to a hierarchical model of a complex computer system. Brown [B6] considers memory contention in an approximate model of an interactive computer system.

An enormous body of literature exists for program behavior and memory management. Surveys of the literature can be found in Denning [D1], Coffman and Denning [C8], and Denning and Graham [D2] for dynamic memory strategies and related

program behavior. Foster [F2] surveys static strategies in his work on determining an optimal static file assignment utilizing queueing network models. Unfortunately, almost the totality of literature in these areas is concerned with page (or address) reference patterns for virtual memory systems. We are concerned with symbolic (name of executable file) reference patterns for which the closest research is reported by Madison and Batson [M1] and Batson and Brundage [B2] in their investigation of array reference patterns for a set of Algol programs.

Finally, the study of resource request patterns in queueing network models has received scant attention to date. The closest work is that of Lo [L1] in his observation and analysis of CPU burst time "modes" and this behavior's impact on CPU scheduling. Noetzel [N2] and Noetzel and Haley [N1] have investigated the impact of resource request repetition in simulation studies of different predictive scheduling algorithms.

Again, the reader is referred to the individual chapters for discussion on the interrelationship of the above research and the work presented.

1.2 Organization of the Chapters

In chapter I the motivation for the research is presented, related work to the research is mentioned, and the chapters summarized.

In chapter II an extended queueing network model incorporating the passive resource of I/O processors is developed.

Patterns in program behavior are not presented in this model. Existing solution methods to the model are surveyed and found inadequate. Approximation methods to the solution are developed and validated.

Patterns in program behavior are studied in chapter III. Empirical reference traces are used to parameterize and test models of referencing processes in program behavior. The impact of the empirically determined patterning upon memory allocation strategies is investigated.

In chapter IV the consequences of resource request patterns in local balance and non-local balance queueing networks are investigated. Resource request patterns are proved to have no impact on the performance measures of local balance models. Their impact is determined to be minimal for non-local balance models.

A summary of the research and its contributions comprises chapter V.

II. A Patternless Passive Resource Model of I/O Activity

2.1 Introduction

The motivation of this chapter is twofold: First, to develop a realistic model of computer systems which incorporates the competition of parallel I/O processes for the passive resource of I/O processors; and second, to formulate accurate and inexpensive solutions for the model. Until this work was developed, a realistic model required solution methods so computationally expensive that they discouraged its evaluation. On the other hand, rapid and accurate solution methods were available for much less realistic models. This work presents an accurate approximate solution method to a realistic model that is computationally inexpensive. The solution method is an improvement over previous methods in that (1) it is significantly more accurate, and (2) it is applicable to models with realistic I/O service time distributions. The approximate solution method we develop is applicable not only to the passive resource model presented in this chapter but also to a larger class of models of systems for which the maximum level of parallelism in any subsystem is fixed.

In this chapter we first present a brief description of an existing computer I/O system with characteristics which cannot be tractably modeled by existing techniques. These characteristics are competition for I/O processors; namely, peripheral processors and channels. We next provide a definition of passive resources and

formulate a queueing network model incorporating peripheral processor activity. Performance metrics for the model are considered. Three solution techniques for solving the model are examined and their applicability considered. These techniques are simulation, exact analytic methods, and approximate analytic methods. Previous work in analytic methods is discussed. An approximate analytic method incorporating a technique for simplifying the passive resource queueing network model is given, and the method is further refined to include more realistic service time distributions for the I/O devices. An extensive validation shows that the accuracy of the approximation method is good. Finally, the contributions of the approximation method to computer modeling and queueing theory are discussed.

2.2 Definitions

Active Resource: A resource characterized by a holding time distribution for processes. Examples include the central processing unit, disks, and drums.

Passive Resource: A resource characterized by the manner in which it is allocated and deallocated to processes by the system. The process holding time of a passive resource is not an inherent characteristic of the resource, but is instead determined by the holding times of associated active resources. Examples include memory, channels, and peripheral processors as used in CDC 6000 series computers.

2.3 Structure of the CDC 6000 Series Computer I/O System

We present the CDC 6000 series computer I/O subsystem as an example of a subsystem where contention for passive resources can result in serious system performance degradation. Although the modeling approach we take is general enough to be applied to many different computer systems, we will restrict ourselves at present to the CDC architecture to illustrate the approach.

The I/O subsystem is composed of peripheral processors (PP's), channels, disks, and drums. A PP is a programmable processor with its own memory and control and performs I/O functions. The PP's can access Central Memory independently of the Central Processing Unit (CPU) which performs most of the numerical computation for the machine. The University of Texas CDC 6600 has 10 PP's and the CDC 6400 has 7. For a description of the University of Texas system see Howard [H2]. For a description of the CDC 6600 see Thornton [T1] The 6000 series channels, capable of transmitting one 12 bit "byte" per microsecond, connect the PP's with CDC 844 and 808 disks.

Let us consider the steps required for a job to read data from the disk. The job executing in the CPU posts an I/O request. The Peripheral Program Monitor, polling the currently executing processes (jobs), picks up the request and assigns it to a PP. If no PP is available, the request is placed in a first-come-first-served queue from which PP's draw service requests. Once the I/O request is assigned a PP, the PP computes the disk sector address of the request and queues (in first-come-first-served order) for an I/O channel. After acquiring the channel, the PP performs the read,

transferring the data to a buffer in the PP's own memory. The PP in turn transfers the data by blocks into the user job's field length in Central Memory, releases the channel, and posts a message to the user job that the I/O request is satisfied. A write is handled similarly.

2.3.1 Dedicated and Pool PP's

PP's perform more than the above I/O functions to disks and drums. Other functions include communicating with the operator through a CRT console, performing the Peripheral Program Monitor function, and communicating with other computers. One PP each must be dedicated to the operator console and to the PP Monitor. Additional PP's may be dedicated to computer communications and input/output drivers if the delay time introduced by first-come-first-served queueing for these resources is intolerable. The remaining, non-dedicated, PP's will be termed "Pool PP's" as they form a pool of available peripheral processors.

2.3.2 The Pool PP as a Limiting Passive Resource

It is clear from the above description that the number of pool PP's available can play an important role in overall system performance. If the number of pool PP's is small compared to the number of jobs simultaneously desiring I/O, then there exists the possibility that pool PP's can be the system bottleneck. In any case, contention between jobs for pool PP's will result in degraded response time for I/O requests and hence degradation of overall system performance. It is this degradation that concerns us in the

following sections.

2.4 A Queueing Network Model of I/O Activity

We shall use the now-classic central server model [B8] as the basis of our analysis. These models have been extensively studied as they represent an economical means of modeling computer system performance using relatively simple workload characteristics (see [A4, B3, B4, B6, B7, M2, S1]).

2.4.1 The Central Server Model

The central server queueing network model of a computer system consists of a closed network of queues, servers, and interconnecting paths (see figure 2.1). The central server is the CPU. The I/O subsystem is composed of L input/output servers for which jobs may queue. The network is populated by M jobs, or customers, corresponding to the degree of multiprogramming. Jobs in the model queue up with probability p_i for I/O processor i to receive a mean processing burst of duration $1/\lambda_i$. We note that the p_i must sum to 1.

In the models of Buzen [B8], Gordon and Newell [G1], and Jackson [J1] the service times are assumed exponentially distributed. This assumption allows for tractable solutions for the performance characteristics of the system. The Markov state diagram for such a model with L=2 and M=3 is shown in figure 2.2. Models lifting this restriction are discussed in a later section.

The defect of this model most pertinent to our analysis is the property that any number of jobs (up to the degree of multi-

programming) are allowed to be engaged in I/O, whether queued or in service. It is evident from the previous discussion on pool PP's that in an I/O system with N pool PP's available, there can be at most N jobs queued in service in the I/O subsystem. We will denote this restriction on the number of active jobs in the I/O subsystem as a limit on the level of parallelism in the I/O subsystem.

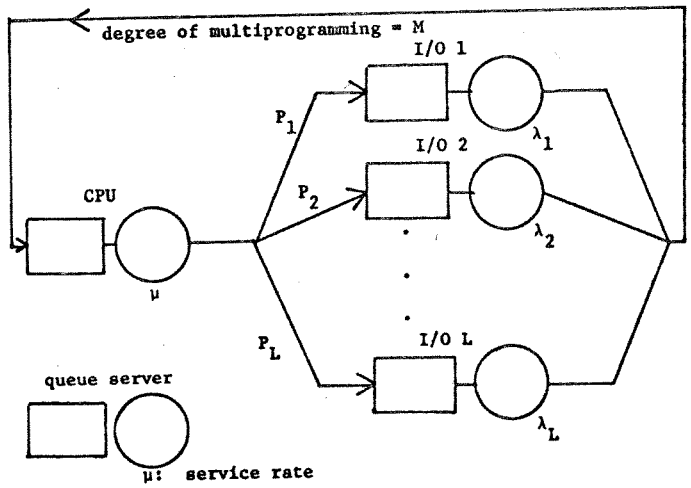
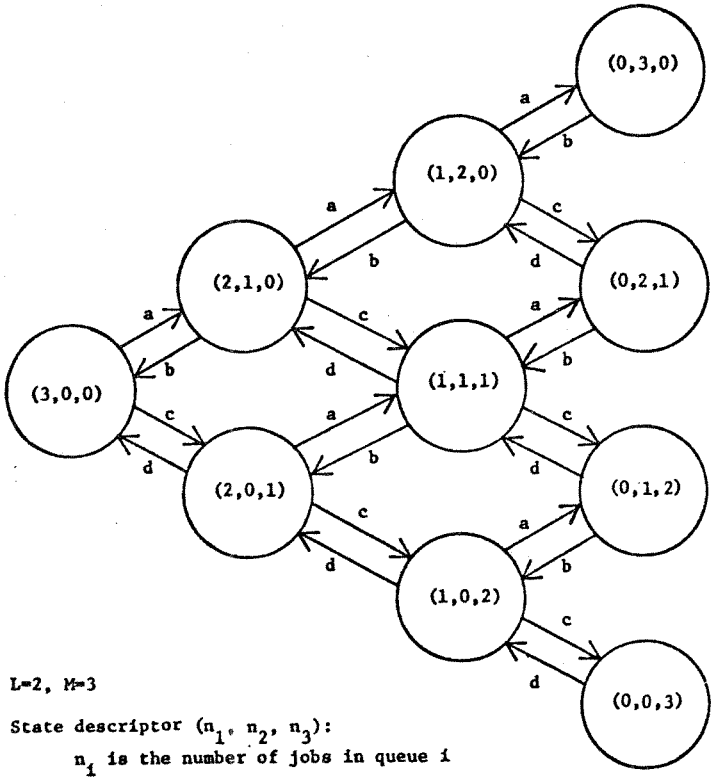


Figure 2.1: The Central Server Queueing Network Model



L=2, M=3

State descriptor (n_1, n_2, n_3):

n_i is the number of jobs in queue i

Transition rates: $a=p_1 \cdot \mu$, $b=\lambda_1$, $c=p_2 \cdot \mu$, $d=\lambda_2$.

Figure 2.2: Markov State Diagram for a Central Server Model with Exponential Service Rates

2.4.2 The Passive Resource Model

Let us now extend the central server model developed in the previous section to include the passive resource of PP's. We will label this extended model "the passive resource model." In this model I/O processing for a job will consist of the assignment of a PP, then the entering of an I/O queue i with branching probability p_i for a server which processes jobs with a mean service rate of λ_i. With no loss of generality we can set the mean service rate for the CPU, μ, to 1. We initially assume exponential holding time distributions throughout. Parameters of the model are μ, M: the degree of multiprogramming, N: the number of PP's, $P = \{p_i, i=2,\ldots,L+1\}$, and $\lambda = \{\lambda_i, i=1,\ldots,L\}$, where L is the number of I/O queues. The model is shown in figure 2.3. In this model PP's can be thought of as tokens; a job is refused admittance to an I/O queue without a token. Tokens are dispensed (when available) to a job awaiting I/O. Tokens are not released by a job until it exits the I/O queue upon completion of service, at which point the tokens are returned to the pool. If $N \geq M$ no contention for PP's exists, and the model reduces to the central server model of the previous section.

The Markov state diagram for the model of figure 2.3 with $M = 3$ and $N = 2$ is shown in figure 2.4. The state of the system is described by the $(L + 1)$-tuple $(n_1,\ldots,n_{L+1})$, where n_i is the number of jobs in queue i, the I/O queues being indexed $2,\ldots,L+1$. Note that in contrast to the central server model,

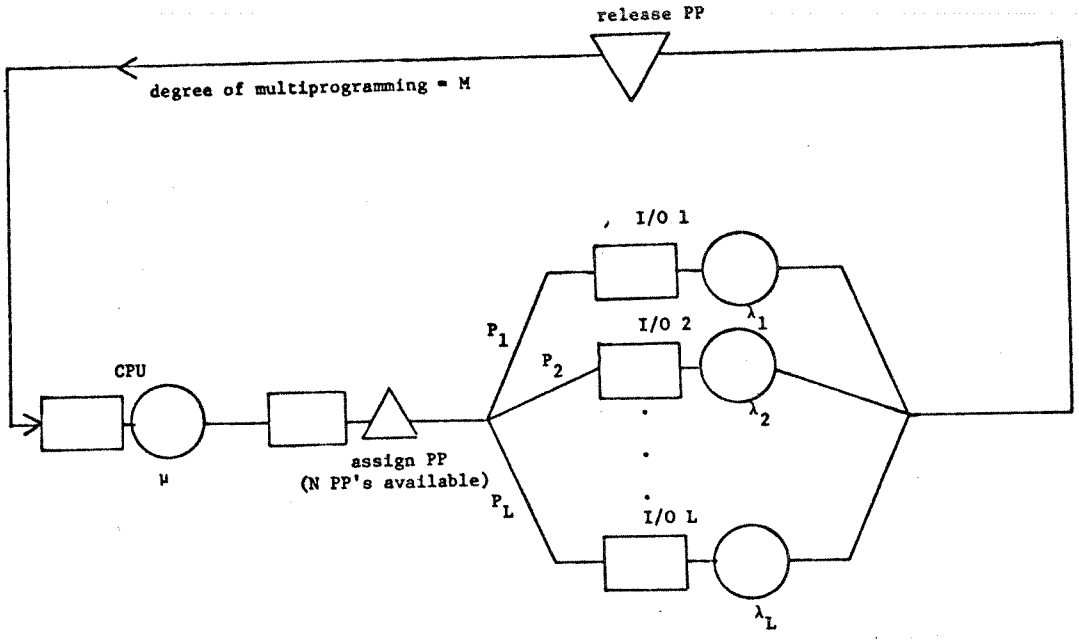
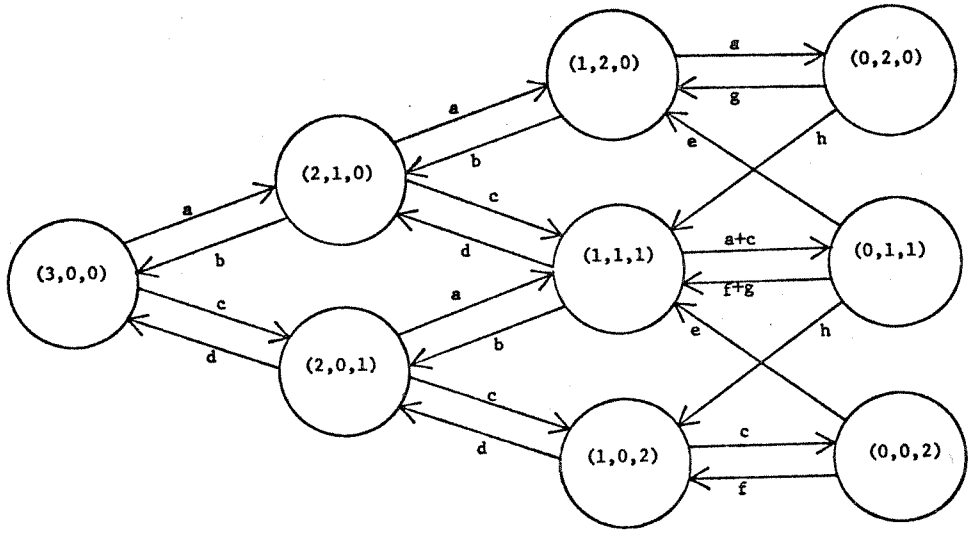


Figure 2.3: The Passive Resource Model



L = 2, M = 3, N = 2

State descriptor (n_1, n_2, n_3): n_i is the number of jobs in queue i

Transition rates: $a=p_1 \cdot \mu$, $b=\lambda_1$, $c=p_2 \cdot \mu$, $d=\lambda_2$, $e=p_1 \cdot \lambda_2$, $f=p_2 \lambda_2$, $g=p_1 \cdot \lambda_1$, $h=p_2 \cdot \lambda_1$.

Figure 2.4: Markov State Diagram for a Passive Resource Model

$\sum_{i=1}^{L+1} n_i$ is not strictly M, for M - $\sum_{i=1}^{L+1} n_i$ jobs may be queued for PP's.

We shall consider job thruput, defined for central server models as the rate of job completion and departure at the CPU queue, as the performance metric by which the degradation to system performance due to limiting the level of parallelism in the I/O subsystem is measured. Note that for μ=1, CPU thruput and utilization (the fraction of time the CPU is busy) are equal. We will use the two measures interchangeably when such is the case. Although other performance metrics such as mean waiting time for I/O service could be used, job thruput is the metric most closely related to the system's capacity to do work. Rather than concentrate on the effect of passive resource contention to individual jobs, we wish to consider the effect upon the system as a whole. We will next discuss the solution methods available toward solving this model and consider the tradeoffs between them.

2.5 Solution Methods to the Passive Resource Model

In this section we discuss the different techniques available to the systems analyst for solution of computer system models and the applicability of these techniques to the passive resource model.

2.5.1 Simulation

Perhaps the most conventional approach to computer system analysis is simulation. This is because simulation is the most

general technique available to the analyst and because many high-level simulation languages are readily available. The primary advantage of simulation over other techniques for the analysis of computer system models is its great generality. Unfortunately, great expense in both human and machine time is required by the technique. Simulation is also an approximate method; for increased accuracy the analyst must pay the cost of longer runs. Unresolved questions frequently facing the user of this technique are the level of detail which must be represented in the simulation for an accurate model, and the length and number of runs to achieve accurate resuls. For the passive resource model the level of detail is simply that stated in the definition of the model. Since the model is a renewal process we can use the techniques of Crane and Iglehart [C13, C14] to obtain confidence intervals for its solution. Thus these questions are resolved.

2.5.2 Exact Analytic Techniques

Exact solution techniques do not suffer from the approximation error inherent in other methods. Unfortunately, many interesting problems cannot be formulated so that they fall into the class of models which are tractable by exact techniques. Although generally less expensive in computer time than simulation methods, the computational requirements of many exact methods can easily outstrip the machine's resources for moderately complex models. We will discuss three exact methods in order of decreasing cost and generality.

2.5.2.1 Matrix Methods

The most general exact technique for solution of queueing network models requires the solution of the steady-state probabilities of each state of the stochastic process. Assume that there are S states in the discrete-state system. A system of S linear equations must be solved to obtain the steady state probabilities. The equations are specified by a matrix of dimensions SxS; hence the name of the method. Various techniques with an operation count on the order of S^3 for the most general cases may be used to solve the set of equations. We find two drawbacks to this method: (1) the specification of the matrix requires an enumeration of every state of the system and every transition between states and (2) the expenses in space for storage of the matrix and in operations for its solution. Wallace and Rosenberg [W1] have developed sparse matrix storage and manipulation techniques to support a power-iteration algorithm that converges to an exact solution. The algorithm greatly reduces the operation count for this method, with the practical result that up to 1,000-state models are tractable.

2.5.2.2 Recursive Methods

Recently Herzog, Woo, and Chandy [H1] and Sauer [S1] have developed algorithms to solve queueing network models recursively by taking advantage of regularities in the matrix specifying the system. The advantage of the technique is its great saving in computer time and space over the more general matrix methods. The disadvantage involves its range of applicability being limited to models displaying

regular state structures. Typically, a different algorithm must be coded by the analyst for each class of queueing network models. For complex models the complexity of transitions between states makes the method cumbersome and eventually intractable.

2.5.2.3 Product Form Method

For a restricted class of queueing network models Jackson [J1] and Gordon and Newell [G1] demonstrated the existence of a closed solution of a product form. Chandy [C1] discovered that the underlying basis of the product form solution is a property of the queueing network that "local balance" holds. Baskett, Muntz, Chandy, and Palacios extended the class of networks for which local balance, and hence the product form solution, holds to include customer classes in [B1]. This property is even more fully investigated in [C5]. Buzen [B8] developed algorithms which made the calculation of certain performance metrics very inexpensive. These and other algorithms were incorporated by Keller in a program for the automatic analysis of queueing network models for which local balance holds [C2]. A property of this class of models is that subsystems of the queueing network model can be coalesced into a single queue without impacting the characteristics of the remainder of the network. This property [C3] greatly reduces the computational requirements for solution.

2.5.3 Approximate Analytic Techniques

Generally, an approximation method is used to solve a more accurate model that proves intractable to exact solution techniques.

Approximation methods are useful when the increased accuracy of the model representation more than offsets the inaccuracy of the approximation. Approximation methods can be categorized (Muntz [M2]) into two types: diffusion and decomposition. The diffusion approximation method approximates a discrete-state random process by a continuous-state process. When applied to queueing network models, the discrete-state process state is queue length. The reader is referred to [M2] and its references for a discussion of the method. At the expense of oversimplifying, this method is most accurate for models with large degrees of multiprogramming and subject to the greatest error for models with small degress of multiprogramming.

The analysis developed in this chapter falls into the decomposition category. Thus our discussion of work employing decomposition techniques will be more detailed. Decomposition techniques analyze subsystems of a typically intractable model and utilize the results of these analyses in refinements resulting in an approximate, tractable model. At this point the commonality of techniques lumped into this category diverges because no uniform method presently exists for decomposition. The historical trend has seen specific techniques developed in an ad hoc manner to suit the specific application of the analyst. Recently, however, attempts at more general techniques applicable to general queueing network models of computer systems have been made.

Courtois [C10,C11] applies the approximation technique of Simon and Ando [S4] to queueing networks. Informally, the approximation technique aggregates a set of system state variables into a

single aggregate variable. Each aggregate variable corresponds to a submatrix lying on the diagonal of the Markovian matrix of the entire system. Each submatrix (usually corresponding to a queueing network subsystem) may be solved to obtain approximate steady-state probabilities for the state variables comprising the aggregate variable. The weak couplings between submatrices supply the information necessary to solve for the steady-state probabilities of the aggregate variables. One application of the technique is a model of an interactive system with a limited degree of multiprogramming. The state aggregation technique is used to reduce a many queue system to an approximate two queue model; one of the queues represents a subsystem of queues in the original model. The decomposition of the queueing network model into subsystems greatly simplifies the computation required for solution, since the number of states in a queueing network model grows combinatorially with the number of queues and customers in the model.

The approximation is only as good as two underlying assumptions upon which the technique relies: (1) a local equilibrium is reached within each subsystem independent of the behavior of the global system, and (2) the global system, ignoring the transient effects within subsystems, tends toward an equilibrium defined by the weak couplings between subsystems. The validity of these assumptions is formally developed in [C1], and [C12] is a thorough, formal treatment of the accuracy of the technique.

Chandy, Herzog, and Woo in [C4] present an iterative method

for approximating closed queueing network models for which local balance does not hold due to an assumption of general service time distributions. The algorithm employed utilizes Norton's theorem [C3] to decompose the general model into an approximate two queue model. One queue in the approximate model represents the composite behavior of all but one queue in the original model--the solitary queue comprising the second queue in the approximate two queue model. In the two queue model exponential service time distributions are assumed for the composite queue while the other queue maintains its original distribution. The two queue model is solved to obtain queue length and wait time distributions. This step is cyclically applied for every queue in the original network. The technique performs this cycle, tests the queue length and thruput results of each queue to assure a proper interface between the queues (conserving the number of jobs in the system, for example), and adjusts the mean service rates for every composite queue for which the interface is not within an error bound. The algorithm iterates until all interfaces are within bounds. Accuracies of within typically 5% can be achieved for many models with non-exponential distributions.

Another iterative method by Florkowski [F1] considers the effect of coincident resource holding by processes in closed queueing network models. Florkowski employs a hierarchic model consisting of "primary" and "secondary" resources, corresponding to the active and passive resources we define. Processes can hold both a primary and a number of secondary resources simultaneously (for example a disk and channel, respectively). This property makes an

exact solution intractable. An approximate iterative method is presented which incorporates mean service times (including queueing) for secondary resources into the mean holding times (excluding queueing) of primary resources. Although the accuracy of the approximation is claimed to be good, [F1] does not present a rigorous validation.

Avi-Itzhak and Heyman in [A4] present approximation techniques for open queueing networks. The analysis of [A4] follows the same line as Courtois' investigation in [C10] for closed queueing network models. A closed queueing network model is reduced by a similar decomposition technique to a single server with an exponential service rate which is a function of the degree of multiprogramming. The property the authors investigate is the limiting of the degree of multiprogramming (call it M) to a constant. This is achieved by the approximation of holding the queue-length dependent service rate of the composite queue to that service rate corresponding to queue length M for all queue lengths exceeding M. Both exponential and constant service time distributions are considered, as well as priority classes. In contrast to other work in this area, error bounds for the method are obtained for some simple two queue models.

Willemain in [W2] presents a technique which considers open queueing network models with primary and secondary servers very similar to Florkowski's closed models. Again, coincident resource holding by processes is the property which makes the model intractable to exact solution. In contrast to the work of Florkowski,

the thrust of Willemain's work is in considering the merits of different primary/secondary resource allocation disciplines. He manages to achieve approximation bounds for the system performance under one discipline.

Decomposition techniques are rigorously applied by Browne, et al. [B7] in their analysis of a large computer system. Imposing a hierarchically structured approach throughout the modeling process results in a set of hierarchically interlinked models. At the highest level in the model hierarchy is a central server model in local balance with each queue typically representing a modular subsystem of the computer system. Separate queueing network models comprise the next hierarchic level and are used to analyze subsystem behavior in detail. The lower level (or "micro") models are solved to parameterize the higher level (or "macro") model. The macro model, in turn, is solved to furnish certain parameters to the micro models. An iterative process is employed which results in the convergence of system performance metrics. The modularization allows perturbation analyses to be conducted which test the sensitivity of overall system performance to the behavior of the subsystems.

Brandwajn [B4] presents a queueing network model of a virtual memory interactive computer system similar to the easily-solved central server models with exponential service distributions. However, the service times at the CPU are represented by lifetime functions which correlate the page-fault rate of jobs to the memory requirements of the workload on the system. Courtois' decomposition technique is used to form a two queue machine repairman approximate

model. The mean service time of the repair queue is assumed exponential and is appropriately adjusted to reflect the effect of memory contention imposed by the lifetime functions. Brandwajn obtains the mean user response time and CPU utilization as functions of the number of terminals, number of page frames, and page-frame allocation strategies, and program fault-rate behavior.

Brown [B6] studies the effect of memory contention for a queueing network model of an interactive system using decomposition techniques. In this non-virtual memory model the empirical job memory requirement distribution was used in a recursive technique to obtain the mean "repair" time as a function of the degree of multiprogramming in an approximate machine repairman model. The approximate model was solved to obtain mean response times, CPU utilization, memory utilization, and other performance metrics.

2.5.4 Application of Methods to the Passive Resource Model

The model is of course tractable to simulation. However, to avoid the expense of simulation we turn our attention to other methods. Unfortunately, the model is not in local balance and thus the relatively inexpensive product-form solution does not hold. Matrix methods must be rejected because the tractability of the methods does not extend throughout the realm of realistic parameters to the passive resource model. The number of states simply grows too large. Let S be the number of states of the system which is parameterized by M, N, and L, all previously defined. We characterize the I/O distribution by T, the number of exponential stages in each

I/O distribution. For purposes of example let each of the L I/O queues be characterized by the same number of states, T. The number of states of the system is

$$S = 1 + \sum_{i=0}^{L-i} \binom{L}{i} T^{(L-i)} \left[\sum_{j=L-i}^{N-1} \binom{j-1}{j-L+1} + (M-N+1) \binom{N-1}{N-L+1} \right] \qquad (2.1)$$

and is derived in detail in Appendix B. Values for S at various M, N, L, and T may be found in table 2.1. We see that the number of states exceeds even the capabilities described in [W1]. Likewise, recursive techniques are impractical for such large values of S due to the complexity of interconnection between the states. Approximation techniques thus represent the only tractable means of solving the model, as they typically are less sensitive to the size of the Markovian state space.

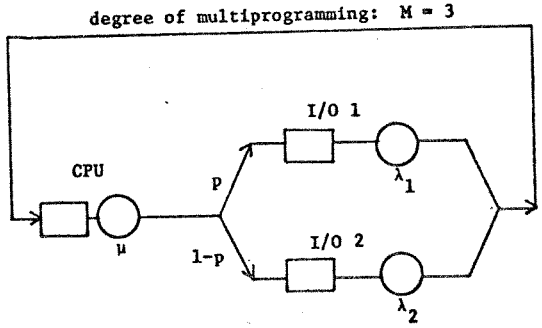
Table 2.1: Number of Markovian States of the Passive Resource Model

M:	degree of multiprogramming	3	3	5	5	8	8
N:	level of parallelism	2	2	4	4	6	6
L:	number of I/O queues	2	2	3	4	3	4
T:	number of I/O stages	2	3	3	3	3	3
S:	number of states	21	37	478	1465	1828	8602

2.6 Norton's Theorem

2.6.1 Discussion

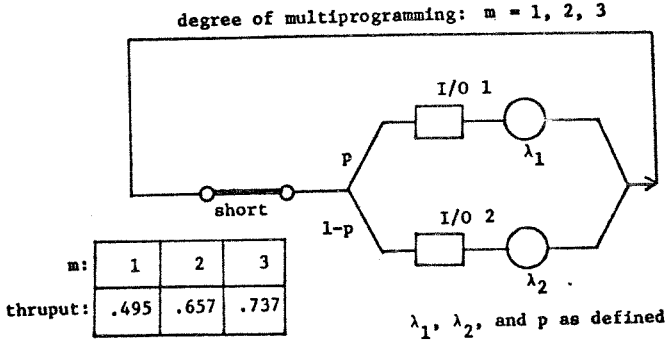
The Norton's theorem technique [C3] for coalescing the behavior of a subsystem of queues in a queueing network model into



Service rates: $\mu = 1.0$, $\lambda_1 = .55$, $\lambda_2 = .45$

Branching probability: $p = .5$

Figure 2.5: Central Server Model



m:	1	2	3
thruput:	.495	.657	.737

λ_1, λ_2, and p as defined in figure 2.5

Figure 2.6: Central Server Model with shorted CPU

a single queue is crucial to the development of our approximation technique. Thus the technique and an example are presented in this section. A detailed discussion and proof of the theorem can be found in Appendix A.

Consider the central server model of section 2.4.2 (figure 2.5, for example) parameterized by M, L, μ, P, and λ as previously defined. We assume exponential service rates in order that the model be in local balance. Norton's theorem allows us to replace the L I/O queues by a single composite queue with queue-length dependent rates to form an equivalent two queue network. By "equivalent" we mean that the queue-length distribution, waiting time distribution, utilization, and thruput for the CPU server will be the same for both models. The queue-length dependent service rates for the composite queue are determined by analyzing a modified version of the original central server model. In the modified model (figure 2.6) the CPU queue is missing, in effect being "shorted out." All other parameters to the modified model remain identical to the original, except the degree of multiprogramming m. The thruput of the modified model, T(m), evaluated at the "short," is computed for m=1,...,M. The two queue model (figure 2.7) may now be parameterized by rate μ for the CPU queue and rate T(i), i=1,...,M for the composite I/O queue. The final network is closed with the two queues in series, with degree of multiprogramming M, and satisfies local balance.

Norton's theorem has been proved [C3] to hold for queueing

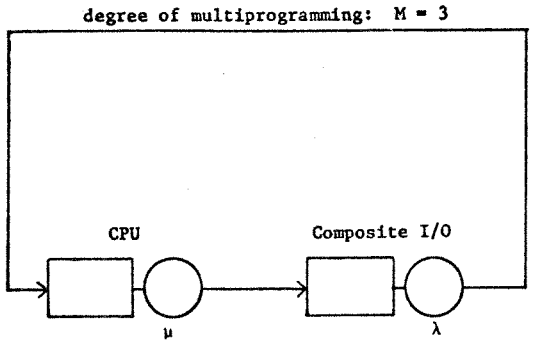
network models in local balance. Approximation techniques [C4, K2, S1, C10] for solutions of queueing network models not in local balance have utilized the above technique, accepting the local balance approximation in return for greater tractability in the solution of the two queue model. An example is presented next to help fix the technique in the reader's mind.

2.6.2 Example

Consider the central server model of figure 2.5. We determine the thruput of the network when the CPU's mean service time is set to 0 (figure 2.6) with m customers in the system, m=1,...,M. These values as a function of m are shown in the table of figure 2.6. The equivalent two queue network is shown in figure 2.7.

2.7 Approximate Solutions to the Passive Resource Model

In this section we present two approximation techniques for the solution of the passive resource model specified in section 2.4.2. The first approximation technique is a straightforward adaptation of Norton's theorem resulting in a two queue model obeying local balance. An example is given. The second approximation technique is more accurate and results in a two queue model which does not obey local balance. The accuracies of both approximations are tested against exact solutions obtained via the general Markov technique of section 2.5.2.1 and via a simulation. An upper bound is proposed to the solution of the passive resource model with both exponential and hypoexponential I/O distributions.



I/O queue size =	1	2	3
I/O service rate =	.495	.657	.737

Figure 2.7: Equivalent Central Server Model

2.7.1 The Local Balance Approximation

Consider the passive resource model of section 2.4.2. We construct an approximate model by applying Norton's theorem to the I/O subsystem to obtain a two queue model. The two queues are the CPU and a "composite" I/O queue with a queue-length dependent service rate which is constrained by the number of available PP's. The approximate model obeys local balance and thus is amenable to product-form solution techniques. A program which solves queueing network models in local balance by product-form techniques, ASQ [C2], is used to analyze all models in the algorithm. The algorithm is specified below.

Step 1. Construct the central server model of section 2.4.1 corresponding to no contention for a passive resource. Apply Norton's theorem via the technique of section 2.6.2 to the I/O subsystem to form an equivalent composite I/O queue with service rate T(m), m=1,...,M, where M is the degree of multiprogramming. Use ASQ to solve the intermediate models.

Step 2. Redefine the I/O composite queue service rate T'(m), m=1,...,M such that for N PP's, $N \leq M$,

$$T'(m) = \begin{cases} T(m) & m \leq N \\ T(N) & m > N \end{cases}.$$

Step 3. Solve the resulting two queue model via ASQ.

Step 4. Stop.

2.7.1.1 Example

Consider the central server model of figure 2.5. We extend the model by considering the restriction of two PP's for degree of

multiprogramming three. Applying Step 1 we obtain an I/O composite queue rate T(m) defined by the table of figure 2.7. We apply Step 2 to redefine the I/O composite service rate as T'(1)=.495, T'(2)=T'(3)=.657. From Step 3 we obtain a CPU utilization of .566.

This approximation technique was developed independently (in sometimes very different forms) by Keller and Chandy [K2], Courtois [C10], and Williams and Bhandiwad [W3] for closed queueing network models, and by Avi-Itzhak and Heyman [A4] for open queueing network models. One reason for the concurrent development was undoubtedly the ease in which the intermediate queueing network models can be solved, as they obey local balance. Another inducement is the simplicity of a technique "whose time had come." Keller and Chandy [K2] presented the approximation technique considered here and gave an error estimate based on simulation runs. Williams and Bhandiwad proposed the technique for general subsystems of queueing network models but gave no quantitative error estimates, dismissing the error as "small." Courtois models an interactive computer system by a closed queueing network model in which jobs in the computer are assumed to be in one of three states (running, ready, or suspended). The rate at which jobs receive service and leave the computer is dependent upon the transition rates between these three internal states and the number of jobs in the machine. Courtois considers the case in which the maximum number of jobs allowed to occupy the computer is fixed at a constant, and the service rate for the degrees of multiprogramming greater than the maximum is held fixed at the service rate for the maximum. Thus, although the subsystem considered by Courtois is different than that

considered in this work, the approximation principle is the same. Courtois does not consider the error of his approximation in [C10] but refers the reader to [C12]. A systematic validation of the technique is not presented. Similarly, an approximate open network model by Avi-Itzhak and Heyman considers the entire machine as a subsystem of queues to be "condensed" into a single queue. Error bounds are obtained for only two cases of a very simple two queue subsystem, that for which the maximum number of jobs permitted in the subsystem is 1, and that for which there is no maximum. Exact analytic solutions for more realistic cases were found intractable. The relative error for the trivial cases analyzed is found to be greatest when the arrival rate into the subsystem is nearly that of the composite service rate.

Considerations stemming from [K2] led us to believe that the error of this technique cannot be dismissed as small for realistic models. This is verified in section 2.8.2. Thus a more accurate approximation method was developed and is presented in the next section.

2.7.2 The Non-Local Balance Approximation

Again, consider the passive resource model of section 2.4.2. In contrast to the assumptions of the preceding section we will consider the second moment of the I/O distributions. Let us characterize the service distributions only by their mean and coefficient of variation, C_v. Again, we construct an approximate model by applying Norton's theorem to the I/O subsystem and limiting the maximum service rate. We make the admittedly false assumption

that the initial central server model obeys local balance in order to obtain these rates. We then assign a coefficient of variation (other than 1) to the composite I/O queue and solve the reduced two queue network for the CPU utilization.

The reader should note we have introduced two important complexities into the previous algorithm: first, determining the C_v of the composite I/O queue; and second, solving a two queue model which does not obey local balance. Fortunately, Sauer in [S1] attacks the second problem with excellent results. We will use Sauer's solution technique and thus must digress to discuss his methods.

In [S1], chapter 5, Sauer develops approximate solutions for central server models with first-come-first-served queueing disciplines and non-exponential service times. He reduces the original model to a two queue model consisting of the CPU and a composite I/O with a service distribution representing an aggregate of all the individual I/O distributions. The means of the composite (queue-length dependent) distribution is obtained by applying Norton's theorem to the I/O subsystem. Attention is restricted to the first two moments of the I/O distributions to keep the computation tractable. He obtains the composite coefficient of variation by weighting each I/O C_v by its branching probability. Thus, while the mean composite service time is queue-length dependent, the composite C_v is not. Given the composite C_v and mean service times he assumes a generalized Erlang distribution (figure 2.8) of standard form so

that the first two moments of the composite distribution completely specify the generalized Erlang distribution. Sauer shows that it is possible to formulate a generalized Erlang distribution for an arbitrary distribution with a differentiable Laplace transform so that the first two moments of both distributions agree. Furthermore, if the composite distribution is hyperexponential of the form of figure 2.9, a generalized Erlang distribution with no imaginary variables may be constructed so that all moments of both distributions agree (they are identical).

Sauer implemented an algorithm that solves a two queue queueing network model with FCFS queueing disciplines and a generalized Erlang distribution for both queues. The algorithm incorporates the recursive techniques put forward by Herzog, et al. in [S1]. We shall incorporate this algorithm into our approximation method and designate it algorithm S.

It should be noted that algorithm S when used as an approximation technique for central server models of the type specified can introduce error at three steps: first, determining the mean of the composite I/O distribution via Norton's theorem; second, by assuming a weighted sum I/O C_v; and third, by ignoring third and higher order moments in the composite I/O distribution. The reader is referred to [S1] for greater detail.

By incorporating algorithm S into the non-local balance approximation technique we then face the remaining problem of estimating the composite I/O C_v. Let us first consider the case of exponential I/O queues for the passive resource model with parameters

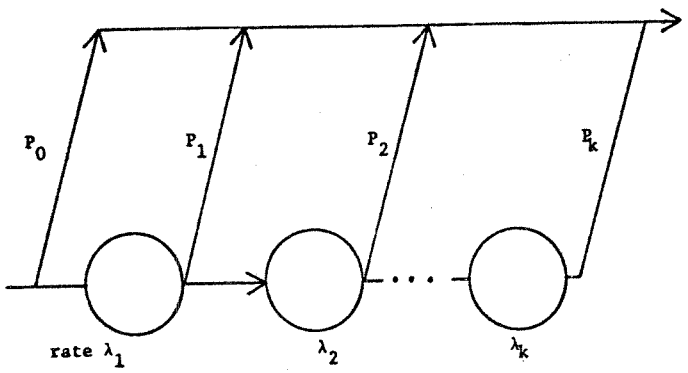


Figure 2.8: A k Stage Generalized Erlang Distribution

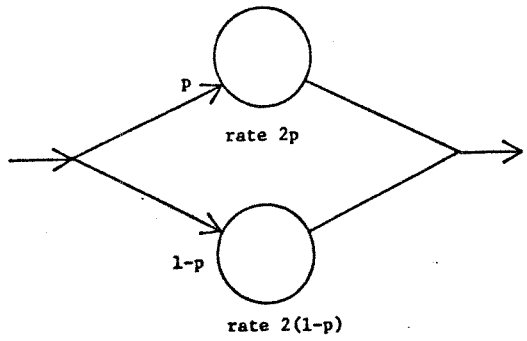


Figure 2.9: A Two Stage Hyperexponential Distribution

as defined. For notational ease we will lump the parameters P, λ, L, μ, and T into the single parameter R. Thus M, N, and R completely specify the model. Now we note that for N=M the local balance approximation is exact and the composite service distribution is exponential. Thus the C_v of the composite I/O queue, C(M,N,R), is 1. We also note that for the case of N=1 the composite distribution is hyperexponential and it can be shown ([K3], p. 142) that

$$C(M,1,R) = \rho^2\left(\sum_{i=1}^{L} (p_i/\lambda_i^2)\right)-1 \qquad (2.2)$$

where

$$\rho = \left(\sum_{i=1}^{L} p_i/\lambda_i\right)^{-1}. \qquad (2.3)$$

We observe that for this case $C(M,1,R) \geq 1$. Let us abbreviate C(M,1,R) by simply C1. Now for models with N such that $1<N<M$, no such simple closed analytic form for C(M,N,R) exists, since even the service time distribution for the composite queue is not well understood. However, it seems reasonable to assume that for the approximate model we are constructing an equivalent service time distribution (equivalence defined as resulting in the exact CPU utilization) will possess a C_v which is monotonically decreasing from C1 to C(M,M,R) as N increases from 1 to M. This implies a linear approximation to the equivalent coefficient of variation would be in order, namely:

$$C(M,N,R) = \left(\frac{M-N}{M-1}\right) C1 + \left(\frac{N-1}{M-1}\right) C(M,M,R). \qquad (2.4)$$

By taking appropriate derivatives of the generating function for the parallel stage Erlang distribution [K3, p. 144] we can obtain C1 for arbitrary I/O service distributions (so long as they have differentiable Laplace transforms). For example, for a model with two I/O queues with 2-stage Erlang hypoexponential distributions and mean service rates of λ_1, λ_2, branching probabilities p_1, p_2, and equal mean holding times per stage, the coefficient of variation for N=1 is

$$\frac{3\rho^2}{2}\left(\frac{p_1}{\lambda_1^2} + \frac{p_2}{\lambda_1^2}\right)^{-1} \qquad (2.5)$$

while for 3 stages it is

$$\frac{4\rho^2}{2}\left(\frac{p_1}{\lambda_1^2} + \frac{p_2}{\lambda_2^2}\right)^{-1} . \qquad (2.6)$$

We now have all the information necessary to parameterize the approximate model which is solved by the following algorithm.

Step 1. Construct the central server model of section 2.4.1 corresponding to no contention for a passive resource. Apply Norton's theorem via the technique of section 2.6.2 to the I/O subsystem to form an equivalent composite I/O queue with service rate T(m), m=1,...,M, where M is the degree of multiprogramming. Use ASQ to solve the intermediate models.

Step 2. Redefine the I/O composite queue service rate T'(m), m=1,...,M such that for N PP's, N≤M,

$$T'(m) = \begin{cases} T(m) & m \le N \\ T(N) & m > N \end{cases} .$$

Step 3. Determine C1, the coefficient of variation of the composite queue when N=1.

Step 4. Estimate the C_v of the I/O composite queue when N=M by forming a weighted sum of individual queues' coefficients of variation, where the weight for queue i is the branching probability p_i. Denote this coefficient of variation by C(M,M,R).

Step 5. Set the coefficient of variation, C, for the composite queue to the linear approximation (2.4).

Step 6. Use algorithm S to solve the two queue model parameterized in steps 2 and 5.

Step 7. Stop.

2.7.3 A Conjectured Upper Bound

The conjecture that the equivalent coefficient of variation monotonically decreases with increasing N (section 2.7.2) suggests that a bound on the CPU utilization might be obtained by substituting a value of C(M,N,R) higher or lower than the linear estimate (2.4) in the non-local balance approximation. We present a conjectured upper bound on the CPU utilization for the passive resource model and support it by an intuitive argument.

Let U(M,N,R) be the CPU utilization for the passive resource model parameterized by M, N, and R which are previously defined. Let U(M,N,R,C) be the CPU utilization for the two queue model resulting from applying the non-local balance approximation to the passive resource model specified by M, N, and R, with a given coefficient of variation C. Define C_{min} as $\min[C(M,N,R), 1 \le N \le M] = C(M,M,R)$.

Conjecture: $U(M,N,R) \le U(M,N,R,C_{min})$.

Thus $U(M,N,R,C_{min})$ is a conjectured upper bound to the CPU utilization of the passive resource model.

Let us consider the passive resource model with two exponential I/O queues (T=1) with $p_1=p$, $p_2=1-p$, $\lambda_1=2p$, and $\lambda_2=2(1-p)$. For N=M, C(M,N,R) as defined in (2.5) is exact. Also, from our discussion of algorithm S, $U(M,1,R)=U(M,1,R,C)$ and $U(M,M,R)=U(M,M,R,C)$, where $C=C(M,M,R)=C_{min}(M,N,R)=1$. Now it is a recognized, though unproven, property of the two queue model that the CPU utilization monotonically decreases with increasing coefficient of variation for the I/O queue (see Shedler [S2], for example). The CPU utilization is greatest when the I/O service time is constant ($C_v=0$) and decreases as C_v increases. So, for the above model we can state

$$U(M,1,R)=U(M,1,R,C)\leq U(M,1,R,C_{min})$$

and likewise

$$U(M,M,R)=U(M,M,R,C)=U(M,M,R,C_{min}).$$

Unfortunately, determining C(M,N,R) for 1<N<M proves intractable since the composite I/O distribution for 1<N<M is not well defined (the motivation behind the linear approximation of equation 2.4). Establishing the upper bound for these two cases is as far as our argument will carry us. We must turn to the validation results of the following section for further support of the conjecture for models with 1<N<M and leave the proof of the conjecture to later work.

2.8 Accuracy of Approximations

In order to test the accuracy of the two approximation methods just presented, a validation was performed against exact and simulation solutions for the passive resource model over a range of parameters. In this section we present the structure of the passive resource model used in the validation, the parameter space explored, the exact and simulation solution methods used, and the validation results.

2.8.1 The Validation Model

In order to keep the cases tractable to exact analysis the number of I/O queues is set to two in the validation model. Both exponential and hypoexponential distributions for the I/O queues are used. Distributions with one, two, and three serial exponential stages are used for the hypoexponential distributions. The CPU service distribution is assumed exponential with mean rate $\mu=1$. Three variables are employed from which the P and λ of the model are derived. The first is ρ, reflecting the total I/O work rate and defined by

$$\rho = (p_1/\lambda_1 + p_2/\lambda_2)^{-1} \tag{2.7}$$

while the second, α, reflects the balance between I/O queues and is defined by

$$\alpha = (p_1\lambda_2)/(p_2\lambda_1) \tag{2.8}$$

while the third, β, is the ratio between I/O service rates;

$$\beta = \lambda_2/\lambda_1. \qquad (2.9)$$

ρ is allowed to range from .1 to 1.0, for when ρ exceeds 1 the CPU becomes saturated and the impact of contention for passive resources in the I/O subsystem is minimal. For values of $\rho \ll 1$ the thruput of the system is so small as to be impervious (within reasonable bounds) to the errors resulting from the approximation methods. α is allowed to range from .1 to 1.0, the upper bound being chosen because if all other variables are held constant in the central server model CPU thruput is symmetric about $\alpha=1$, for the roles of the I/O queues are merely reversed for the cases $\alpha=r$ and $\alpha=1/r$, where $r>0$. The lower bound is in consideration of realistic models of I/O subsystems for which it is assumed the device workloads are balanced within an order of magnitude. β is arbitrarily allowed to range from .1 to 1.0, reflecting a device transfer rate imbalance not greater than an order of magnitude. M and N are drawn from the integer set [B3, with the condition that N<M. The coefficient of variation for the I/O queues ranges from $1/\sqrt{3}$ to 1 and is determined by the number of exponential stages, T, comprising the service distribution.

An extensive validation in the parameter space so defined was made for exponential I/O servers. Points in the space for hypoexponential I/O servers were chosen by selecting all exponential I/O cases with error greater than 1% and evaluating them with the number of exponential stages, T, equal to 2 and 3 (corresponding to a C_v of $1/\sqrt{2}$ and $1/\sqrt{3}$, respectively).

2.8.2 Solution Methods and Costs

Two methods for obtaining exact and near-exact solutions to the passive resource model were used. For cases with less than 100 states a matrix technique was used which provides an exact value for the CPU utilization. It was possible to take advantage of the regular structure of the state space associated with each value of N, N=1,...,3, in formulating the matrices, so that only one algorithm for each value of N was necessary. The algorithms were coded in Fortran. For cases with large numbers of states, a simulation was used. The simulation was coded in the high-level simulation language ASPOL [C9] and the confidence interval techniques of Crane and Iglehart [C14,C3] were employed. For all simulation results the CPU utilizatio- obtained is within $\pm$.005 of the exact value with 90% confidence.

Solutions from both of the above techniques were validated against ASQ models for which N=M and each against the other for numerous cases in which N<M. Simulation results were within $\pm$.005 (with 90% confidence) of results derived by the exact analytic techniques. We thus hold the results obtained from the above techniques to be accurate because of this 3-way consistency.

Costs (in time) of the various methods are measured in CDC 6600 CPU seconds and do not include compilation and loading. The time for each approximation was less than .05 seconds for cases with both small and large (>100) states. The matrix method ranged in cost from .05 seconds to .4 seconds, depending on the number of

states of the system. Simulation runs varied from 8 to 195 seconds to obtain results of the desired accuracy for cases with large numbers of states.

2.8.3 Error of Approximations

Two hundred and six (206) models were solved in the parameter space defined. The values of p_1, λ_1, and λ_2 obtained from (2.7), (2.8), and (2.9) for values of ρ, α, and β are displayed in table 2.2. The error of the approximations are displayed in tables 2.3-2.7. We will define the values displayed as they appear in the tables from left to right. The model number labels the case considered. The parameters ρ, α, β, M, N, and T, which have already been defined, are given next. U_{exact} is the exact CPU utilization obtained by the techniques discussed in the preceding section. U_{nt} is the CPU utilization obtained by employing the Norton's theorem technique for the model with the artifice of N=M and exponential I/O service distributions, corresponding to a model ignoring passive resource contention and non-exponential I/O service distributions. ERR_{nt} is defined as $(U_{nt} - U_{exact})\cdot 100$, representing the percentage error. Subsequent errors are likewise defined and displayed. U_{lb} is the CPU utilization obtained by the local balance approximation. U_{ap} is the CPU utilization obtained from the non-local balance approximation. For those cases with T>1 (163-206) the value for U_{bnd}, the estimated upper bound conjectured in section 2.7.3, is given. Finally, the estimated coefficient of variation used in the non-local balance approximation is given.

A brief summary of the different approximations' errors for all exponential I/O cases may be found in the following limits of error by method:

(1) $ERR_{nt} \in [0, 33.8\%]$

(2) $ERR_{lb} = ERR_{bnd} \in [0, 9.7\%]$

(3) $ERR_{ap} \in [-2.5\%, 2.0\%]$.

Similarly, an error summary of the methods for hypoexponential I/O distributions is:

TABLE 2.2: Model Parameters

ρ	α	β	p_1	λ_1	λ_2
.1	.1	.1	.500	.550	.055
.1	.1	1.0	.090	.100	.100
.1	.5	.1	.833	.250	.025
.1	.5	1.0	.333	.100	.100
.1	1.0	.1	.909	.181	.018
.1	1.0	1.0	.500	.100	.100
.5	.1	.1	.500	2.750	.275
.5	.1	1.0	.090	.500	.500
.5	.5	.1	.833	1.250	.125
.5	.5	1.0	.333	.500	.500
.5	1.0	.1	.909	.909	.090
.5	1.0	1.0	.500	.500	.500
1.0	.1	.1	.500	5.500	.550
1.0	.1	1.0	.090	1.000	1.000
1.0	.5	.1	.833	2.500	.250
1.0	.5	1.0	.333	1.000	1.000
1.0	1.0	.1	.909	1.818	.181
1.0	1.0	1.0	.500	1.000	1.000

TABLE 2.3 - ERROR OF APPROXIMATIONS FOR MODELS 1-40

U.EXACT : EXACT CPU UTILIZATION
U.NT : CPU UTILIZATION FOR N=M, LOCAL BALANCE
U.LB : CPU UTILIZATION FOR LOCAL BALANCE APPROXIMATION
U.AP : CPU UTILIZATION FOR NON-LOCAL BALANCE APPROXIMATION
ERR : (U - U.EXACT)*100
COVAR : ESTIMATED COEFFICIENT OF VARIATION

MODEL	RHO	ALPHA	BETA	M	N	T	U.EXACT	U.NT	ERR.NT	U.LB	ERR.LB	U.AP	ERR.AP	COVAR
1	.1	.1	.1	2	1	1	.098	.107	.91	.099	.11	.099	.08	1.53
2	.1	.1	.1	3	1	1	.108	.110	1.01	.106	.04	.108	.03	1.53
3	.1	.1	.1	3	2	1	.108	.110	.12	.109	.04	.109	.03	1.27
4	.1	.1	.1	5	1	1	.108	.110	1.08	.108	0.00	.108	0.00	1.53
5	.1	.1	.1	5	2	1	.109	.110	.10	.109	.01	.109	0.00	1.48
6	.1	.1	.1	5	4	1	.110	.110	0.00	.110	0.00	.110	0.00	1.13
7	.1	.1	.1	10	1	1	.108	.110	1.00	.108	0.00	.108	0.00	1.53
8	.1	.1	.1	10	2	1	.109	.110	.10	.109	0.00	.109	0.00	1.47
9	.1	.1	.1	10	4	1	.110	.110	0.00	.110	0.00	.110	0.00	1.35
10	.1	.1	1.0	2	1	1	.099	.107	.88	.099	0.00	.099	0.00	1.00
11	.1	.1	1.0	3	1	1	.108	.110	.97	.108	0.00	.108	0.00	1.00
12	.1	.1	1.0	3	2	1	.109	.110	.08	.109	.01	.109	.01	1.00
13	.1	.1	1.0	5	1	1	.108	.110	1.00	.108	0.00	.108	0.00	1.00
14	.1	.1	1.0	5	2	1	.109	.110	.10	.109	0.00	.109	0.00	1.00
15	.1	.1	1.0	5	4	1	.110	.110	0.00	.110	0.00	.110	0.00	1.00
16	.1	.1	1.0	10	1	1	.108	.110	1.00	.108	0.00	.108	0.00	1.00
17	.1	.1	1.0	10	2	1	.109	.110	.10	.109	0.00	.109	0.00	1.00
18	.1	.1	1.0	10	4	1	.110	.110	0.00	.110	0.00	.110	0.00	1.00
19	.1	.5	.1	2	1	1	.096	.124	2.56	.099	.08	.099	.03	2.14
20	.1	.5	.1	3	1	1	.100	.138	3.81	.100	.02	.100	.01	2.14
21	.1	.5	.1	3	2	1	.128	.138	1.02	.128	.04	.128	.01	1.57
22	.1	.5	.1	5	1	1	.100	.147	4.71	.100	0.00	.100	0.00	2.14
23	.1	.5	.1	5	2	1	.129	.147	1.86	.129	0.00	.129	0.00	1.86
24	.1	.5	.1	5	4	1	.145	.147	.22	.145	.01	.145	0.00	1.29
25	.1	.5	.1	10	1	1	.100	.150	4.99	.100	0.00	.100	0.00	2.14
26	.1	.5	.1	10	2	1	.129	.150	2.13	.129	0.00	.129	0.00	2.02
27	.1	.5	.1	10	4	1	.145	.150	.47	.145	0.00	.145	0.00	1.76
28	.1	.5	1.0	2	1	1	.099	.124	2.48	.099	0.00	.099	0.00	1.00
29	.1	.5	1.0	3	1	1	.100	.138	3.79	.100	0.00	.100	0.00	1.00
30	.1	.5	1.0	3	2	1	.128	.138	.99	.128	.01	.128	.01	1.00
31	.1	.5	1.0	5	1	1	.100	.147	4.71	.100	0.00	.100	0.00	1.00
32	.1	.5	1.0	5	2	1	.129	.147	1.85	.129	0.00	.129	0.00	1.00
33	.1	.5	1.0	5	4	1	.145	.147	.21	.145	0.00	.145	0.00	1.00
34	.1	.5	1.0	10	1	1	.100	.150	4.99	.100	0.00	.100	0.00	1.00
35	.1	.5	1.0	10	2	1	.129	.150	2.13	.129	0.00	.129	0.00	1.00
36	.1	.5	1.0	10	4	1	.145	.150	.47	.145	0.00	.145	0.00	1.00
37	.1	1.0	.1	2	1	1	.099	.128	2.93	.099	.05	.099	0.00	2.25
38	.1	1.0	.1	3	1	1	.100	.147	4.76	.100	.01	.100	0.00	2.25
39	.1	1.0	.1	3	2	1	.132	.147	1.44	.133	.02	.132	-.01	1.62
40	.1	1.0	.1	5	1	1	.100	.165	6.52	.100	0.00	.100	0.00	2.25

TABLE 2.4 - ERROR OF APPROXIMATIONS FOR MODELS 41-80

U.EXACT : EXACT CPU UTILIZATION
U.NT : CPU UTILIZATION FOR N=M, LOCAL BALANCE
U.LB : CPU UTILIZATION FOR LOCAL BALANCE APPROXIMATION
U.AP : CPU UTILIZATION FOR NON-LOCAL BALANCE APPROXIMATION
ERR : (U - U.EXACT)*100
COVAR : ESTIMATED COEFFICIENT OF VARIATION

MODEL	RHO	ALPHA	BETA	M	N	T	U.EXACT	U.NT	ERR.NT	U.LB	ERR.LB	U.AP	ERR.AP	COVAR
41	.1	1.0	.1	5	2	1	.133	.165	3.19	.133	0.00	.133	0.00	1.94
42	.1	1.0	.1	5	4	1	.160	.165	.56	.160	0.00	.160	-.01	1.31
43	.1	1.0	.1	10	1	1	.100	.181	8.14	.100	0.00	.100	0.00	2.25
44	.1	1.0	.1	10	2	1	.133	.181	4.81	.133	0.00	.133	0.00	2.11
45	.1	1.0	.1	10	4	1	.160	.181	2.14	.160	0.00	.160	0.00	1.83
46	.1	1.0	1.0	2	1	1	.099	.128	2.88	.099	0.00	.099	0.00	1.00
47	.1	1.0	1.0	3	1	1	.100	.147	4.69	.100	0.00	.100	0.00	1.00
48	.1	1.0	1.0	3	2	1	.133	.147	1.42	.133	.01	.133	.01	1.00
49	.1	1.0	1.0	5	1	1	.100	.165	6.52	.100	0.00	.100	0.00	1.00
50	.1	1.0	1.0	5	2	1	.133	.165	3.19	.133	0.00	.133	0.00	1.00
51	.1	1.0	1.0	5	4	1	.160	.165	.56	.160	0.00	.160	0.00	1.00
52	.1	1.0	1.0	10	1	1	.100	.181	8.14	.100	0.00	.100	0.00	1.00
53	.1	1.0	1.0	10	2	1	.133	.181	4.81	.133	0.00	.133	0.00	1.00
54	.1	1.0	1.0	10	4	1	.160	.181	2.14	.160	0.00	.160	0.00	1.00
55	.5	.1	.1	2	1	1	.404	.450	4.57	.429	2.44	.417	1.33	1.53
56	.5	.1	.1	3	1	1	.439	.500	6.02	.467	2.72	.454	1.46	1.53
57	.5	.1	.1	3	2	1	.481	.500	1.88	.498	1.68	.490	.87	1.27
58	.5	.1	.1	5	1	1	.474	.536	6.19	.492	1.81	.484	1.01	1.53
59	.5	.1	.1	5	2	1	.511	.536	2.49	.532	2.09	.523	1.25	1.40
60	.5	.1	.1	5	4	1	.531	.536	.47	.536	.47	.533	.18	1.13
61	.5	.1	.1	10	1	1	.496	.549	5.29	.500	.33	.499	.22	1.53
62	.5	.1	.1	10	2	1	.537	.549	1.21	.544	.73	.543	.54	1.47
63	.5	.1	.1	10	4	1	.545	.549	.43	.549	.43	.548	.29	1.35
64	.5	.1	1.0	2	1	1	.429	.450	2.12	.429	0.00	.429	0.00	1.00
65	.5	.1	1.0	3	1	1	.467	.500	3.30	.467	0.00	.467	0.00	1.00
66	.5	.1	1.0	3	2	1	.496	.500	.39	.498	.19	.498	.19	1.00
67	.5	.1	1.0	5	1	1	.492	.536	4.38	.492	0.00	.492	0.00	1.00
68	.5	.1	1.0	5	2	1	.530	.536	.57	.532	.18	.532	.18	1.00
69	.5	.1	1.0	5	4	1	.535	.536	.07	.536	.07	.536	.07	1.00
70	.5	.1	1.0	10	1	1	.500	.549	4.96	.500	0.00	.500	0.00	1.00
71	.5	.1	1.0	10	2	1	.544	.549	.51	.544	.03	.544	.03	1.00
72	.5	.1	1.0	10	4	1	.549	.549	.04	.549	.03	.549	.03	1.00
73	.5	.5	.1	2	1	1	.403	.491	8.79	.429	2.56	.411	.75	2.14
74	.5	.5	.1	3	1	1	.435	.579	14.38	.467	3.15	.445	.98	2.14
75	.5	.5	.1	3	2	1	.539	.579	4.04	.558	1.96	.537	-.11	1.57
76	.5	.5	.1	5	1	1	.467	.668	20.10	.492	2.52	.476	.89	2.14
77	.5	.5	.1	5	2	1	.578	.668	8.98	.612	3.41	.583	.48	1.86
78	.5	.5	.1	5	4	1	.654	.668	1.34	.664	.97	.650	-.42	1.29
79	.5	.5	.1	10	1	1	.493	.733	24.07	.500	.72	.496	.35	2.14
80	.5	.5	.1	10	2	1	.615	.733	11.83	.640	2.47	.623	.82	2.82

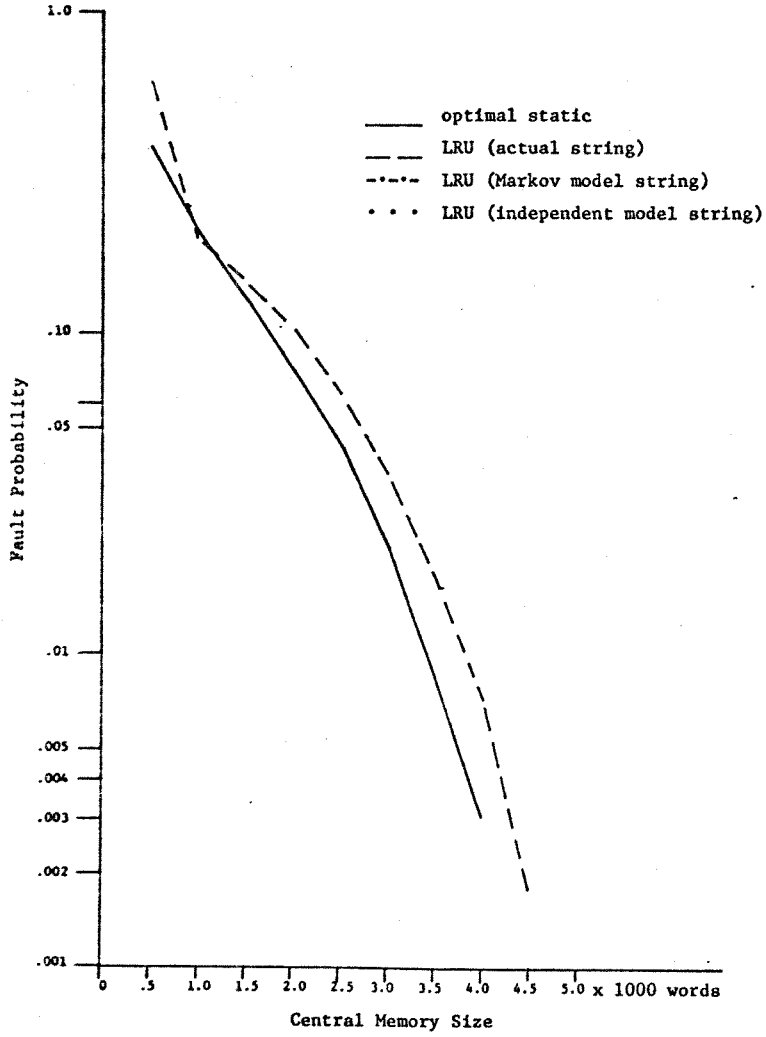


Figure 3.15: Reference Fault Probability vs. CM Size for Tape5

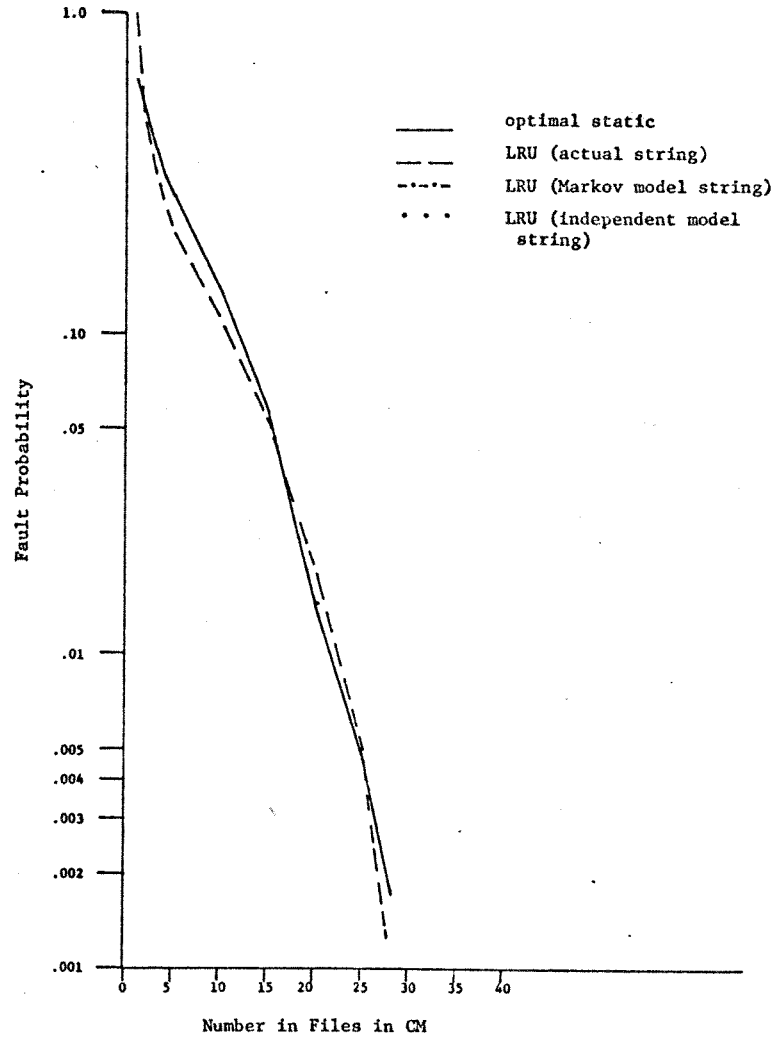


Figure 3.16: Reference Fault Probability vs. M for Tape5

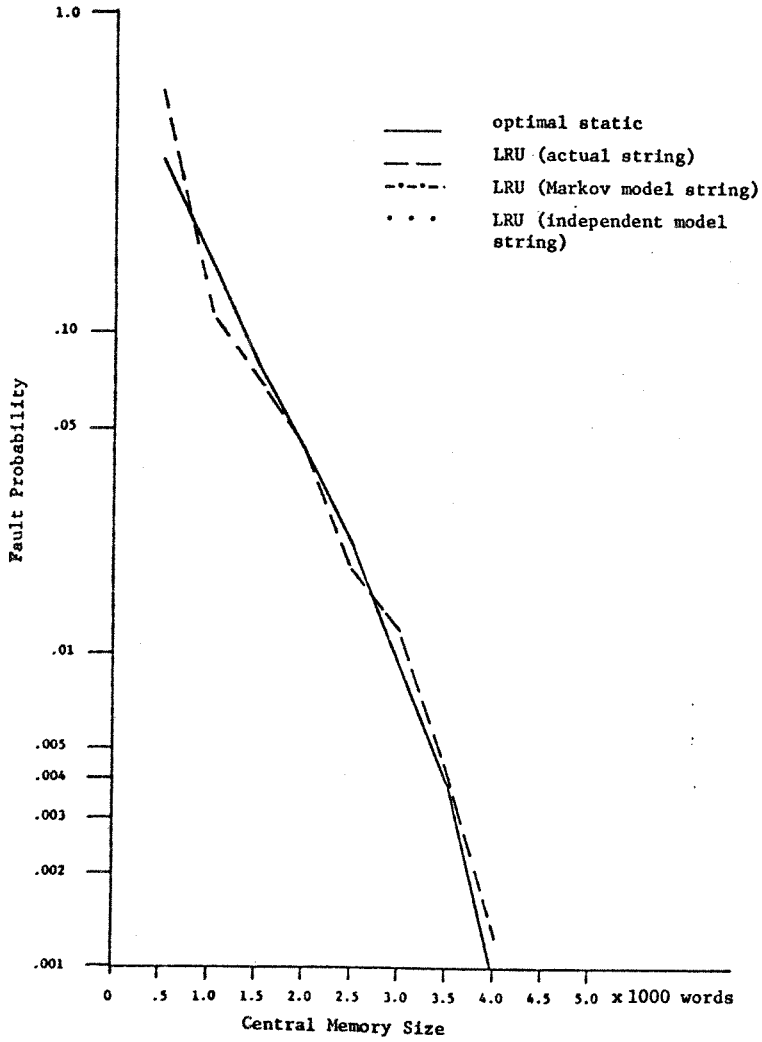


Figure 3.17: Reference Fault Probability vs. CM Size for Tape4a

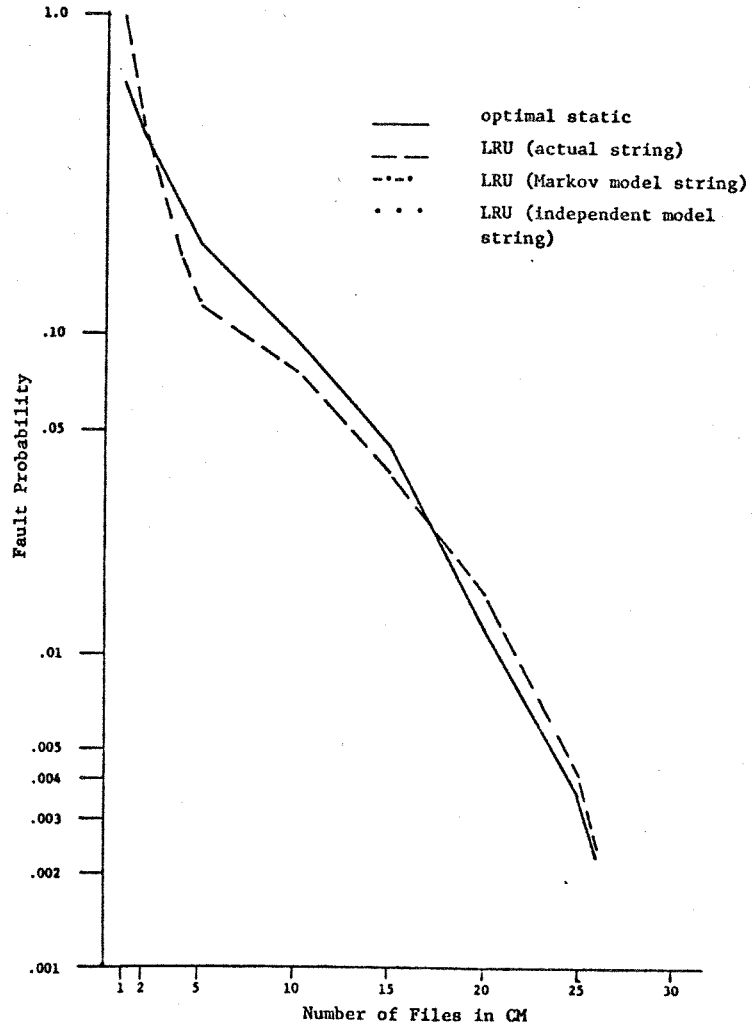


Figure 3.18: Reference Fault Probability vs. M for Tape4a

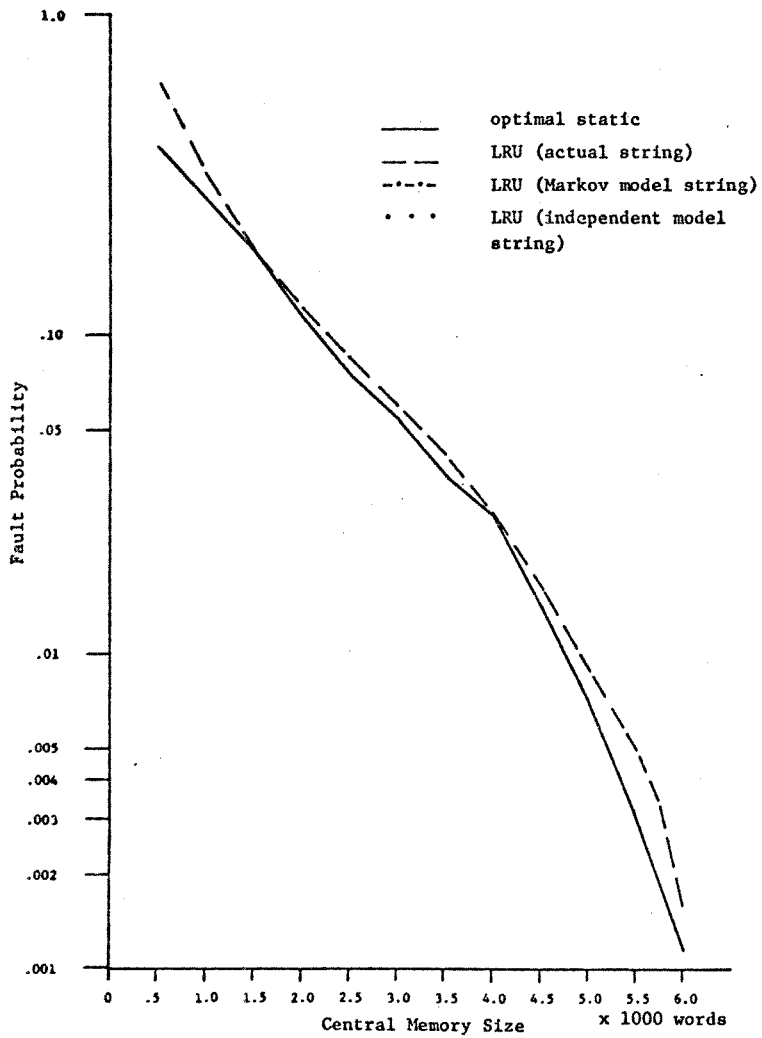


Figure 3.19: Reference Fault Probability vs. CM Size for Tape4

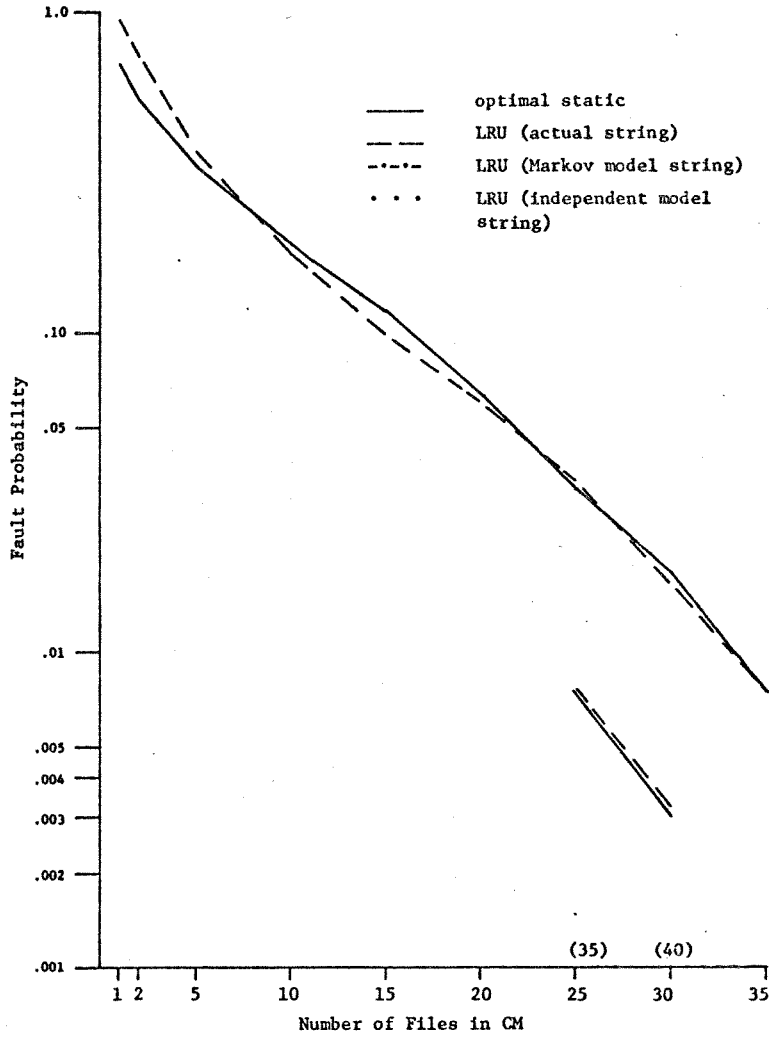


Figure 3.20: Reference Fault Probability vs. M for Tape4

better for larger CM and M. The same is true for the composite tape (figure 3.15) for CM. For different values of M, however, we note that the two curves interweave throughout the range studied. The same is true for the batch monostream and multistream strings (figures 3.18 and 3.20, respectively). We note that the batch monostream curves (figure 3.17) is also interleaved for CM. Curiously, for the batch multistream, CM case (figure 3.19) the optimal static is superior to LRU over the entire range, although the curves are very close. Thus the effect of file sizing makes a difference in this qualitative analysis; a difference so slight, however, that sizing could be ignored.

A comparison between the batch (tape4) and interactive traces reveals the impact of the workload environment upon the strategies. Note that the batch environment is more favorable to the dynamic strategy than the interactive environment. This is a reasonable result, as the total number of user programs active during the course of the trace is smaller for the batch environment. Thus the total behavior of programs over the time interval studied is less homogenized (more dynamic).

3.10 Conclusions

This chapter considered the effect of possible patterns in passive resource requests on a library file system under different memory management strategies. Two goals of our investigation were (1) to increase our knowledge of program behavior, specifically file reference patterning, and (2) to determine the effect of this patterning

on memory management strategies for the library file system. The investigation was broken into two principal phases. The conclusions we draw from each phase follow.

First, a performance comparison was made between allocation strategies via trace-driven simulation models of the file system. We conclude that a static allocation strategy will outperform a dynamic strategy in the actual system. This conclusion is based on the fact that an LFU (∞) allocation strategy outperformed an LRU strategy over a range of memory sizes. The optimal static strategy, in turn, will outperform LFU (∞), since in the long run LFU (∞) produces a static allocation with CM packed according to greatest-frequency-first. This intuitively obvious relationship was verified in initial comparisons which we do not present.

Second, an investigation was made of the user program reference generation process by testing the validity of reference generation models. We conclude that user program file references do display the property of locality but not to the degree to which a dynamic allocation strategy would better the performance of an optimal static strategy in actual implementation. The reason for this conclusion is the observation that the dynamic strategy is, at best, only slightly better than the static. This slight advantage is far outweighed by greater overhead for the dynamic strategy. We observe that a simple Markov process is a poor model for reference generation, but more accurate than a model assuming the independence of references. From this we conclude that the assumption of independence of references does not hold for this system.

More complex Markov models to adequately predict the reference string are intractable.

A positive result of this analysis is the recommendation that the existing static strategy in use by the system be changed to the optimal static strategy. The expense of implementing the strategy is minimal, consisting of the collection of accurate long-term reference frequencies. From the reference frequencies and program sizes algorithm 3.1 can be used to generate a partitioning of the existing library between CM and disk so as to minimize the reference fault probability. How often the library should be re-partitioned depends upon the rate at which the frequencies change. From our analysis we conclude that the reference frequencies do not change significantly within the approximately 30 minute interval recorded by the trace mechanism. This does not mean the reference frequencies do not change over longer periods.

IV. The Effects of Resource Request Patterns in Queueing Network Models

4.1 Introduction

We have considered the possibility of locality in system file references in the previous chapter. In this chapter we examine the effects of more generalized customer behavior patterns upon system performance. Again we base our examination upon queueing network models. The central question in this investigation is "How do customer resource request behavior patterns impact existing queueing network models of computer system performance?" The motivation for asking this question comes from the observation that jobs do exhibit patterns in their behavior. One behavior pattern is locality of references, discussed in the last chapter. Another pattern is CPU burst time "modes" investigated by Lo [L1]. The increasingly popular queueing network models for which product-form solutions hold are seemingly insensitive to many types of patterned job behavior, remaining accurate performance predictors of systems in which behavior patterns surely exist. Why? The answer to this question affects the robustness of existing queueing network models (and hence our confidence in their accuracy). If the impact of behavior patterns is important, then clearly the analyst must carefully characterize the workload behavior patterns, a formidable task since the practical definition of pattern is necessarily vague. If, however, the impact of patterning is negligible, then the analyst may ignore behavior

patterns and be satisfied that grosser workload characterizations will suffice.

In this chapter we define a means of formally modeling patterns of active resource requests for customers in queueing network models, designated customer task graphs (ctg's). It is shown that this formalism can represent a very large class of behavior patterns. We prove that for queueing network models obeying local balance [C1] customer behavior patterns cannot impact performance measures of the models, so long as the ctg's representing arbitrary behavior patterns affect the same server workloads upon the model. We next consider queueing network models with arbitrary ctg's which are not in local balance. Since closed form solutions to these models do not exist, we cannot make as definitive a statement as for local balance models. A series of experiments are conducted over what are considered reasonable computer job behavior patterns and computer system models. The results of the experiments for these models show that job behavior patterns have little impact on the common performance metrics of the systems. The implications of these results for the systems analyst are then considered.

4.2 Customer Task Graphs

We define the sequencing of customer service requests by a directed graph which we label a customer task graph (ctg). The ctg is composed of n nodes, N_i, i=1,...,n. Associated with each N_i is the server S_i requested by the customer and a customer type T_i. Let the set of servers be S, of size s, and the set of types T, of

size t, such that $S_i \epsilon S$ and $T_i \epsilon T$, i=1,...,n. Let an arc from N_i to N_j represent the possibility of the customer next specifying a request for server S_j with type T_j upon completion of service at server S_i with type T_i. Let $q_{i,j}$ represent the non-zero probability associated with this event, with $\sum_j q_{ij} = 1$.

Now consider a queueing network populated by M customers whose service requests are determined by a ctg, with all customers obeying the same ctg. The service time distribution at each server S_j, j=1,...,s, may vary by customer type. Chandy [C1] and Baskett, Chandy, Muntz, and Palacios [B1] have determined closed form solutions to a class of these queueing networks which have the property of local balance. The ctg is implicitly defined in this class of networks by the branching probabilities p(i,r:j,s), which is the probability of a customer of type r transiting to queue j with type s upon completion of service at queue i. We define a graph notation instead of utilizing the p(i,r:j,s) convention because the ctg notation utilizes fewer customer types, and because the graph convention appears more natural for the representation of customer behavior.

Consider the example ctg in figure 4.1. The p(i,r:j,s) notation would require n_1+n_2-1 types to insure the sequencing of n_1 accesses to queue 1 followed by n_2 accesses to queue 2, whereas the equivalent ctg requires but two types (at the expense of n_1+n_2 nodes). This example makes clear the equivalence of the two notations, as we need only to specify one type in the p(i;r:j,s)

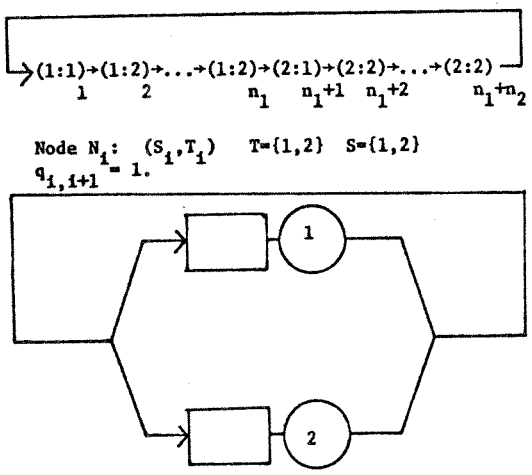


Figure 4.1: A Customer Task Graph and Its Associated Queueing Network

notation for each node in the ctg.

4.3 Proof of Equivalence of Customer Task Graphs

In this section we consider ctg queueing networks which meet the local balance condition. We prove that for every ctg queueing network with many customer types there exists an equivalent queueing network with a single type. Two queueing networks are equivalent if they have the same queue-length distributions by server, with queue length being defined as the total number of customers occupying the queue and its server. If two ctg queueing networks are equivalent to the same single-type queueing network, then they are obviously equivalent. The consequences of this property are discussed at the end of this section.

Consider the class of ctg queueing networks described in [B1]. In the preceding section we established that the ctg notation and that of [B1] are equivalent. We slightly modify the notation of [B1] for ease of exposition. We define a closed queueing network, QN, as composed of L queues, their associated servers, and M customers. There is an arbitrary but finite number of customer types R. Customers traverse the network and change type by branching probabilities p(i,r:j,s), with the transition being from queue i to queue j and from type r to type s. The service discipline at each queue is either processor-shared (PS) or first-come-first-served (FCFS). Service times for FCFS servers are assumed exponentially distributed and identical for all types. Service time distributions for PS servers must have rational Laplace transforms and may vary by customer

type.

Let μ_{ir}^{-1} be the mean service time for a customer of type r at server i with $\mu = \{\mu_{ir}, i=1,\ldots,L \text{ and } r=1,\ldots,R\}$. Let e_{ir} be defined by the set of linear equations

$$\sum_{L,R} e_{ir} \cdot p(i,r:j,s) = e_{js}.$$

The e_{ir} may be interpreted as the relative frequency of visits to queue i for customers of type r and are defined to within an arbitrary multiplicative constant. Closed form solutions for aggregate states of the system are derived in [B1], with the probability of the system in state S being

$$P[S=(y_1,y_2,\ldots,y_L)] = Gg_1(y_1)g_2(y_2)\ldots g_L(y_L) \qquad (4.1)$$

where $y_i=(n_{i1},n_{i2},\ldots,n_{iR})$ and n_{ir} is the number of customers in queue i of type r. G is a normalizing constant and is defined as $\Sigma P[S=(y_1,y_2,\ldots,y_L)]$, summed over $y_1+\ldots+y_L=M$. The g_i are defined as

$$g_i(y_i) = n_i! \prod_{r=1}^{R} (1/n_{ir}!)\beta_{ir}^{n_{ir}} \qquad (4.2)$$

where

$$\beta_{ir} = \begin{cases} e_{ir}/\mu_{ir} \text{ for PS queues} \\ e_{ir}/\mu_i \text{ for FCFS queues} \end{cases} \qquad (4.3)$$

Let us define another marginal distribution by introducing the aggregate state $A = (n_1,n_2,\ldots,n_L)$ so that

$$P[A=(n_1,n_2,\ldots,n_L)] = Gf_1(n_1)f_2(n_2)\ldots f_L(n_L) \qquad (4.4)$$

where n_i is the total number of customers in queue i and

$$f_i(n_i) = \sum_{T} g_i(y_i) \qquad (4.5)$$

where $T = \{n_{i1}+n_{i2}+\ldots+n_{iR} = n_i\}$.

Theorem. For every queueing network QN as defined there exists an equivalent queueing network QN′ with R′=1 such that for any aggregate states A and A′, P[A] = P[A′].

Proof. Substituting (4.2) into (4.5) we obtain

$$f_i(n_i) = \sum_{T} n_i! \prod_{r=1}^{R} (1/n_{ir}!)\beta_{ir}^{n_{ir}}. \qquad (4.6)$$

We note that (6) is a multinomial expansion, thus

$$f_i(n_i) = (\beta_{i1}+\beta_{i2}+\ldots+\beta_{iR})^{n_i} \qquad (4.7)$$

We can define a single type for QN′ such that

$\beta_i' = \sum_{r=1}^{R} \beta_{ir}$. Thus $f_i'(n_i) = f_i(n_i)$ by (4.7).

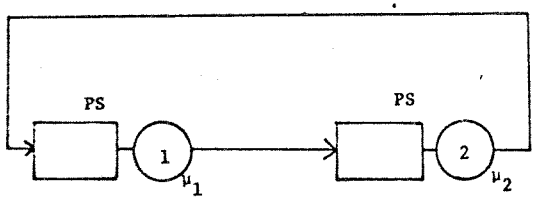
From (4.4) P[A] = P[A]′.

The implications of this property of ctg queueing network models in local balance is profound: specific patterns in customer behavior have no impact upon the queue-length distributions of the network. The practical implication is that specific job behavior patterns play no role in determining the performance of computer systems which are adequately described by local-balance queueing network models. The importance of this property to the computer system analyst is more fully discussed in the last section of this

chapter.

4.3.1 Example

Consider the queueing network of figure 4.2. The $p(i,r;j,s)$, e_{ir}, M, and L are defined in the preceding section. The queueing discipline is PS for both queues. β_1 and β_2 are defined in the proof. We observe that the marginal aggregate state distribution for both the original and equivalent networks are identical.



$p(1,1:2,1)=1/2$, $p(1,1:2,2)=1/2$, $p(1,2:2,1)=1$, $p(1,2:2,2)=0$

$p(2,1:1,1)=1$, $p(2,1:1,2)=0$, $p(2,2:1,1)=1/3$, $p(2,2:1,2)=2/3$.

$e_{11}=1/2$, $e_{12}=1/2$, $e_{21}=1/4$, $e_{22}=3/4$.

$\mu_{11}=1$, $\mu_{12}=2$, $\mu_{21}=2$, $\mu_{22}=3$.

$\beta_{11}=1/2$, $\beta_{12}=1/4$, $\beta_{21}=1/8$, $\beta_{22}=1/4$.

Figure 4.2: An Example of CTG Equivalence

TABLE 4.1: State Probabilities for the Example of Figure 4.2

State description: (y_1,y_2), $y_1=(n_{11},n_{12})$, $y_2=(n_{21},n_{22})$ where n_{ij} is the number of customers of type j in queue i. M=2.

n_{11},n_{12}	n_{21},n_{22}	$g(y_1)\ g(y_2)$	$s(y_1)\ s(y_2)$
2 0	0 0	β_{11}^2	.25
0 2	0 0	β_{12}^2	.0625
1 1	0 0	$2\beta_{11}\beta_{12}$	.25
1 0	1 0	$\beta_{11}\beta_{21}$	.0625
1 0	0 1	$\beta_{11}\beta_{22}$	.125
0 1	1 0	$\beta_{12}\beta_{21}$	.03125
0 1	0 1	$\beta_{12}\beta_{22}$	.0625
0 0	2 0	β_{21}^2	.015625
0 0	1 1	$2\beta_{21}\beta_{22}$	.0625
0 0	0 2	β_{22}^2	.0625

$G = .984375$

Aggregate state probabilities: $P(n_1,n_2)$

n_1	n_2	$P(n_1,n_2)$
2	0	.5625
0	2	.140625
1	1	.28125

Equivalent network $\beta_1 = \beta_{11} + \beta_{12} = 3/4$, $\beta_2 = \beta_{21} + \beta_{22} = 1/4$.

n_1	n_2	$f(n_1)\ f(n_2)$
2	0	.5625
0	2	.140625
1	1	.28125

4.4 The Effect of Service Time Patterns in CTG Queueing Models

In the preceding section we established that specific behavior patterns did not impact queueing network models in local balance. In this section we consider the effect of specific behavior patterns upon non-local balance models. The property violating local balance in the models we consider is that of allowing different service time distributions by customer type in FCFS queues. The practical motivation for considering this particular structure is the speculation that a job's service request distribution may vary according to its past history. Lo [L1] observed that CPU burst time distributions could be characterized as alternating between long-burst and short-burst modes. A possible interpretation of this behavior is that job service requests alternate between long and short modes. We can generalize this property to include both CPU and I/O requests. CPU service disciplines are commonly modeled as PS, while I/O service disciplines are commonly modeled as FCFS. In keeping with the previous section, we evaluate the impact of ctg's with specific sequences of requests to "equivalent" (if the queueing network was in local balance) ctg's which allow arbitrary sequences of requests.

In order to keep the ctg models tractable we must restrict our evaluation to small numbers of queues and job types.

4.4.1 The Models

The two ctg's evaluated are displayed in figure 4.3. In ctg I the customer changes type and resource cyclically. This is

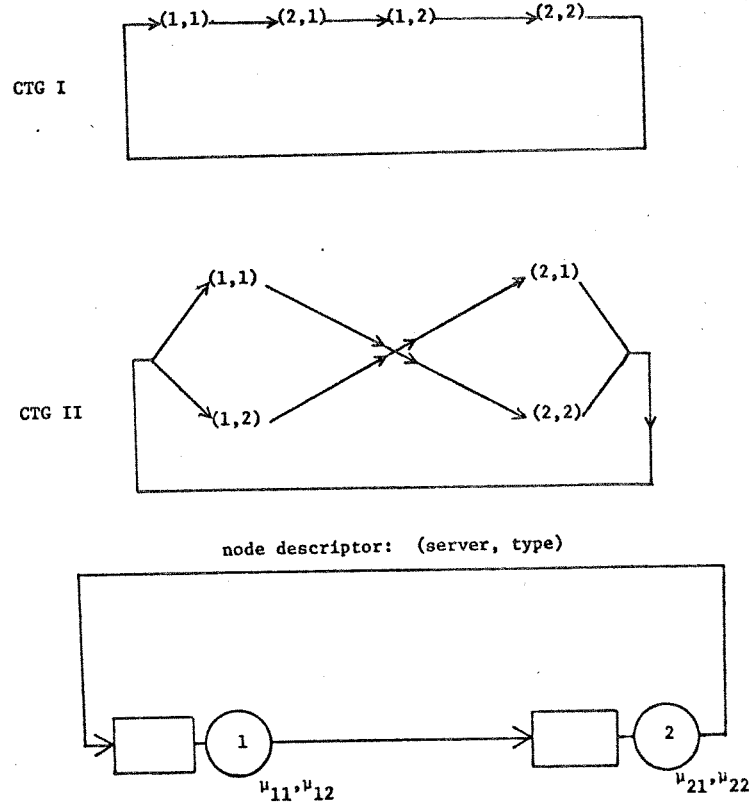


Figure 4.3: Two Customer Task Graph Models (A)

the "patterned" ctg. In ctg II the customer's type is chosen probabilistically while his resource requests are still cyclic. We note the queueing networks associated with these ctg's are identical. Referring to the notation of the previous section, let $\beta_{I,i}$ and $e_{I,i}$ be defined as the composites for server i in ctg I, and similarly in ctg II. Note from the structure of the ctg's that $e_{I,i}=1$ and $e_{II,i}=1/2$, i=1,2 upon inspection. Since the e are defined to within an arbitrary multiplicative constant, for ease of exposition allow $e_{II,i}$ to be redefined as 1 for all i and r. Thus $\beta_{I,i} = \beta_{II,i}$, i=1,2. If the two ctg queueing network models were in local balance they would be equivalent, and hence their performance characteristics would be identical.

The two ctg models are evaluated for different service disciplines (FCFS and PS), for a number of values for μ_{ir} (i=1,2 and r=1,2), and for different numbers of customers. Service times at each server are assumed exponential. For all three models the service discipline of server 2 is FCFS. For model 1 the service discipline of server 1 is FCFS while for models 2 and 3 it is PS.

4.4.2 Solution Method and Validation

The models were solved by an exact matrix method. The number of states for models 1, 2, and 3 are 12, 32, and 26, respectively. A property of model 2, ctg I is that the system is non-ergodic, the state space partitioning itself into two non-intersecting ergodic chains. The steady-state probabilities for states of the larger of the two chains were used in the comparison.

The solution method was validated for all models by setting $\mu_{i1}=\mu_{i2}$ for FCFS server i, bringing the model into local balance. Results were then compared against those obtained from ASQ [K1], a program for the solution of local balance models.

4.4.3 Results

A range of values for the mean services rates μ were chosen for the investigation of equivalence. With no loss of generality μ_{11} was set at 1. Tables 4.2-4.4 display the results of the investigation. An inspection of the tables reveals that the queue-length distributions of the two models usually agree to within 1%, with the greatest disagreement being 13%. Note that server utilization (1-P(2) for server 1 and 1-P(0) for server 2) agree in all cases within 13%.

Note that the queue-length distributions for ctg's I and II of model 1 appear identical. This identity was tested by coding the solution method used for additional precision for model 1. The queue-length distributions obtained for ctg's I and II were identical to within the inherent accuracy of the solution method (at least 10^{-20}).

The close agreement of the queue-length distributions for ctg I and ctg II over the range of parameters leads us to the conclusion that the two ctg models are nearly equivalent.

4.5 The Effect of Branching Patterns in CTG Models

In keeping with the preceding section we consider the impact of specific ctg patterns upon non-local balance models. Non-

TABLE 4.2: COMPARISON OF EQUIVALENCY
FOR MODEL 1, CTG I VS. CTG II

SERVICE DISCIPLINES: SERVER 1 - FCFS, SERVER 2 - FCFS
P(I): PROBABILITY OF I JOBS IN QUEUE 1
NUMBER OF CUSTOMERS: 2

CTG	μ_{11}	μ_{12}	μ_{21}	μ_{22}	P(0)	P(1)	P(2)
I	1.0	1.0	1.0	4.0	.216	.275	.510
II					.216	.275	.510
I	1.0	4.0	1.0	4.0	.371	.258	.371
II					.371	.258	.371
I	1.0	16.0	1.0	4.0	.443	.212	.345
II					.443	.212	.345
I	4.0	1.0	1.0	4.0	.371	.258	.371
II					.371	.258	.371
I	4.0	4.0	1.0	4.0	.649	.228	.123
II					.649	.228	.123
I	4.0	16.0	1.0	4.0	.768	.161	.071
II					.768	.161	.071
I	16.0	1.0	1.0	4.0	.443	.212	.345
II					.443	.212	.345
I	16.0	4.0	1.0	4.0	.768	.161	.071
II					.768	.161	.071
I	16.0	16.0	1.0	4.0	.901	.086	.013
II					.901	.086	.013
I	1.0	1.0	1.0	16.0	.201	.223	.576
II					.201	.223	.576
I	1.0	4.0	1.0	16.0	.345	.212	.443
II					.345	.212	.443
I	1.0	16.0	1.0	16.0	.410	.180	.410
II					.410	.180	.410
I	4.0	1.0	1.0	16.0	.345	.212	.443
II					.345	.212	.443
I	4.0	4.0	1.0	16.0	.619	.190	.190
II					.619	.190	.190
I	4.0	16.0	1.0	16.0	.741	.141	.118
II					.741	.141	.118
I	16.0	1.0	1.0	16.0	.410	.180	.410
II					.410	.180	.410
I	16.0	4.0	1.0	16.0	.741	.141	.118
II					.741	.141	.118
I	16.0	16.0	1.0	16.0	.886	.082	.032
II					.886	.082	.032

TABLE 4.3: COMPARISON OF EQUIVALENCY
FOR MODEL 2, CTG I VS. CTG II

SERVICE DISCIPLINES: SERVER 1 - PS, SERVER 2 - FCFS
P(I): PROBABILITY OF I JOBS IN QUEUE 1
NUMBER OF CUSTOMERS: 2

CTG	μ_{11}	μ_{12}	μ_{21}	μ_{22}	P(0)	P(1)	P(2)	P(3)
I	1.0	1.0	1.0	4.0	.127	.155	.264	.455
II					.135	.158	.247	.460
I	1.0	4.0	1.0	4.0	.284	.149	.284	.284
II					.302	.198	.198	.302
I	1.0	16.0	1.0	4.0	.362	.130	.259	.249
II					.385	.179	.159	.277
I	4.0	1.0	1.0	4.0	.284	.284	.149	.284
II					.302	.198	.198	.302
I	4.0	4.0	1.0	4.0	.620	.216	.115	.050
II					.623	.211	.108	.058
I	4.0	16.0	1.0	4.0	.755	.136	.087	.022
II					.757	.156	.059	.028
I	16.0	1.0	1.0	4.0	.362	.309	.081	.249
II					.385	.179	.159	.277
I	16.0	4.0	1.0	4.0	.755	.189	.034	.022
II					.757	.156	.059	.028
I	16.0	16.0	1.0	4.0	.900	.086	.013	.001
II					.900	.085	.013	.002
I	1.0	1.0	1.0	16.0	.113	.121	.237	.529
II					.126	.131	.203	.536
I	1.0	4.0	1.0	16.0	.249	.081	.309	.362
II					.277	.159	.179	.385
I	1.0	16.0	1.0	16.0	.317	.048	.317	.317
II					.352	.148	.148	.352
I	4.0	1.0	1.0	16.0	.249	.259	.130	.362
II					.277	.159	.179	.385
I	4.0	4.0	1.0	16.0	.574	.173	.160	.094
II					.587	.167	.123	.123
I	4.0	16.0	1.0	16.0	.719	.085	.154	.043
II					.726	.130	.077	.067
I	16.0	1.0	1.0	16.0	.317	.317	.048	.317
II					.352	.148	.148	.352
I	16.0	4.0	1.0	16.0	.719	.197	.042	.043
II					.726	.130	.077	.067
I	16.0	16.0	1.0	16.0	.883	.081	.031	.004
II					.884	.078	.028	.011

TABLE 4.4: COMPARISON OF EQUIVALENCY
FOR MODEL 3, CTG I VS. CTG II

SERVICE DISCIPLINES: SERVER 1 - PS, SERVER 2 - FCFS
P(I): PROBABILITY OF I JOBS IN QUEUE 1
NUMBER OF CUSTOMERS: 3

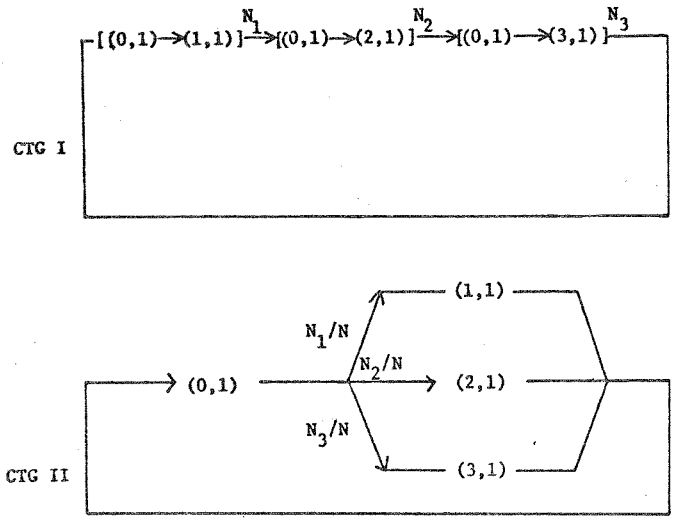
CTG	μ_{11}	μ_{12}	μ_{21}	μ_{22}	P(0)	P(1)	P(2)	P(3)
I	1.0	1.0	1.0	4.0	.134	.157	.250	.459
II					.135	.158	.247	.460
I	1.0	4.0	1.0	4.0	.260	.238	.243	.260
II					.274	.220	.231	.274
I	1.0	16.0	1.0	4.0	.309	.275	.228	.187
II					.332	.239	.215	.214
I	4.0	1.0	1.0	4.0	.284	.207	.224	.284
II					.274	.220	.231	.274
I	4.0	4.0	1.0	4.0	.623	.212	.109	.057
II					.623	.211	.108	.058
I	4.0	16.0	1.0	4.0	.753	.182	.051	.014
II					.755	.170	.056	.019
I	16.0	1.0	1.0	4.0	.348	.214	.205	.233
II					.332	.239	.215	.214
I	16.0	4.0	1.0	4.0	.756	.161	.060	.023
II					.755	.170	.056	.019
I	16.0	16.0	1.0	4.0	.900	.085	.013	.002
II					.900	.085	.013	.002
I	1.0	1.0	1.0	16.0	.124	.129	.211	.535
II					.126	.131	.208	.536
I	1.0	4.0	1.0	16.0	.228	.201	.228	.343
II					.249	.182	.208	.361
I	1.0	16.0	1.0	16.0	.258	.244	.240	.258
II					.294	.205	.208	.294
I	4.0	1.0	1.0	16.0	.262	.166	.199	.373
II					.249	.182	.208	.361
I	4.0	4.0	1.0	16.0	.585	.168	.128	.119
II					.587	.167	.123	.123
I	4.0	16.0	1.0	16.0	.717	.172	.075	.036
II					.721	.147	.078	.053
I	16.0	1.0	1.0	16.0	.318	.178	.187	.318
II					.294	.205	.208	.294
I	16.0	4.0	1.0	16.0	.724	.133	.081	.062
II					.721	.147	.078	.053
I	16.0	16.0	1.0	16.0	.884	.078	.028	.010
II					.884	.078	.028	.011

exponential service distributions in FCFS queues is the property violating local balance in these models. We investigate the effect of deterministic routing through the ctg queueing networks upon the models' equivalence.

The practical motivation for this ctg structure is the observation for some computer systems that disk requests may be clustered in time. One example is a large system in which data bases are tape resident. When a job requiring a data base which is not disk resident is to be run, the appropriate data base must be copied from tape to a single disk--resulting in a preponderant number of accesses to the disk clustered in time. When the data base must be copied back to tape the same behavior is observed. Clustering of device requests is also likely to be observed in data base systems in which data is dumped from disk to tape as part of a rollback/recovery scheme. Deterministic routing is used in the following models to enforce extreme examples of device request clustering.

4.5.1 The Models

The two ctg's evaluated are displayed in Figure 4.4. In ctg I the customer cycles through server 0 and server 1 N_1 times, then cycles through server 0 and server 2 N_2 times, etc. In ctg II the customer chooses server j with probability $N_j/(N_1+N_2+N_3)$, j=1,...,3, upon completion of service at server 0. Using the notation defined in section 4.4, it is intuitively obvious that $e_{I,i} = e_{II,i}$. Let the μ be constant over customer



$N = N_1 + N_2 + N_3$

$[(i,j)\rightarrow(k,\ell)]^n$ is defined as $(i,j)\rightarrow(k,\ell)_1\rightarrow(i,j)\rightarrow(k,\ell)_2\rightarrow\ldots\rightarrow(k,\ell)_n$

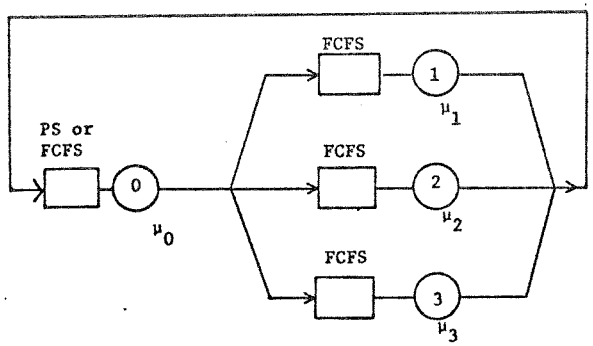


Figure 4.4: Two Customer Task Graph Models (B)

type. Thus the two ctg models would be equivalent if local balance held.

The two ctg models are evaluated for different values of N_i (i=1,...,3), μ_j(j=1,...,3), M (the number of customers), and C_v, the coefficient of variation of the service time distribution for servers 1, 2, and 3. Two values for C_v were chosen for evaluation, 0 and $1/\sqrt{3}$. C_v=0 defines a constant service time and C_v=$1/\sqrt{3}$ lies between 0 and 1, with C_v=1 being a property of the exponential distribution. We note that for C_v=1 the queueing network described is in local balance if its service times are exponential and the two ctg models are known to be equivalent. With no loss of generality μ_0 is set at 1. Servers 1-3 are FCFS while server 0 may be either PS or FCFS.

A central server form for the models was chosen so as to more easily relate the possible behavior patterns for actual systems to the investigation. Thus server 0 represents the CPU, with servers 1-3 representing I/O devices. In the following sections we shall parameterize the model according to assumptions drawn from experience with running computer systems.

4.5.2 Solution Methods, Validation, and an Equivalence Metric

The state space defined by the models was too large for matrix solution methods so a simulation, coded in ASPOL [C9], was used. A Students T test approach [G2] for confidence intervals was used, as the size of the state space made the preferable Crane-Iglehart methods [C13, C14] impractical. Each run iteration of the

simulation model was long enough to insure that all known distribution means (such as device holding times) were within 1% of the simulation input parameters. Values obtained from the simulation are estimated to be within $\pm$ 1% of the actual values with 90% confidence. For $C_v=1/\sqrt{3}$ the actual holding time distribution used is 3-stage Erlangian.

The service discipline for server 0 in the simulation is round-robin with fixed quantum (RRFQ). In the RRFQ discipline the server processes each service request for the duration of a fixed quantum. If the job's service request is complefed during this quantum, it leaves the server, otherwise the job is cycled back to the end of the queue to await its next quantum of service. New arrivals are placed at the end of the queue. The PS discipline is approximated, when appropriate for the models, by setting the quantum to be less (by a factor of 4) than the mean service request time (μ_0^{-1}) for the server. The FCFS discipline is approximated by setting the quantum larger (by a factor of 4) than μ_0^{-1}. Realism was the motivation for using the round-robin discipline for server 0. Actual CPU service disciplines for computer systems are better approximated by round-robin than PS or FCFS, as true PS is impossible to achieve and true FCFS is not desirable (from the results of Sherman, Baskett and Browne [S3]). The service disciplines for servers 1-3 are FCFS, in accordance with their role as I/O devices in the central server model.

The simulation was validated at $C_v=1$ against solutions

obtained via ASQ. Results were in agreement by $\pm$ 1% for all cases considered.

In contrast to the preceding sections, the equivalency metric used was not queue-length--the reason being the awkwardly large amount of data which would result and the greatly increased expense of data gathering in the simulation. An "aggregate" equivalency metric of server 1 utilization was instead chosen. This metric is reasonable since it is also the thruput of server 1 (since μ_0 is set at 1), and represents the work capacity of the system.

4.5.3 Model Parameters and Results

Table 4.5 displays the results of the comparison for different parameterizations of the models. For models 1-7 the discipline for server 0 is PS, for the others it is FCFS. ρ is defined as in chapter 2 and represents the balance between server 0 and the servers 1-3 subsystem. Values for μ_1, μ_2, and μ_3 were chosen to satisfy the relationship $\mu_2^{-1}=\mu_3^{-1}=3\mu_1^{-1}$ and to satisfy the definition of ρ. The factor 3 was selected as the balance between servers since that is typically the disparity in mean access times between disks and drums for accesses to single records. The values for N_1, N_2, and N_3 such that $N_1=N_2=N_3$ reflect equal average frequencies of visits to servers 1-3 (corresponding to $e_1=e_2=e_3$) while $N_1=3N_2=3N_3$ reflects an equal workload balance between servers 1-3 ($\beta_1=\beta_2=\beta_3$). An integer value for each N_i in the table represents a ctg I model, while a dash represents a ctg II model.

TABLE 4.5: Comparison of CTG Models with Branching Patterns

Model	M	μ_1^{-1}	μ_2^{-1}, μ_3^{-1}	C_v	ρ	N_1	N_2	N_3	Server 0 Utilization
1	4	.429	1.29	0	1	1	1	1	.98
						10	10	10	.97
						-	-	-	.97
2	4	.429	1.29	.58	1	1	1	1	.96
						10	10	10	.94
						-	-	-	.95
3	2	.858	2.58	0	2	1	1	1	.58
						10	10	10	.56
						-	-	-	.55
	4	.858	2.58	0	2	1	1	1	.84
						10	10	10	.80
						-	-	-	.79
4	2	.858	2.58	.58	2	1	1	1	.56
						10	10	10	.53
						-	-	-	.55
	4	.858	2.58	.58	2	1	1	1	.79
						10	10	10	.77
						-	-	-	.76
5	4	.556	1.67	.58	1	3	1	1	.97
						30	10	10	.95
						-	-	-	.96
6	2	1.12	3.36	.58	2	3	1	1	.58
						30	10	10	.57
						-	-	-	.56
	4	1.12	3.36	.58	2	3	1	1	.86
						30	10	10	.81
						-	-	-	.83
7	2	1.12	3.36	.58	2	3	1	1	.55
						30	10	10	.55
						-	-	-	.55
	4	1.12	3.36	.58	2	3	1	1	.80
						30	10	10	.79
						-	-	-	.79
8	4	.429	1.29	.58	1	1	1	1	.96
						10	10	10	.96
						-	-	-	.96

Model	M	μ_1^{-1}	μ_2^{-1}, μ_3^{-1}	C_v	ρ	N_1	N_2	N_3	Server 0 Utilization
9	2	.858	2.58	.58	2	1	1	1	.56
						10	10	10	.55
						-	-	-	.54
10	2	1.12	3.36	0	2	3	1	1	.56
						30	10	10	.56
						-	-	-	.56
	4	1.12	3.36	0	2	3	1	1	.82
						30	10	10	.80
						-	-	-	.79
11	2	1.12	3.36	.58	2	3	1	1	.59
						30	10	10	.57
						-	-	-	.56

For models 8-11 all queues are disciplined FCFS.

An inspection of the results reveals that the two ctg models are nearly equivalent, with disparities typically less than 2% and occasionally as great as 3%. It must be remembered that the values displayed in table 4.2 are approximate, and subject to simulation error within the described bounds.

4.6 Conclusions

The conclusion we draw is that specific active resource request patterns by customers have little or no impact upon the accuracy of queueing network models of computer systems. For one class of queueing network models we have proved that patterns play no role in determining the performance measures of the models. Experiments on a wider class of models saw that patterns had negligible impact upon the performance measures.

The practical significance of this conclusion is that the computer system analyst can ignore active resource request patterns over different workloads provided that: (1) the work ascribed to each active resource remains constant, and (2) no scheduling of service requests is made. Obviously, a scheduling scheme which could predict job resource requests by recognizing patterns in job behavior might result in significantly different system performances for different workloads.

Further research investigating the impact of service request patterns quickly diverges according to the nature of patterns and scheduling schemes considered.

V. Summary and Conclusions

This work makes contributions in the area of analytic modeling solution techniques, broadens existing understanding of the robustness of a class of analytical models of computer systems, and executes a case study of workload characterization and program behavior.

Chapter II extends central server models of computer systems to include a passive resource and develops two analytical approximation techniques for their solution. The first solution technique applies to a model assuming exponential service rates. It was discovered independently by a number of authors ([A4], [C10] and [K2]), including this one. The second, and newer, technique is significantly more accurate than the first and applies to the larger class of models assuming non-exponential service rates. The accuracy of the approximations was demonstrated by an extensive study utilizing simulation and (other) analytical methods over a realistic parameter space for computer system models. Our techniques are a step in the direction of more accurate analytical models of computer systems without sacrificing the chief virtues of easy parameterization and solution. Previous models including passive resources were either accurate and expensive (if not intractable) or inaccurate. Of great practical significance is the fact that the solution techniques are easily incorporated into widespread analytical tools (ASQ [K1], for example). The techniques are not limited in application to the Peripheral Processor model considered in the chapter but are applicable to more general queueing

network models in which the level of parallelism in any subsystem is limited. Thus the techniques are a contribution to queueing theory in general.

In chapter III we sought evidence of locality and patterning in file requests to determine if predictive scheduling would be advantageous. The investigation proceeded in two steps: first, an analysis of program behavior to determine the existence and the nature of the patterning; and second, a study to determine if the patterning could be advantageous to a predictive scheduling memory management scheme of the library file system.

Data for our study was extracted from an event trace taken from an actual system in operation. The performance of static and dynamic memory management strategies was studied by trace driven simulation models and an analytical model. Models of the reference generation process (an aspect of workload behavior) were constructed in order to test assumptions about the process. We found that assuming the process to be described by a Markov locality model, while unrealistic, was more accurate than assuming it to be described by an independent reference model. We concluded that despite the slight degree of locality this implied, a static allocation was, in general, superior in performance to a dynamic allocation strategy over the empirical trace for the given system.

This approach embodied two original contributions. First, this is the first investigation of locality in system file references. Previous investigations have been concerned with user file (page) references. Second, we consider the reference trace comprised of

actual system file names, not the address where they reside. The motivation for this approach is shared by other investigators ([M1], for example) who have studied symbolic (name of) array references in user programs; namely that the mapping of names to addresses by a compiler (in our case system memory manager) distorts what patterning there is. The impact of this investigation would have been profound and far-reaching if strong locality had been found in the system file reference trace. The degree of locality the system exhibits limits the impact to the recommendation of a different memory management scheme for the system.

Chapter IV investigates the consequences of specific active resource request sequences of customers in queueing network models. The restriction that requests be probabilistically sequenced is an often raised criticism of a class of queueing network models. We prove this criticism, while intuitive, to be unfounded. We show that the enforcing of any arbitrary sequence of active resource requests for the local balance models has no impact on the performance measures of the models. We conclude from analytical and simulation results that the impact on performance measures of sequencing is negligible for realistic non-local balance models of computer systems.

The significance of this counter-intuitive property is that it helps explain the robustness of simple models which ignore patterns in resource requests. The simple models retain the same performance measures (and are thus approximately equivalent to) models with arbitrarily sequenced requests, providing that the same long-term workload is placed on the resources. The practical impact of these

results to the computer systems analyst/designer is that he can ignore sequencing in the modeling of many systems, confident in the accuracy of "unsequenced" (and simpler) models.

An important area for further research is the generalization of passive resource models and solution techniques to include interrelating passive resources. Different queueing network models of separate passive resources now exist (this work, [T2], and [B6], for example). The possibility of fusing these disparate analytical models into a common model appears promising. To also provide inexpensive solutions to the model is a problem which will not be solved without considerable expense in both time and effort.

Appendix A: Proof of Norton's Theorem for the Central Server Model

The following proof is taken from Chandy, et al. [C3]. Only slightly modifying the notation in [C3] let there be L-1 I/O queues in the central server model. The queues are indexed 1 to L, with the CPU indexed 1. The service rate for the i^{th} queue when there are k customers in the queue is $U_i(k)$, where i=1,...,M, and k=1,...,M. We assume the service times are exponentially distributed. Let p_i be the branching probability from the CPU to I/O queue i, $i \neq 1$. The states of the system are L-tuples, $(n_1,\ldots,n_L)$, where n_i is the number of customers in queue i, including the customer being served. We note that the n_i are non-negative integers and $n_1+\ldots+n_L = M$, the total number of customers. Let $P(n_1,\ldots,n_L)$ be the steady-state probability that the system is in state $(n_1,\ldots,n_L)$. From Gordon and Newell [G1]

$$P(n_1,\ldots,n_L) = g(n_1,\ldots,n_L)/G \tag{1}$$

$$g(n_1,\ldots,n_L) = \begin{cases} \prod_{i=1}^{L} x_i(n_i) & \text{for any feasible state} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

and

$$G = \sum_{\substack{\text{all} \\ \text{feasible} \\ \text{states}}} g(n_1,\ldots,n_L), \text{ a normalization constant.} \tag{3}$$

We define x_i recursively:

$$x_i(0) = 1,\ x_i(k) = x_i(k-1)\cdot y_i/U_i(k) \tag{4}$$

where k=1,...,M and i=1,...,L. The y_i are defined by the structure of the central server model as

$$y_i = \begin{cases} 1 & i=1 \\ p_i & i=2,\ldots,L \end{cases}. \qquad (5)$$

By reviewing the computational techniques of Buzen [B8], we can define a notation which greatly simplifies the proof. Let a "convolution" between two (R+1)-dimensional vectors, $\underline{A} = [a(0),a(1),\ldots,a(R)]$ and $\underline{B} = [b(0),b(1),\ldots,b(R)]$, be the (R+1)-dimensional vector $\underline{C} = [c(0),c(1),\ldots,C(R)]$, where $C(i) = \sum_{j=0}^{R} a(j)\cdot b(R-j)$ and denote this operation by * , so $\underline{C} = \underline{A}*\underline{B}$.

Applying this definition to the problem, we define the (L+1)-dimensional vector $\underline{X}_i = [x_i(0),\ldots,x_i(L)]$, with the x_i defined by (4). Define L+1 vectors $\underline{G}_0, \underline{G}_1,\ldots,\underline{G}_L$ each of dimension M+1:

$$\underline{G}_0 = (1,0,0,\ldots,0) \text{ and}$$

$$\underline{G}_i = \underline{X}_1*\underline{X}_2*\ldots*\underline{X}_i, \quad i=1,\ldots,M. \qquad (6)$$

Thus

$$\underline{G}_i = \underline{G}_{i-1}*\underline{X}_i \text{ for } i=1,\ldots,M. \qquad (7)$$

Let $G_i(r)$ be the r^{th} element of $\underline{G}_i$, r=0,...,L.

From the definition of convolution

$$G_i(r) = \sum_{R} g(n_1,\ldots,n_i,0,\ldots,0), \qquad (8)$$

where $R = \{n_1+\ldots+n_i = r\}$.

From this definition, G of (3) is equal to $G_L(M)$.

Now let $P_i(n)$ be the marginal probability distribution that there are n customers in queue i. Restricting our attention to queue L, we note that

$$P(n_1,\ldots,n_{L-1},n) = g(n_1,\ldots,n_{L-1},0)\cdot X_m(n)/G. \qquad (9)$$

Summing over the set $\{n_1+\ldots+n_{L-1} = M-n\}$

$$P_L(n) = X_L(n)\cdot G_{L-1}(M-n)/G \qquad (10)$$

for n=0,...,M. Note that the marginal probability for any queue i can be obtained by reordering the indexing so that queue i is re-indexed L.

Finally, let T_i be the thruput of queue i, that is, the rate at which customers are serviced and leave queue i. Again restricting our attention to queue M:

$$T_L = \sum_{n=1}^{M} P_L(n)\cdot U_L(n). \qquad (11)$$

From equation (4)

$$X_L(n)\cdot U_L(n) = X_L(n-1)\cdot y_L, \quad n=1,\ldots,M. \qquad (12)$$

From equations (10), (11) and (12)

$$T_L = y_L\cdot \sum_{n=1}^{M} x_M(n-1)\cdot G_{L-1}(M-n)/G$$

$$= y_L\cdot G_L(M-1)/G. \qquad (13)$$

Norton's Theorem for a Closed Network

Consider the above network of L queues. We first determine the queue-length distribution for queue L as a function of its service rate. We next construct an equivalent network consisting of queue L and a composite queue with rate T(n), where n is the number of customers in the composite queue, n=0,...,M. Set T(n) equal to the thruput of queue L when there are n customers in the network and the service time for queue L is set to zero. Let $P_L'(n)$ be the marginal probability of the queue length L in the equivalent network.

Theorem. The queue length distribution for queue L in the equivalent network is the same as in the given network. Or,

$$P_L'(n) = P(n) \text{ for } n=0, 1,\ldots,M. \tag{14}$$

Proof. From equations (1), (2), and (3) it follows that $P_L'(n)$ of M-n customers in the composite queue and n customers in queue L in the equivalent network is proportional to

$$\prod_{j=1}^{n} U_L^{-1}(j) \prod_{k=1}^{M-n} T(k)^{-1} \tag{15}$$

for a detailed derivation see [C3]. When U_L is set to $+\infty$ we see $x_L(n)=1$ for n=0 and $x_L(n)=0$ for $n\neq 0$; thus

$$T(M) = y_L \cdot G_{L-1}(M-1)/G_{L-1}(M). \tag{16}$$

Similarly,

$$T(n) = y_L \cdot G_{L-1}(n-1)/G_{L-1}(n),\ n=1,\ldots,M. \tag{17}$$

Substituting (17) into (15) we find $P_L(n)$ is directly proportional to

$$\frac{X_L(n)}{y_L^M} \cdot \frac{G_{L-1}(M-n)}{G_{L-1}(0)} \text{ for } n=0,1,\ldots,M. \tag{18}$$

Hence,

$$P_L'(n) \alpha X_L(n) \cdot G_{L-1}(M-n) \text{ for } n=0,1,\ldots,M. \tag{19}$$

However, from equation (10)

$$P_L(n) \alpha X_L(n) \cdot G_{L-1}(m-n) \text{ for } n=0,1,\ldots,M. \tag{20}$$

So,

$$P_L'(n) = P_L(n) \text{ for } n=0,1,\ldots,M. \tag{21}$$

Clearly from the theorem and equation (11) $T_L = T_L'$, and similarly the other performance metrics of queue L for both networks are identical.

Appendix B: The Number of States of the Model of Section 2.5.4

We wish to derive the number of states of the model of section 2.5.4. Let L be the number of I/O queues in the L+1 queue network. The I/O queues are assumed FCFS. Let M be the total number of customers (degree of multiprogramming) and let N be the number of PP's available (the level of parallelism in the I/O subsystem). Furthermore, let the service distribution in each I/O queue be represented by a network of T exponential stages. The structure of the network of service stages (hyperexponential, general Erlang, etc.) does not impact the total number of states of the system and so will not be considered. The CPU service rate is assumed exponential.

A state of the system is the (2L+1)-tuple $(n_0, n_1, s_1, n_2, s_2, \ldots, n_L, s_L)$ where n_0 is the number of customers in the CPU, n_i is the number of customers in queue i, and s_i is the index of the stage of service ($1 \leq s_i \leq T$) for the customer in service at queue i. For $n_i=0$, s_i is arbitrarily defined as zero. We wish to find the number of all combinations of members of the (2L+1)-tuple such that $0 \leq n_i \leq m$, $n_0+\ldots+n_L = m$, and $1 \leq s_i \leq T$ (for $n_i > 0$). We note that for T=1, $s_i=1$, i=1,...,L. The total number of states for this case is $\binom{M+L}{M}$, as the total number of ways to distribute M customers among L queues is $\binom{M+L-1}{M}$.

Let us approach the problem by considering the aggregate state descriptor for L I/O queues $(n_1,\ldots,n_L)$, ignoring the number of customers in the CPU and the stage of service in the I/O queues. We know the number of aggregate states to be $\binom{j+L-1}{j}$ for j customers attempting service in the L I/O queue network. We note for $j > N$ there are N customers queued in the I/O subsystem, and j-N customers waiting to be queued. From each aggregate state can be inferred the number of states which comprise the aggregate, namely $T^{(L-i)}$, where is is the count of the n_i such that $n_i=0$, i=1,...,L.

Define S(i,j) as the number of ways of distributing j jobs among L queues with exactly i queues empty. Thus for exactly i empty queues there are L-i queues with one or more customers. Now there are j-(L-i) customers to be allocated among the L-i queues which already contain exactly one customer so

$$S(i,j) = \binom{L}{i}\binom{j-1}{j-L+1} \quad 1 \leq j \leq N$$

$$= \binom{L}{i}\binom{j-1}{N-L+1} \quad N \leq j \leq M,$$

by the constraint that for $N \leq j \leq M$ the number of customers in the queues is N. Summing S(i,j) over j

$$S(i) = \sum_{j=L-i}^{n-1} \binom{L}{i}\binom{j-1}{j-L+1} + \binom{L}{i}\binom{N-1}{N-L+1} \quad (M-N+1).$$

The index j is initialized at L-i because there must be at least L-i customers in the L I/O queues to satisfy the constraint of exactly i empty queues.

Now define R(i) as the number of states of the system with i empty I/O queues comprising the aggregate state. By our previous discussion

$$R(i) = T^{(L-i)}S(i).$$

Note that there is additionally one state where all customers are in the CPU (j=0). So the total number of states is

$$R = 1 + \sum_{i=0}^{L-i} \binom{L}{i} T^{(L-i)} \left[\sum_{j=L-i}^{N-1} \binom{j-1}{j-L+1} + (M - N + 1) \binom{N-1}{N-L+1} \right].$$

Appendix C: Trace Tapes

Table C.1: Environment

TAPE	Tape1	Tape2	Tape3	Tape4
DATE	10/18/74	05/21/75	08/06/75	11/11/75
TIME	15.34.69	15.03.18	15.56.42	22.02.13
MACHINE	6400	6400	6400	6400

Duration of the traces was approximately 30 minutes.

TRACE	TAPE	ENVIRONMENT	REDUCTION
Tape1	Tape1	interactive	multistream
Tape2a	Tape2	interactive	monostream
Tape2b	Tape2	interactive	monostream
Tape3a	Tape3	interactive	monostream
Tape3b	Tape3	interactive	monostream
Tape4	Tape4	batch	multistream
Tape4a	Tape4	batch	monostream
Tape5	composite of tape2a, tape2b, tape3a, and tape3b.		

TABLE C.2: PROGRAM REFERENCES FOR TAPE1 AND TAPE5

	TAPE1			TAPE5	
NAME	SIZE	FREQUENCY	NAME	SIZE	FREQUENCY
CIO	124	1640	CIO	133	33373
2RD	82	529	2CT	470	15118
2CT	470	400	2RD	83	7351
2WD	85	388	2WD	86	6702
1RJ	230	383	1TM	382	3155
1SJ	183	375	1AJ	157	2669
5DG	39	289	2WM	193	2659
CPU	270	268	2PD	183	2258
1DB	153	248	5DG	38	2215
PFM	258	213	5DB	36	2182
2WM	191	198	OPE	72	1751
OPE	72	195	2TS	333	1551
2PD	179	186	PFM	268	1314
5DB	36	159	SNP	55	1239
2TS	351	153	MSG	38	1195
1TM	382	141	2DF	31	924
1AJ	166	136	RFL	110	916
MSG	38	122	CPU	275	789
SIS	166	102	2PU	215	751
RFL	120	78	RTA	52	447
PCC	45	74	LDR	191	234
3EA	187	67	1SR	146	195
SNP	55	51	RCC	39	146
2DF	31	44	3AJ	104	145
2PU	266	42	1RJ	232	137
LDR	199	27	2EF	142	130
1SS	282	20	3CT	39	88
2EF	122	10	PCC	45	72
RCC	39	9	RSF	217	60
3AJ	101	8	1SJ	183	45
5DE	35	6	ERR	289	16
ERR	439	4	2MT	458	12
1CJ	67	3	2JE	21	7
1SR	136	3	1PS	138	7
2JE	15	1	CLO	12	4
1PS	138	1	DMP	359	3
			3EA	187	1
TOTAL NUMBER OF PROGRAMS		36			37
TOTAL SIZE OF PROGRAMS		5752			6012
NUMBER OF REFERENCES		6575			89861

TABLE C.3: PROGRAM REFERENCES FOR TAPE2

	TAPE2A			TAPE2B	
NAME	SIZE	FREQUENCY	NAME	SIZE	FREQUENCY
CIO	133	4165	CIO	133	11407
2RD	83	1274	2CT	470	6026
2CT	470	1203	2RD	83	2159
2WD	86	1197	2WD	86	1886
5DB	36	600	1TM	382	1206
5DG	38	550	2WM	193	1025
1AJ	157	401	1AJ	157	1003
1TM	382	330	2PD	183	711
2WM	193	324	5DG	38	675
OPE	72	288	5DB	36	674
SNP	55	273	OPE	72	594
2PD	183	264	2TS	333	574
2TS	333	262	PFM	268	496
PFM	268	208	MSG	38	469
2PU	215	180	SNP	55	322
CPU	275	180	RFL	110	291
RFL	110	162	2DF	31	280
2DF	31	131	CPU	275	235
MSG	38	119	2PU	215	213
RTA	52	74	RTA	52	188
RCC	39	41	LDR	191	82
1SR	146	28	1SR	146	74
LDR	191	24	3AJ	104	68
1RJ	232	18	2EF	142	46
2EF	142	15	RCC	39	45
3AJ	104	13	1RJ	232	41
PCC	45	5	PCC	45	25
ERR	289	5	RSF	217	22
RSF	217	5	1SJ	183	19
1SJ	183	3	ERR	289	6
DMP	359	1	2JE	21	5
3CT	39	1	1PS	138	5
			CLO	12	3
TOTAL NUMBER OF PROGRAMS		32			33
TOTAL SIZE OF PROGRAMS		5196			4969
TOTAL NUMBER OF REFERENCES		12344			30875

TABLE C.4: PROGRAM REFERENCES FOR TAPE3

TAPE3A			TAPE3B		
NAME	SIZE	FREQUENCY	NAME	SIZE	FREQUENCY
CIO	133	8282	CIO	133	9519
2CT	470	3232	2CT	470	4657
2RD	83	1904	2RD	83	2014
2WD	86	1800	2WD	86	1819
1TM	382	1000	1TM	382	619
2WM	193	784	2PD	183	540
1AJ	157	775	2WM	193	526
2PD	183	743	1AJ	157	490
5DG	38	653	5DB	36	456
OPE	72	581	5DG	38	337
5DB	36	452	2TS	333	291
2TS	333	424	OPE	72	288
PFM	268	416	SNP	55	252
SNP	55	392	MSG	38	227
MSG	38	380	RFL	110	226
2DF	31	290	2DF	31	223
RFL	110	237	PFM	268	194
CPU	275	227	CPU	275	147
2PU	215	217	2PU	215	141
RTA	52	147	3CT	39	83
LDR	191	84	LDR	191	44
1SR	146	50	1RJ	232	44
RCC	39	34	1SR	146	43
1RJ	232	34	3AJ	104	42
2EF	142	30	2EF	142	39
PCC	45	28	RTA	52	38
3AJ	104	22	RCC	39	26
RSF	217	9	RSF	217	24
1SJ	183	9	PCC	45	14
3CT	39	4	1SJ	183	14
ERR	289	3	2MT	458	12
CLO	12	1	DMP	359	2
2JE	21	1	ERR	289	2
1PS	138	1	2JE	21	1
			1PS	138	1
			3EA	187	1
TOTAL NUMBER OF PROGRAMS		34			36
TOTAL SIZE OF PROGRAMS		5008			6000
NUMBER OF REFERENCES		23246			23396

TABLE C.5: PROGRAM REFERENCES FOR TAPE4

TAPE4A			TAPE4			TAPE4(CONTD.)		
NAME	SIZE	FREQ.	NAME	SIZE	FEQ.	NAME	SIZE	FREQ.
CIO	133	9574	CIO	133	22772	RSF	217	106
2MT	458	4384	2RD	83	10796	1SR	146	101
2WD	86	2351	2WD	86	033	RCC	39	98
2RD	83	2173	2MT	458	212	2AM	62	90
2WM	193	872	1DB	153	539	1TD	116	74
2PD	183	780	2PD	183	948	2SP	362	73
MSG	38	468	2WM	193	579	2TJ	26	54
1AJ	157	385	1PS	138	115	2JE	21	53
OPE	72	361	OPE	72	347	2E1	95	49
2TS	333	353	MSG	38	175	2E2	96	49
SNP	55	291	RFL	110	085	3EA	187	35
2DF	31	275	1AJ	157	032	2EF	142	26
RFL	110	270	2TS	333	979	90L	9	24
5DG	38	242	EPR	230	973	9C7	36	22
2PU	215	201	SNP	55	943	CLO	12	18
CPU	275	201	5DG	38	914	2PL	94	17
5DB	36	184	5DB	36	913	DMP	359	9
PFM	268	134	2DF	31	814	2SC	73	4
RTA	52	129	PFM	268	796	2DS	149	4
3AJ	104	116	PAT	93	510	20U	81	2
PCC	45	47	1PL	148	478	ERR	289	2
RCC	39	47	1SS	287	462	2FE	10	2
RSF	217	35	CPU	275	429	9C8	70	1
LDR	191	34	2PU	215	422	9DI	66	1
1SR	146	34	3AJ	104	332			
1RJ	232	34	1RJ	232	314			
1SJ	183	31	RTA	52	313			
2EF	142	11	1SJ	183	309			
3EA	187	7	LDR	191	184			
DMP	359	4	4MD	104	146			
ERR	289	1	PCC	45	124			
TOTAL NUMBER OF PROGRAMS		31						55
TOTAL SIZE OF PROGRAMS		4950						7481
NUMBER OF REFERENCES		24029						71903

Bibliography

[A1] J. W. Anderson, "Primitive Process Level Modeling and Simulation of a Multiprocessing Computer System," Ph.D. Thesis, University of Texas, Austin, Tx., May 1972.

[A2] S. Arora and A. Gallo, "The Optimal Organization of Multiprogrammed Multi-Level Memory," Proc. ACM Workshop on System Performance Evaluation, pp. 104-141, April 1971.

[A3] S. Arora and A. Gallo, "Optimization of Static Loading of Multilevel Memory Systems," J. ACM, Vol. 20, No. 2, pp. 307-319, April 1973.

[A4] B. Avi-Itzhak and D. P. Heyman, "Approximate Queueing Models for Multiprogrammed Computer Systems," Operations Research, Vol. 21, No. 6, pp. 1212-1231, 1973.

[B1] F. Baskett, 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, p. 248, April 1975.

[B2] A. P. Batson and R. E. Brundage, "Measurements of the Virtual Memory Demands of ALGOL-60 Programs," Proc. SIGMETRICS 74 Conf. Montreal - Performance Evaluation Review, 3, pp. 121-126, 1974.

[B3] A. Brandwajn, "A Model of a Time Sharing Virtual Memory System Solved Using Equivalence and Decomposition Methods," Acta Informatica, Vol. 4, pp. 11-47, 1974.

[B4] A. Brandwajn, J. P. Buzen, E. Gelenbe, and D. Potier, "A Model of Performance for Virtual Memory Systems," Proc. ACM SIGMETRICS Symp., September 1974.

[B5] R. S. Brice, "A Study of Feedback Coupled Resource Allocation Policies in a Multiprocessing Computer Environment," Ph.D. Thesis, University of Texas, Austin, Tx., August 1973.

[B6] R. M. Brown, "An Analytic Model of a Large Scale Interactive System Including the Effects of Finite Main Memory," M.A. Thesis, Dept. of Computer Sciences, Univ. of Texas, Austin, Tx., 1974.

[B7] J. C. Browne, K. M. Chandy, R. M. Brown, T. W. Keller, D. F. Towsley, and C. W. Dissly, "Hierarchical Techniques for the Development of Realistic Models of Complex Computer Systems," Proc. IEEE, Vol. 63, No. 6, pp. 966-975, June 1975.

[B8] J. P. Buzen, "Queueing Network Models of Multiprogramming," Ph.D. Thesis, Harvard University, Cambridge, Mass., 1971.

[B9] J. P. Buzen, "I/O Subsystem Architecture," Proc. IEEE, Vol. 63, No. 6, pp. 871-878, June 1975.

[C1] K. M. Chandy, "The Analysis and Solutions for General Queueing Networks," Proc. 6th Annual Princeton Conf. Information Sciences and Systems, Princeton Univ., Princeton, N. J., March 1972.

[C2] K. M. Chandy, T. W. Keller, and J. C. Browne, "Design Automation and Queueing Networks: An Interactive System for the Evaluation of Computer Queueing Models," Proc. Design Automation Workshop, June 1975.

[C3] K. M. Chandy, U. Herzog, and L. Woo, "Parametric Analysis of Queueing Network Models," IBM J. Res. Develop., Vol. 19, No. 1, p. 36, 1975.

[C4] K. M. Chandy, U. Herzog, and L. Woo, "Approximate Analysis of General Queueing Networks," IBM J. Res. Develop., Vol. 19, No. 1, p. 43, 1975.

[C5] K. M. Chandy, J. H. Howard, and D. F. Towsley, "Product Form and Local Balance in Queueing Networks," submitted J. ACM.

[C6] W. Chang and D. J. Wong, "Computer Channel Interference Analysis," IBM Sys. J., Vol. 4., No. 2, p. 162, 1965.

[C7] C. R. Chow, "On Optimization of Storage Hierarchies," IBM J. Res. Develop., Vol. 18, No. 3, pp. 194-203, May 1974.

[C8] E. G. Coffman and P. J. Denning, Operating Systems Theory, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1973.

[C9] Control Data Corporation, A Simulation Process-Oriented Language (ASPOL) Reference Manual, C.D.C. Special Support Division, Sunnyvale, Calif., 1972.

[C10] P. J. Courtois, "On the Near-Complete-Decomposability of Networks of Queues and of Stochastic Models of Multiprogramming Computing Systems," Computer Science Dept. Rep. CMU-CS-72-111, Carnegie-Mellon Univ., Pittsburgh, Pa., November 1971.

[C11] P. J. Courtois, "Decomposability, Instabilities, and Saturation in Multiprogramming Systems," Com. ACM, Vol. 18, No. 7, pp. 371-376, July 1975.

[C12] P. J. Courtois, "Error Analysis in Nearly Decomposable Stochastic Systems," Econometrica, Vol. 43, No. 4, p. 691, July 1975.

[C13] M. A. Crane and D. I. Iglehart, "Simulating Stable Stochastic Systems, I: General Multiserver Queues," J. ACM, Vol. 21, p. 103, 1974.

[C14] M. A. Crane and D. I. Iglehart, "Simulating Stable Stochastic Systems, II: Markov Chains," J. ACM, Vol. 21, p. 114, 1974.

[D1] P. J. Denning, "Virtual Memory," Computing Surveys, Vol. 2, No. 3, pp. 153-189, September 1970.

[D2] P. J. Denning and G. S. Graham, "Multiprogrammed Memory Management," Proc. IEEE, Vol. 63, No. 6, 924-939, June 1975.

[F1] J. H. Florkowski, "Extended Analytic Models for Systems Evaluation," IBM Poughkeepsie Lab. Technical Report TR 00.2549, August 1974.

[F2] D. V. Foster, "File Assignment in Memory Hierarchies," Ph.D. Thesis, Dept. of Computer Sciences, Univ. of Texas, Austin, Tx., 1974.

[G1] W. J. Gordon and G. F. Newell, "Closed Queueing Systems with Exponential Servers," Operations Research, Vol. 15, pp. 254-265, March 1967.

[G2] H. Greenberg, Integer Programming, Academic Press, New York, p. 99, 1971.

[H1] U. Herzog, L. Woo, and K. M. Chandy, "Solution of Queueing Problems by a Recursive Technique," IBM J. Res. Develop., Vol. 19, No. 3, pp. 295-300, May 1975.

[H2] J. H. Howard, Jr., "A Large-Scale Dual Operating System," Proc. ACM 1973 Annual Conf., pp. 242-248, 1973.

[H3] J. H. Howard and W. M. Wedel, "The UT-2 Operating System Event Recorder," TSN-37, Computation Center, University of Texas, Austin, Tx., February 1974.

[I1] Information Research Associates (IRA), "A Limiting Capability Analysis of the Cyber 70 ALS," Austin, Tx., 1973.

[J1] J. R. Jackson, "Jobshop-like Queueing Systems," Management Science, Vol. 10, pp. 131-142, 1963.

[K1] T. W. Keller, ASQ Manual, Dept. of Computer Sciences Report TR-27, University of Texas, Austin, Tx., 1973.

[K2] T. W. Keller and K. M. Chandy, "Computer Models with Constrained Parallel Processors," Proc. Sagamore Conference on Parallel Processing, 1974.

[K3] L. Kleinrock, Queueing Systems, Vol. I: Theory, John Wiley and Sons, Inc., New York, 1975.

[L1] T.-C. Lo, "Computer Aids in Computer Systems Analysis," Ph.D. Thesis, University of Texas, Austin, Tx., 1973.

[M1] A. W. Madison and A. P. Batson, "Characteristics of Program Localities," submitted to Comm. ACM, 1975.

[M2] R. R. Muntz, "Analytic Modeling of Interactive Systems," Proc. IEEE, Vol. 63, No. 6, pp. 946-953, June 1975.

[N1] A. S. Noetzel and J. R. Haley, "The Decomposition of a Multiprocessor Event Trace into User Program Resource Demand Patterns," TR-49, Department of Computer Sciences, University of Texas, Austin, Tx., May 1975.

[N2] A. S. Noetzel, "Measurements of Processing Repetition in Interactive Computation," TR-52, Department of Computer Sciences, University of Texas, Austin, Tx., October 1975.

[R1] C. V. Ramamoorthy and K. M. Chandy, "Optimization of Memory Hierarchies in Multiprogrammed Systems," J. ACM, Vol. 17, No. 3, pp. 426-445, July 1970.

[S1] C. H. Sauer, "Configuration of Computing Systems: An Approach Using Queueing Network Models," Ph.D. Thesis, University of Texas, Austin, Tx., 1975.

[S2] G. S. Shedler, "A Cyclic-Queue Model of a Paging Machine," IBM Thomas J. Watson Research Center Report RC 2814, Yorktown Heights, N.Y., 1970.

[S3] S. W. Sherman, F. Baskett, and J. C. Browne, "Trace Driven Modeling and Analysis of CPU Scheduling in a Multiprogramming Systems," Proc. of ACM Workshop on Performance Evaluation, pp. 173-199, Cambridge, Mass., April 1971

[S4] H. A. Simon and A. Ando, "Aggregation of Variables in Dynamic Systems," Econometrica, Vol. 29, pp. 111-138, 1961.

[S5] A. J. Smith, "Analysis of a Locality Model for Disk Reference Patterns," available from the author at Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Calif., 1975.

[S6] J. Staudhammer, C. A. Combs, and G. Wilkinson, "Analysis of Computer Peripheral Interference," Proc. ACM National Meeting, p. 97, 1967.

[T1] J. E. Thornton, Design of a Computer: The CDC 6600, Scott, Foresman and Co., Glenview, Ill., 1970.

[T2] D. F. Towsley, "Local Balance Models of Computer Systems," Ph.D. Thesis, Dept. of Computer Sciences, University of Texas, Austin, Tx., 1975.

[W1] V. L. Wallace and R. S. Rosenberg, "Markovian Models and Numerical Analysis of Computer System Behavior," Proc. Spring Joint Computer Conference (AFIPS), Vol. 28, Spartan Books, N. Y., 1966.

[W2] T. R. Willemain, "Approximate Analysis of a Hierarchical Queueing Network," Operations Research, Vol. 22, No. 3, p. 522-545, 1974.

[W3] A. C. Williams and R. Bhandiwad, private communication.

TABLE 2.5 - ERROR OF APPROXIMATIONS FOR MODELS 81-120

U.EXACT : EXACT CPU UTILIZATION
U.NT : CPU UTILIZATION FOR N=M, LOCAL BALANCE
U.LB : CPU UTILIZATION FOR LOCAL BALANCE APPROXIMATION
U.AP : CPU UTILIZATION FOR NON-LOCAL BALANCE APPROXIMATION
ERR : (U - U.EXACT)*100
COVAR : ESTIMATED COEFFICIENT OF VARIATION

MODEL	RHO	ALPHA	BETA	M	N	T	U.EXACT	U.NT	ERR.NT	U.LB	ERR.LB	U.AP	ERR.AP	COVAR
81	.5	.5	.1	10	4	1	.687	.733	4.59	.715	2.78	.693	.53	1.76
82	.5	.5	1.0	2	1	1	.429	.491	6.23	.429	0.00	.429	0.00	1.00
83	.5	.5	1.0	3	1	1	.467	.579	11.23	.467	0.00	.467	0.00	1.00
84	.5	.5	1.0	3	2	1	.555	.579	2.40	.558	.31	.558	.31	1.00
85	.5	.5	1.0	5	1	1	.492	.668	17.58	.492	0.00	.492	0.00	1.00
86	.5	.5	1.0	5	2	1	.608	.668	5.95	.612	.38	.612	.38	1.00
87	.5	.5	1.0	5	4	1	.662	.668	.61	.664	.24	.664	.24	1.00
88	.5	.5	1.0	10	1	1	.500	.733	23.35	.500	0.00	.500	0.00	1.00
89	.5	.5	1.0	10	2	1	.639	.733	9.48	.640	.12	.640	.12	1.00
90	.5	.5	1.0	10	4	1	.711	.733	2.24	.715	.44	.715	.44	1.00
91	.5	1.0	.1	2	1	1	.410	.500	9.02	.429	1.88	.410	0.00	2.25
92	.5	1.0	.1	3	1	1	.444	.600	15.62	.467	2.29	.444	0.00	2.25
93	.5	1.0	.1	3	2	1	.568	.600	4.01	.571	1.15	.548	-1.17	1.62
94	.5	1.0	.1	5	1	1	.475	.714	23.96	.492	1.74	.475	0.00	2.25
95	.5	1.0	.1	5	2	1	.610	.714	10.48	.630	1.99	.596	-1.47	1.94
96	.5	1.0	.1	5	4	1	.701	.714	1.29	.706	.45	.688	-1.35	1.31
97	.5	1.0	.1	10	1	1	.496	.833	33.77	.500	.41	.496	0.00	2.25
98	.5	1.0	.1	10	2	1	.649	.833	18.42	.662	1.31	.648	-.91	2.11
99	.5	1.0	.1	10	4	1	.762	.833	7.12	.777	1.45	.740	-2.24	1.83
100	.5	1.0	1.0	2	1	1	.429	.500	7.14	.429	0.00	.429	0.00	1.00
101	.5	1.0	1.0	3	1	1	.467	.600	13.33	.467	0.00	.467	0.00	1.00
102	.5	1.0	1.0	3	2	1	.569	.600	3.14	.571	.28	.571	.28	1.00
103	.5	1.0	1.0	5	1	1	.492	.714	22.22	.492	0.00	.492	0.00	1.00
104	.5	1.0	1.0	5	2	1	.626	.714	8.79	.630	.37	.630	.37	1.00
105	.5	1.0	1.0	5	4	1	.704	.714	1.81	.706	.17	.706	.17	1.00
106	.5	1.0	1.0	10	1	1	.500	.833	33.36	.500	0.00	.500	0.00	1.00
107	.5	1.0	1.0	10	2	1	.661	.833	17.24	.662	.14	.662	.14	1.00
108	.5	1.0	1.0	10	4	1	.772	.833	6.12	.777	.44	.777	.44	1.00
109	1.0	.1	.1	2	1	1	.625	.686	6.07	.667	4.19	.643	1.80	1.53
110	1.0	.1	.1	3	1	1	.695	.778	8.28	.750	5.53	.714	1.95	1.53
111	1.0	.1	.1	3	2	1	.746	.778	3.11	.776	2.97	.756	.97	1.27
112	1.0	.1	.1	5	1	1	.777	.867	9.08	.833	5.68	.790	1.39	1.53
113	1.0	.1	.1	5	2	1	.815	.867	5.28	.865	5.03	.838	1.59	1.40
114	1.0	.1	.1	5	4	1	.854	.867	1.29	.867	1.28	.856	.11	1.13
115	1.0	.1	.1	10	1	1	.866	.945	7.98	.909	4.29	.872	.55	1.53
116	1.0	.1	.1	10	2	1	.896	.945	4.95	.942	4.67	.908	1.27	1.47
117	1.0	.1	.1	10	4	1	.914	.945	3.17	.945	3.16	.928	.65	1.35
118	1.0	.1	1.0	2	1	1	.667	.686	1.89	.667	0.00	.667	0.00	1.00
119	1.0	.1	1.0	3	1	1	.750	.778	3.75	.750	0.00	.750	0.00	1.00
120	1.0	.1	1.0	3	2	1	.773	.778	.41	.776	.27	.776	.27	1.00

TABLE 2.6 - ERROR OF APPROXIMATIONS FOR MODELS 120-162

U.EXACT : EXACT CPU UTILIZATION
U.NT : CPU UTILIZATION FOR N=M, LOCAL BALANCE
U.LB : CPU UTILIZATION FOR LOCAL BALANCE APPROXIMATION
U.AP : CPU UTILIZATION FOR NON-LOCAL BALANCE APPROXIMATION
ERR : (U - U.EXACT)*100
COVAR : ESTIMATED COEFFICIENT OF VARIATION

MODEL	RHO	ALPHA	BETA	M	N	T	U.EXACT	U.NT	ERR.NT	U.LB	ERR.LB	U.AP	ERR.AP	COVAR
121	1.0	.1	1.0	5	1	1	.833	.867	3.40	.833	0.00	.833	0.00	1.00
122	1.0	.1	1.0	5	2	1	.860	.867	.73	.865	.47	.865	.47	1.00
123	1.0	.1	1.0	5	4	1	.866	.867	.14	.867	.14	.867	.14	1.00
124	1.0	.1	1.0	10	1	1	.909	.945	3.61	.909	0.00	.909	0.00	1.00
125	1.0	.1	1.0	10	2	1	.938	.945	.69	.942	.42	.942	.42	1.00
126	1.0	.1	1.0	10	4	1	.941	.945	.39	.945	.38	.945	.38	1.00
127	1.0	.5	.1	2	1	1	.614	.720	10.60	.667	5.26	.627	1.26	2.14
128	1.0	.5	.1	3	1	1	.670	.833	16.30	.750	7.97	.687	1.69	2.14
129	1.0	.5	.1	3	2	1	.787	.833	4.61	.821	3.39	.781	-.64	1.57
130	1.0	.5	.1	5	1	1	.736	.935	19.87	.833	9.72	.752	1.58	2.14
131	1.0	.5	.1	5	2	1	.842	.935	9.32	.913	7.17	.844	.19	1.86
132	1.0	.5	.1	5	4	1	.921	.935	1.38	.934	1.27	.911	-1.03	1.29
133	1.0	.5	.1	10	1	1	.821	.992	17.13	.909	8.83	.829	.84	2.14
134	1.0	.5	.1	10	2	1	.903	.992	8.89	.980	7.64	.915	1.18	2.02
135	1.0	.5	.1	10	4	1	.955	.992	3.73	.991	3.60	.954	-.06	1.76
136	1.0	.5	1.0	2	1	1	.667	.720	5.33	.667	0.00	.667	0.00	1.00
137	1.0	.5	1.0	3	1	1	.750	.833	8.33	.750	0.00	.750	0.00	1.00
138	1.0	.5	1.0	3	2	1	.817	.833	1.65	.821	.44	.821	.44	1.00
139	1.0	.5	1.0	5	1	1	.833	.935	10.14	.833	0.00	.833	0.00	1.00
140	1.0	.5	1.0	5	2	1	.906	.935	2.88	.913	.73	.913	.73	1.00
141	1.0	.5	1.0	5	4	1	.931	.935	.35	.934	.24	.934	.24	1.00
142	1.0	.5	1.0	10	1	1	.909	.992	8.31	.909	0.00	.909	0.00	1.00
143	1.0	.5	1.0	10	2	1	.975	.992	1.74	.980	.49	.980	.49	1.00
144	1.0	.5	1.0	10	4	1	.986	.992	.58	.991	.45	.991	.45	1.00
145	1.0	1.0	.1	2	1	1	.625	.727	10.25	.667	6.19	.625	0.00	2.25
146	1.0	1.0	.1	3	1	1	.684	.846	16.23	.750	6.61	.684	0.00	2.25
147	1.0	1.0	.1	3	2	1	.809	.846	3.68	.830	2.08	.787	-2.23	1.62
148	1.0	1.0	.1	5	1	1	.747	.950	20.32	.833	8.65	.747	0.00	2.25
149	1.0	1.0	.1	5	2	1	.873	.950	7.66	.922	4.85	.848	-2.58	1.94
150	1.0	1.0	.1	5	4	1	.943	.950	.70	.948	.50	.925	-1.81	1.31
151	1.0	1.0	.1	10	1	1	.823	.997	17.45	.909	8.63	.823	0.00	2.25
152	1.0	1.0	.1	10	2	1	.926	.997	7.11	.984	5.81	.917	-.88	2.11
153	1.0	1.0	.1	10	4	1	.979	.997	1.80	.995	1.61	.964	-1.54	1.83
154	1.0	1.0	1.0	2	1	1	.667	.727	6.06	.667	0.00	.667	0.00	1.00
155	1.0	1.0	1.0	3	1	1	.750	.846	9.62	.750	0.00	.750	0.00	1.00
156	1.0	1.0	1.0	3	2	1	.826	.846	2.01	.830	.41	.830	.41	1.00
157	1.0	1.0	1.0	5	1	1	.833	.950	11.67	.833	0.00	.833	0.00	1.00
158	1.0	1.0	1.0	5	2	1	.915	.950	3.49	.922	.67	.922	.67	1.00
159	1.0	1.0	1.0	5	4	1	.946	.950	.37	.948	.17	.948	.17	1.00
160	1.0	1.0	1.0	10	1	1	.909	.997	8.82	.909	0.00	.909	0.00	1.00
161	1.0	1.0	1.0	10	2	1	.980	.997	1.71	.984	.48	.984	.48	1.00
162	1.0	1.0	1.0	10	4	1	.993	.997	.42	.995	.23	.995	.23	1.00

TABLE 8.7 - ERROR OF APPROXIMATIONS FOR MODELS 163-206

U.EXACT : EXACT CPU UTILIZATION
U.NT : CPU UTILIZATION FOR N=M, LOCAL BALANCE
U.LB : CPU UTILIZATION FOR LOCAL BALANCE APPROXIMATION
U.AP : CPU UTILIZATION FOR NON-LOCAL BALANCE APPROXIMATION
U.BND : UPPER BOUND ON CPU UTILIZATION
ERR : (U - U.EXACT)*100
COVAR : ESTIMATED COEFFICIENT OF VARIATION

MODEL	RHO	ALPHA	BETA	M	T	U.EXACT	U.NT	ERR.NT	U.LB	ERR.LB	U.AP	ERR.AP	U.BND	ERR.BND	COVAR
163	.5	1.	1.	2	2	.413	.430	3.72	.429	1.60	.423	1.84	.442	8.94	1.23
164	.5	1.	1.	3	3	.413	.450	9.47	.429	1.35	.426	1.07	.451	3.62	1.11
165	.5	1.	1.	4	4	.445	.500	5.44	.467	2.14	.461	1.54	.478	3.31	1.23
166	.5	1.	1.	5	5	.456	.500	4.34	.467	1.84	.464	.75	.485	2.84	1.11
167	.5	1.	1.	6	6	.477	.536	5.91	.492	1.53	.489	1.19	.497	2.08	1.23
168	.5	1.	1.	7	7	.479	.536	5.73	.492	1.35	.491	1.19	.498	1.96	1.11
169	.5	1.	1.	8	8	.522	.536	1.40	.532	1.08	.530	.80	.539	1.70	1.10
170	.5	1.	1.	9	9	.517	.536	1.93	.532	1.53	.533	1.59	.541	2.49	.98
171	.5	1.	1.	10	10	.558	.600	4.15	.571	1.29	.559	.85	.591	3.27	1.29
172	.5	1.	1.	11	11	.568	.600	3.23	.571	.37	.564	-.33	.602	2.44	1.16
173	.5	1.	1.	12	12	.591	.714	12.36	.630	2.94	.607	1.62	.645	5.38	1.39
174	.5	1.	1.	13	11	.598	.714	11.67	.630	2.25	.612	1.44	.651	5.38	1.45
175	.5	1.	1.	14	14	.721	.714	-.62	.706	-1.46	.706	-1.46	.727	.66	1.00
176	.5	1.	1.	15	14	.735	.714	-2.89	.706	-3.93	.716	-1.96	.737	.13	.87
177	.5	1.	1.	16	16	.763	.833	7.00	.777	1.33	.756	-.76	.787	2.39	1.49
178	.5	1.	1.	17	15	.771	.833	6.25	.777	.58	.762	-1.89	.791	2.00	1.35
179	.5	1.	1.	18	18	.644	.686	4.16	.667	2.27	.655	1.11	.690	4.62	1.23
180	.5	1.	1.	19	17	.651	.686	3.47	.667	1.39	.661	1.01	.703	5.25	1.11
181	.5	1.	1.	20	20	.715	.778	6.27	.750	3.32	.733	1.84	.779	6.45	1.23
182	.5	1.	1.	21	18	.727	.778	5.10	.750	2.35	.742	1.53	.793	6.69	1.11
183	.5	1.	1.	22	22	.798	.867	6.89	.833	3.49	.814	1.59	.861	6.25	1.23
184	.5	1.	1.	23	19	.831	.867	3.56	.833	.18	.824	-.73	.872	4.08	1.11
185	.5	1.	1.	24	24	.830	.867	3.74	.865	3.49	.856	2.63	.893	6.28	1.18
186	.5	1.	1.	25	20	.849	.867	1.84	.865	1.59	.867	1.83	.904	5.58	.98
187	.5	1.	1.	26	26	.901	.945	4.43	.942	4.15	.931	2.98	.968	5.93	1.17
188	.5	1.	1.	27	21	.918	.945	2.72	.942	2.44	.939	2.12	.966	4.84	1.05
189	.5	1.	1.	28	28	.626	.720	9.44	.667	4.11	.635	.92	.698	6.46	1.79
190	.5	1.	1.	29	22	.633	.720	8.67	.667	3.34	.639	.54	.703	7.00	1.63
191	.5	1.	1.	30	30	.686	.833	14.76	.750	6.43	.701	1.54	.779	9.36	1.79
192	.5	1.	1.	31	23	.694	.833	13.97	.750	5.64	.708	1.40	.793	9.98	1.63
193	.5	1.	1.	32	32	.755	.833	17.99	.833	7.84	.773	1.76	.861	10.60	1.79
194	.5	1.	1.	33	24	.765	.935	16.94	.833	6.79	.782	1.62	.872	10.69	1.65
195	.5	1.	1.	34	34	.945	.935	-1.00	.934	-1.11	.936	-.92	.956	1.08	.98
196	.5	1.	1.	35	25	.944	.935	-.91	.934	-1.01	.946	-.19	.963	1.92	.85
197	.5	1.	1.	36	36	.913	.992	7.92	.980	6.67	.939	2.62	.998	7.71	1.67
198	.5	1.	1.	37	26	.930	.992	6.21	.980	4.96	.948	1.62	.993	6.27	1.53
199	.5	1.	1.	38	38	.816	.846	2.99	.830	1.39	.808	-.84	.859	4.24	1.29
200	.5	1.	1.	39	27	.817	.846	2.95	.830	1.35	.818	.10	.871	5.48	1.16
201	.5	1.	1.	40	40	.866	.950	8.44	.922	5.63	.873	.72	.946	8.03	1.59
202	.5	1.	1.	41	28	.885	.950	6.47	.922	3.66	.884	-.16	.954	6.92	1.45
203	.5	1.	1.	42	42	.964	.950	-1.79	.948	-1.99	.948	-1.99	.967	-.18	1.00
204	.5	1.	1.	43	29	.957	.950	-1.36	.948	-1.56	.957	-.64	.973	.95	.87
205	.5	1.	1.	44	44	1.000	.997	-.27	.995	-.46	.988	-1.98	.999	-1.49	1.49
206	.5	1.	1.	45	30	1.000	.997	-.27	.995	-.46	.986	-1.44	.999	-1.49	1.35

(1) $ERR_{nt} \in [-2.1\%, 16\%]$

(2) $ERR_{lb} \in [-2.9\%, 7.8\%]$

(3) $ERR_{ap} \in [-2\%, 3\%]$

(4) $ERR_{bnd} \in [-.1\%, 10.7\%]$

For the exponential I/O cases, U_{lb} is also the conjectured upper bound since the minimum estimated coefficient of variation for the upper bound (1.0) is also the coefficient of variation assumed in the local balance approximation. We see from the range of ERR_{lb} for the exponential I/O cases that our conjecture holds. We also note that the conjectured upper bound holds for the hypoexponential I/O cases, within the $\pm 0.5\%$ range of accuracy for simulation results with 90% confidence.

The relative accuracies of the different approximate models may be evaluated by a glance at the above error limits. The central server model in local balance, which ignores the passive resource entirely, is accurate within 34%. The local balance approximation is accurate within 10%, while the non-local balance approximation is accurate within 5%.

Let us discuss at this point those cases of the validation model for which the approximation methods are exact. As previously stated, for N=1 models the I/O subsystem reduces to a first-come-first-served queue with a network of exponential service stages. If the individual I/O distributions are exponential then the composite distribution is hyperexponential with a coefficient of variation which can be exactly determined by equation 2.2. The behavior of the

composite queue degenerates further if $\beta=1$ ($\lambda_1=\lambda_2$), since the coefficient of variation is 1 and the composite distribution reduces to a single exponential stage with rate $\lambda_1=\lambda_2$. Consequently, for these models the local balance and non-local balance approximations yield exact solutions. The non-local balance approximation furthermore yields exact solutions for $N=1,\alpha=1$ models, since $\lambda_1=2p_1$ and $\lambda_2=2(1-p_2)$. This is coincidentally the standard form assumed by algorithm S for hyperexponential distributions in its determination of an equivalent general Erlang distribution with identical first and second moments. Thus in these cases, the non-local balance approximation will be exact. The local balance approximation is exact only if additionally $\beta=1$.

Let us address the question of why the non-local balance approximation is not exact for all N=1 models. The value of ρ is completely specified by the first two moments of the desired hyperexponential distribution, and thus only the first two (but no higher) moments of the standard form can be made to agree with any two stage hyperexponential distribution. The effect these higher moments play in the accuracy of the approximation can be deduced from a study of the N=1 models for which $\alpha\neq1$ and $\beta\neq1$ with exponential I/O's. Since the first two moments of the composite I/O distribution are known exactly, the only error introduced by the non-local balance approximation is that incurred by ignoring third and higher order moments. We see this error is at most 2%. Likewise, the error in ignoring second and higher order moments (the local balance approximation) is 9.7%. We conjecture that extending algorithm S's standard form to

match third and higher order moments for hyperexponential distributions would result in greater accuracy for these cases. Doing so, however, would severely increase the expense of solving the state system by the algorithm's recursive technique, thus limiting its applicability to only the most trivial models.

2.9 Summary and Applications

We have provided a tool to the analyst by developing an extended central server model which incorporates the behavior of a passive resource by limiting the level of parallelism in the I/O subsystem. It was shown that the classic central server model, which ignores this behavior, results in unacceptable error (16% in thruput). Two means of solving the extended model approximately were developed. The first, labeled the local balance approximation, was independently developed by a number of authors [A4, C10, K2] including this one. Validation against exact solutions showed the approximation to be good (less than 5% error in thruput) in most cases and adequate (between 5% and 8% in thruput) in a few. The local balance approximation utilizes the same computational techniques as the original central server model, thus incurring little or no additional expense to the analyst. The second approximation, labeled the non-local balance approximation, requires some additional techniques but results in a significant improvement in accuracy. The additional techniques utilized by the second approximation allowed us to conjecture an upper bound on the exact utilization. The bound was found to hold in the validation study. Bounds have been developed in previous

work [A4], but are applicable to only trivial structurings of the model.

The question arises, "When will the additional accuracy provided by the non-local balance approximation be worth the additional expense?" Analysis of the validation data suggests an informal heuristic, based on the observations that the non-local balance approximation is significantly more accurate than the local balance approximation when: (1) the local balance central server model results and the local balance approximation results most disagree, and (2) when the estimated coefficient of variation, C, is farthest from 1. If the measures derived from both the central server model and the local balance approximation are close (say 10%), then stop. Otherwise, apply the estimate for C of section 2.7.2, which involves trivial expense. If C is close to 1 (say 20%), then stop. Otherwise apply the non-local balance approximation. This heuristic will minimize the expense of the analysis.

The passive resource model finds a variety of applications. The computer system designer faces the problem of how many I/O processors are required by the system. The operating systems analyst must decide how to best allocate the processors and when it is desirable to purchase additional processors. Some simple workload characteristics and device processing rates are sufficient to parameterize the passive resource model. The effect of varying the number of passive resources (I/O processors) for a large number of system configurations and workload characteristics is easily evaluated by the model. The approach for analyzing the effect of

various I/O processor allocations may be seen by referring to the CDC system discussed earlier. In this system some processors must be dedicated to special tasks while the remaining tasks draw processors from a common pool. The extension of the passive resource model to include both dedicated and pool processors is straightforward. The impact of various partitionings of processors between the special tasks and the pool is easily provided by the model. Another question facing the analyst wishing to better the performance of an existing system is "When should additional I/O processors be purchased?" The passive resource model finds immediate application to this problem. For example, the local-balance approximation was applied to the interactive computer system model of the University of Texas CDC 6400 of Brown [B6] to determine the benefit of increasing the number of peripheral processors. The model revealed that additional PP's only negligibly increased system thruput.

Although we have limited our consideration of passive resources to I/O processors the techniques developed can be applied to more general models. Recall that Courtois [C11] uses the local-balance approximation to solve a memory contention model of an interactive system in which the degree of multiprogramming is restricted to prevent thrashing. An approximate model incorporating both memory and I/O processor contention is easy to visualize. The I/O subsystem of the central server model would be collapsed into a single queue. We are then able to collapse the CPU and composite I/O queues. At each step the local balance approximation would be used to parameterize the queue-length dependent service rates of the

composite queues, the rates being constrained according to the number of I/O processors and the maximum degree of multiprogramming. The accuracy of such a general approach is not known. The non-local balance approximation would undoubtedly prove more accurate. A difficulty lies in assigning a composite coefficient of variation to the single queues--but this is a problem which should easily be overcome. The problem of interrelating passive resources is an area for further research.

Buzen states that the trend in I/O architecture has been toward increased parallelism. Knowledge of the effects upon performance of this parallelism is at this time qualitative and not quantitative for the reason that accurate methods easily modeling this parallelism are not available. The work in this chapter is a step toward providing this quantitative insight to the analyst.

III. Analysis of Resource Request Patterns in a Library File System

3.1 Introduction

In this chapter we investigate the consequences that the patterning of resource requests can play in the performance of a library file system of a multiprogrammed computer. The investigation proceeds in two steps: (1) an analysis of program behavior to determine the existence and the nature of the patterning, and (2) a study to determine if we can take advantage of the patterning by predictive scheduling in the memory management of the library system. The assumption that patterning occurs in the library system references stems from the observation of "clustering" in the file references by the calling programs (the process graphs derived by Anderson [A1, ch.4]). Assumptions on the existence of patterning and its nature are tested by determining the degree of validity of reference generation models embodying the assumptions. From an analysis of the models' ability to predict a characteristic metric of empirical reference strings, we conclude that patterning exists but that the nature of the patterning is too complex to be captured by our models. The correspondence of the characteristic metric to the performance of the library system is established by results from trace driven simulation and analytical performance models.

Next a study is made to determine if we can take advantage of the patterning by the use of a dynamic memory management strategy. If the partitioning of the library between different levels of memory

remains fixed in size and content over time, then we label the allocation strategy static. If either the size or the contents of the partitions may change in time, we label the allocation strategy dynamic. A performance comparison of a dynamic and a static memory strategy over a number of empirical reference strings reveals the general superiority of the static strategy. This result leads us to conclude that the degree of patterning is too slight for a dynamic memory management strategy to be effective.

The library file system we study is the Peripheral Processor Library of the University of Texas UT-2D dual computer system. Each file of the system is a peripheral processor program. The memory management strategy considered controls the allocation of files between two levels of memory in the system. A desirable allocation strategy is one in which the number of accesses to files on the slower memory is minimized. The problem of static partitions in file systems has been extensively studied. If queueing (an inevitable consequence of multiprogramming) is ignored and the references to files can be described by independent random variables, then an optimal allocation of the files among memories exists [A3, C7, R1]. Foster [F2] considers the above case with queueing for memory devices and formulates an iterative procedure for determining an optimal allocation. The reader is referred to [F2] for an excellent summary of work in this general field. If the file references do not obey the independence assumption, then the analysis becomes much more complicated. Under such conditions a dynamic allocation strategy may prove advantageous. Determination of appropriate dynamic allocation

strategies is an active area of research being motivated in part by increasing development of data base systems.

A parallel to the dynamic file allocation problem is seen in a demand-paged virtual memory system. We draw the parallel between variable sized files and fixed size pages. The performance of paged systems depends upon the appropriateness of the page replacement policy of the memory manager [D2]. Many page replacement policies are chosen with the assumption that the page references obey Denning's principle of locality [C8]. The speculation that file references in the PP library system may obey this principle is tested in this chapter. If file references do obey locality, then a dynamic allocation strategy may improve the performance of the library system. This is the thrust of this chapter, and we investigate this question in the following manner.

We first define static and dynamic file allocation strategies and consider the conditions under which each strategy is advantageous. We next describe the peripheral processor library file system in detail. Trace driven simulation and analytical models are constructed of the library system under existing and proposed allocation strategies. Comparison of the various strategies reveals the superiority of a static strategy. One purpose of the models is to verify the importance of the reference fault probability (the probability of referencing a file on disk given a reference) to the performance of the library system and the appropriateness of using this fault probability as a performance metric. An advantage of this metric is that it can be obtained directly from the file reference trace,

avoiding the expense of simulation. We next evaluate a number of reference traces under various allocation strategies, comparing the performance of each strategy by the fault probability metric. We examine both reference traces as they are encountered by the library system in a multiprogramming environment and reference traces simulating a monoprogramming environment. As expected, the monoprogrammed traces imply a greater degree of locality.

We test the assumption of independence of references (an assumption in direct contradiction to the assumption of locality of references) by constructing a reference generation model obeying the independence assumption. We conclude that the assumption of independence is unrealistic.

In a similar vein of investigation, a Markovian model of referencing is constructed to determine the appropriateness of analyzing the reference trace as a Markovian process. Results lead us to conclude that, although more accurate than the independent reference model, this is a poor approach to the problem.

Finally, comparison between a dynamic strategy and a static strategy are made over a number of reference traces taken from the library file system. Results of this study lead us to the conclusion that a static allocation scheme is, in general, superior to the dynamic schemes studied for the PP library file system, given a multiprogramming environment. Under extreme constraints, such as a very small amount of fast memory available, one dynamic scheme is superior. In the last section we summarize the results of this chapter and make a suggestion on how the allocation strategy of the existing

library system may be improved, based on this study.

3.2 The UT-2D Peripheral Processor Library

UT-2D is the operating system for a CDC 6600 and a CDC 6400 at the University of Texas at Austin. The two machines run as independent, though communicating, computers. During normal operation the 6600 runs a batch workload while the 6400 runs an interactive workload. As briefly described in chapter II, each machine is compared of a central processing unit (CPU), a number of peripheral processing units (PP's), and a central memory (CM). Extended Core Storage (ECS), disks, and tape drives comprise the external memory hierarchy of the system. The two machines share ECS and the disks, while tape drives are dedicated to a single machine. PP's maintain the interface between a machine's CM and the external memory hierarchy. The set of programs which are executed by PP's to maintain the interface and perform other functions are labeled peripheral programs. At present, users are not allowed to write peripheral programs although they may execute peripheral programs maintained by the system. A PP executes a peripheral program out of its own 4096 12-bit memory. This memory also serves as a buffer in which data is transferred between CM and the external memory hierarchy. Since a PP's memory is much too small to contain all programs it may be called upon to execute, the programs reside in a library denoted the peripheral processor library.

The PP library, in turn, is large enough so that it is desirable that not all of it be kept in central memory. Thus the

library is apportioned between central memory and the external memories. In the present system the library is apportioned between CM and disk. The set of PP programs required for each machine are slightly different, and thus the composition of each machine's CM (or resident) portion of the library is slightly different. Both machines access the identical disk resident portion of the library, which is maintained on the system disk. Presently, the composition of resident programs and non-resident (disk resident) programs is fixed when the operating system is initially loaded and does not change during the course of the day. The PP library is constantly undergoing slight modifications by the system's support staff. During the interval between the first and last trace tapes taken, the library grew from approximately $12{,}000_8$ words for both resident and nonresident partitions to the sizes indicated in tables 3.1 and 3.2.

Communication between an executing job in CM and the peripheral programs is maintained via a special communication area within the job's field length ("mailboxes"). When a job requires the service of a PP it posts a message in its mailbox. A PP monitor (to which a PP is dedicated) routinely polls the mailboxes for messages from executing jobs. When a request for service is encountered the monitor assigns the first available PP to the request. The PP picks up the request, which is in the form of a peripheral program call, and examines a library directory to obtain the address of the peripheral program. The address is a CM location if the program is resident and a disk track address if it is not. If the program is resident in central memory, the PP loads the program from

the appropriate address in CM into its own memory. This is accomplished by a loader which is resident in the PP's memory. If the program is not resident in CM, the PP presents a reserve channel request for the system disk channel to the PP monitor. If the channel is busy, the request is queued until the channel becomes free. After gaining possession of the channel, the PP loads the program from disk, then releases the channel. In either case, the PP runs the program after loading it.

Many PP programs require overlays of other PP programs (library files). The overlay program is accessed in the manner just specified, with the exception that the program request originates from the PP instead of a user mailbox. Overlays may be several layers deep, the deepest consisting of the disk drivers, for example. The peripheral programs perform a variety of functions. The preponderant function is I/O for executing programs via the peripheral program CIO and its overlays. Program CIO effects transfers from I/O buffers in central memory into files on the disk, and vice-versa. A number of other PP programs, as an illustration of the duties performed by the programs, include: OPE, which opens files; SNP, which obtains a snapshot of central memory for debugging purposes; and RCC, which reads a control card. Tables 3.1 and 3.2 present a typical configuring of the peripheral processor library for the 6400, specifying which programs are resident and which are non-resident.

3.3 Static and Dynamic File Allocation Strategies

When confronted with a set of files which must be allocated

TABLE 3.1: Resident Portion of Peripheral Processor Library

MACHINE: CDC 6400
DATE: April 30, 1975

NAME	LENGTH (OCTAL)	NAME	LENGTH (OCTAL)
1AJ	235	9C4	33
1DB	231	9C5	43
1PL	224	9C6	16
1PS	212	9C9	145
1RJ	350	9CH	133
1SJ	267	9DA	36
1SR	222	9DD	34
1SS	437	9DE	54
1TD	164	9DH	65
1TM	576	9DM	75
2CT	726	9DP	64
2DF	37	9DS	66
2JE	25	9DV	66
2MT	712	9EO	42
2PD	267	CIO	205
2PU	327	CPU	423
2RD	123	EPR	346
2TJ	32	LDR	277
2TS	515	MSG	46
2WD	126	OPE	110
2WM	301	PAT	135
3AJ	150	PCC	55
3EA	273	PFM	414
5DB	44	QIO	400
5DE	42	RCC	47
5DG	46	RFL	156
9C0	101	RJE	1056
9C1	43	RTA	64
9C2	57	SIS	252
9C3	40	SNP	67
		TOTAL	17,312

TABLE 3.2: Non-Resident Portion of Peripheral Processor Library

MACHINE: CDC 6400
DATE: April 30, 1975

NAME	LENGTH (OCTAL)	NAME	LENGTH (OCTAL)
1CD	740	9DJ	102
1ED	1022	9DK	44
1EI	467	9DL	43
1XX	216	9DQ	21
2AM	76	9DR	30
2DS	225	9DT	60
2E1	137	9DW	52
2E2	140	9DX	67
2EF	216	9DY	72
2FE	12	9DZ	52
20U	121	9E1	33
2PL	136	9ED	174
2SC	111	9OL	11
2SP	552	ABS	130
3CT	47	CLD	323
9C7	44	CLO	14
9C8	106	DMP	547
9DB	36	ERR	441
9DC	77	INT	107
9DF	30	MUX	356
9DG	55	RSF	331
9DI	65	SPR	120
		TOTAL	12,017

over memory devices comprising a memory hierarchy, a file allocation strategy must be devised. If there is no constraint on the number of files that may reside in the fastest memory, then the optimal allocation is to assign all files to the fastest memory. Unfortunately, this is seldom the case, as it is usually desirable to constrain the space in the fastest device which the files may occupy, or the set of files is larger than the space available. Thus the files must be partitioned between the various devices in the hierarchy.

3.3.1 An Optimal Static Allocation Strategy

We assume a library of N files, each indexed by i, i=1,...,N. Each file i is characterized by a size, s_i, and a frequency of access, f_i. The optimal allocation strategy must partition the library among two memories (characterized by different access times and capacities) so that the reference fault probability (the probability that a file, when accessed, does not reside in the memory) is minimized. Alternatively, we wish to maximize the reference success probability. We assume that the slower memory has capacity greater than the size of the library, and the faster memory has capacity L which is less than the size of the library. An optimal static allocation strategy exists assuming that the access frequencies of the files do not change in time. If, for the sake of tractability, we make this assumption, the situation degenerates into a knapsack problem for which solution techniques are well known. We modify a dynamic programming technique [G2] to obtain an algorithm for determing the optimal allocation.

Formalizing, let an optimal success frequency for n files be defined by

$$F(n,\ell) = \max \sum_{i=1}^{n} x_i f_i$$

subject to the constraints

$$\sum_{i=1}^{n} x_i s_i \leq \ell \quad \text{and} \quad x_i = 0,1$$

where $0<\ell\leq L$ and $0\leq n\leq N$. The x_i specify the optimal choice of files from the set up to and including file n given a size constraint of ℓ. Clearly, F(N,L) defines the optimal success frequency of the allocation problem. The algorithm determines $F(n,\ell)$ as n steps from 1 to N and ℓ steps from 1 to L. At each step we determine $F(n,\ell)$ by

$$F(n,\ell) = \max \begin{Bmatrix} F(n-1,\ell) \\ F(n-1,\ell+s_n) + s_n \end{Bmatrix}. \qquad (3.1)$$

The algorithm terminates with the maximum success frequency left in F(N,L). The maximum success probability may be obtained from F(N,L) by normalization of the frequencies.

Algorithm 3.1 The Optimal Static Algorithm

Step 1. Initialize F(n,0)=0 for n=1,...,N.

Step 2. For n=1 until N do step 3.

Step 3. For ℓ=1 until L do step 4.

Step 4. Choose $F(n,\ell)$ by equation (3.1).

The algorithm has the computation advantage of being very fast (of order N·L) and space efficient, as only 2L values for $F(n,\ell)$

need be maintained. However, the storage requirement can still grow very large for realistic values of memory capacities. We can obtain upper and lower bounds on the solution at a substantial space saving by reformulating the problem so that $s_i' = [s_i \cdot L'/L]$ and $s_i'' = [s_i \cdot L'/L]$, where $L' < L$ is the new memory constraint. This merely "scales down" the size constraint so that the algorithm runs in $2L'$ space at the expense of definition of the file sizes. The s_i' result in a lower bound for the success function while the s_i'' result in an upper bound. We will frequently use these bounds when the algorithm is applied in the following sections.

3.3.2 Dynamic Strategies

Dynamic allocation strategies outperform static strategies when the file reference frequencies change significantly with time. The success of a particular dynamic strategy depends upon the nature of this change. Although a host of dynamic allocation strategies exist [D1, D2, F2] we shall limit our consideration to fixed partition strategies, that is, strategies in which the memory constraints remain fixed. The reason for this choice is that we wish to keep the size of the PP library's CM partition fixed so as to minimize the overhead due to interaction between the library memory manager and the general memory manager.

A dynamic allocation implies the migration of files between memories. The two variables controlled by a strategy are: (1) when the migration occurs, and (2) which files migrate. The strategies we will consider allow migration at the time of a reference fault. The

desired file will be brought into CM at the expense of an inactive file(s) which will be deleted from CM for the desired space. The choice of a "demand paging" algorithm is motivated by the proof of Coffman and Denning that for any "pre-paging" algorithm there exists a more desirable demand paging algorithm. The choice of file(s) to delete is determined by the file replacement policy. We consider two policies. Our description of the policies is deliberately informal since formal treatments of both policies are prevalent in the literature.

3.3.2.1 The Least-Frequently-Used-Policy

The LFU replacement policy deletes the least-frequently-used file from CM. Frequency counts are kept for every file. Each time a file is referenced its count is incremented. When a reference fault occurs those file(s) with the smallest counts are deleted to make space available, in the order smallest-count-first, until adequate space is provided for the referenced file. It remains to specify the intervals at which counts are decremented (or zeroed) in order to take advantage of the (assumed) changing reference frequencies. We shall label an interval determined by a fixed number of references the quantum and specify that all counts be zeroed or decremented at the conclusion of each quantum. Note that if the quantum is set to infinity and if the ordering of files by frequency is static, then the partitioning enforced by LFU is static.

3.2.2.2 The Least-Recently-Used Policy

Under the LRU policy the indices of files in CM are kept ordered

in a stack. A file's index, when the file is referenced, is drawn from its place in the stack and pushed onto the top of the stack. The stack mechanism insures that at any time the LRU file's index occupies the bottom of the stack. When a reference fault occurs, files are deleted in opposite order to their position in the stack until adequate space in CM is provided.

3.3.3 Trade-Offs

The properties of the static and dynamic strategies most relevant to performance considerations are: (1) the degree of overhead each incurs, and (2) how that overhead is distributed. The static strategy incurs no run-time overhead once the partitioning is made and requires the negligible overhead of gathering reference frequencies over some period of time before partitioning. Fortunately, the number of library files for this study is modest and so the size of the tables required is reasonable. The dynamic strategies, on the other hand, incur significant run-time overhead due to the necessity of (1) a stack mechanism or running frequency counts and (2) a dynamic memory manager. The overhead incurred by the first mechanism is constant, while that incurred by the second is dependent upon the degree to which the chosen policy minimizes the fault rate.

3.4 Models of the Library File System with Static File Allocation

Two models of the library file system under the static allocation are considered. Both models attempt to capture the behavior of the library file system under the allocation scheme implemented on the UT-2D system. The first model is a trace driven simulation.

The purpose of the simulation model is twofold: First, by rigid validation the model establishes that our understanding of the library file system is sufficient to derive an accurate model; and second, to serve as a basis of validation of the dynamic model. The second model is analytical, and establishes the importance of the file reference fault probability in determining the performance of the library system. The performance metric chosen for comparison is the saturation request throughput, that is, the rate at which file requests could be satisfied if the system was operating at full capacity.

3.4.1 A Trace Driven Simulation Model

A trace driven simulation model of the library file system was constructed. The model attempts to capture the behavior of the library file system under the memory management scheme in existence when the trace tapes were taken. This model is labeled static because the memory management of the PP library consists of static partitioning. The library at system initialization time is partitioned between CM and disk. The contents of the partitions do not change so long as that version of the system is executing.

The simulation is implemented in ASPOL [C9] and consists, in part, of a set of resources and a set of peripheral processes which utilize these resources in an orderly fashion. The resources are PP's, central memory, the system channel, and the system disk. A peripheral process holds a PP for the duration of its lifetime. The process graph for a peripheral process is seen in figure 3.1. The states of

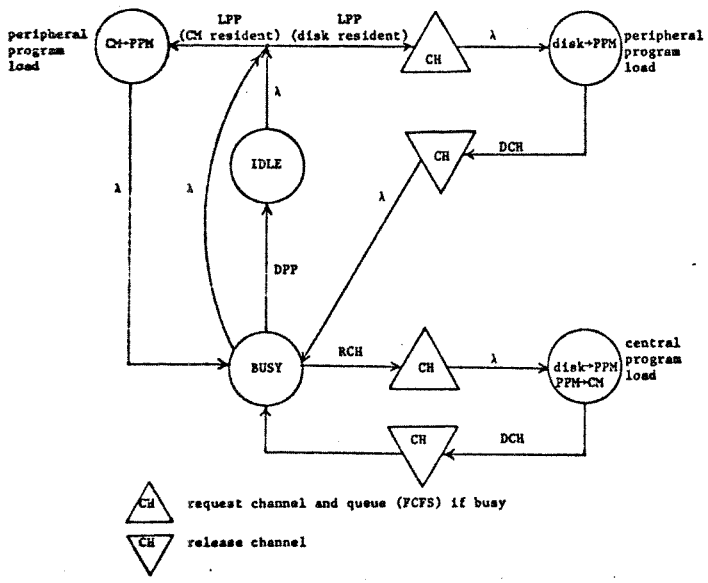


Figure 3.1: Static Modal Process Graph

the process are represented by circles. Each state is characterized by the holding time during which the process occupies that state. Triangles represent allocation requests and releases for the system resources of channel and disk. If the system resource is not available when requested, the process queues in first-come-first-served order until the resource becomes available. Directed edges define the transitions between states. Since the simulation is event-trace driven each edge is labeled by the event which, when encountered in the trace, causes the process to transit from the present state to its future state. The letter λ is used to denote transitions which are not trace event driven. For example, the "peripheral program load, disk→PPM" state in Figure 3.1 has a holding time which is not obtained from the trace but instead from one which is internal to the simulation. Thus the process changes state at the appropriate time in the simulation without advancing the trace input.

The reader will note that the static model process graph closely conforms to the behavior of the actual system previously described. A peripheral program request is denoted by the event "LPP" (locate peripheral program). If the peripheral program is CM resident, the process holds for the amount of time necessary for a CM to PPM load, estimated at .01 ms + .01 ms per word transferred.

It is assumed that the time necessary to make the determination of residency from the appropriate system tables is negligible. Once the program is loaded from CM, the process enters the busy state, corresponding to the execution of the loaded PP program. The busy period is terminated when event DPP (drop PP) is encountered. From

the busy state the process may terminate (already described), request an overlay (the λ edge), or request a central library file. Central library files are the CPU-executable codes residing on the system disk which users or the system may request. Examples of central library files are the compilers, file manipulation routines, and editors. The system channel is requested and, when allocated, is held for the time necessary to effect the copying of the code from the system disk into CM. This time is known explicitly from the event DCH ("drop channel") in the trace. This completes the description of paths originating at the "peripheral program load CM→PPM" state.

If the peripheral program requested is disk resident, the system channel is requested. When the channel is allocated, the process holds the channel for the time required for the program to be loaded from disk into the PP's memory. This holding time is estimated to be a distribution comprised of three stages in series. The first stage approximates the seek time and is a uniformly distributed random variable between 20 ms and 45 ms. The second stage represents the wait for rotational latency and is a uniformly distributed random variable between 0 ms and 54.5 ms. The third and final stage represents the time required to make the transfer of the program from disk to PP memory, estimated at .045 ms/word. The figures are taken from [B5], chapter 4. The initial simulations revealed the composite distribution to have a mean of 43 ms and variance of .0009 ms^2. Later simulations used an Erlang distribution of the same mean and variance for computational ease. It was verified that the effect of the Erlang distribution (as opposed to the original estimate) upon the performance

metrics was negligible. This completes the description of the model's structure.

The events (already defined) which drive the model are extracted from an event trace produced by the UT-2D event recorder [H3]. We shall label the trace of extracted events the reduced trace. The methodology of extracting the interesting events is straightforward but tedious and will not be presented. The interested reader is referred to [A1, B5] for a discussion of the general methodology and to [F2, ch. 4] for a description of how channel queueing times may be subtracted from the channel holding times. Extracted with each event is the time of the event and the particular PP associated with it. These times are absolute clock times. The holding time intervals between two events are obtained by the appropriate subtraction of the absolute times of the events. The central program load time, the BUSY time, and IDLE time for one incarnation of a process are thus obtained. The number of pool PP's available for the 6600 is 7, for the 6400, 4. The number of pool PP's is the number of processes active in the simulation.

3.4.1.1 Validation

The model was validated against a trace tape analyzer constructed separately by J. R. Haley. The validation metric was the file reference thruput defined as the total number of references divided by the total run time. A trace tape taken of the interactive (6400) system running under a moderate load in mid-afternoon was chosen for validation. The composition of the PP library, at the

time the tape was taken (somewhat different from the library of tables 3.1 and 3.2) was used to parameterize the simulation. The results of the validation are found in table 3.3.

3.4.2 Analytical Model

An analytical queueing network model was constructed of the actual static allocation strategy in order to test the dependence of the library system's performance to the reference fault probability. The model is shown in figure 3.2. The servers without queues represent servers where all customers are processed at the specified rate. The structure of the model is taken directly from the process flow graph of the static simulation model of the previous section. Mean holding times for the servers were obtained by an analysis of trace tapel. The mean service times for all servers except CM were taken from an analysis of trace tapel. Let us order those files CM resident from 1 to n_{CM} and those non-resident from $n_{CM}+1$ to L. The CM mean service time $\bar{t}_{CM}$ was derived by the formula

$$\bar{t}_{CM} = \left(\sum_{i=1}^{n_{CM}} f_i \right)^{-1} \sum_{i=1}^{n_{CM}} f_{ii} \cdot (.01\ s_i\ \text{words}^{-1} + .01)\text{ms}$$

where f_i and s_i are the frequency and size of a file, respectively. Residency is determined by inspection of the PP library partitioning at the time the trace was taken. The routing probabilities were likewise taken from an analysis of the trace. The usual local balance assumptions were made to make the solution of the model by ASQ [K1] possible. Results of the model validation are found in table 3.3.

TABLE 3.3: Validation Results

TRACE: TAPE1
DATE TAKEN: October 18, 1974
TIME: 15.34.39

	TRACE ANALYZER	SIMULATION	ANALYTICAL MODEL
INITIAL TIME (sec)	27510.09		
END TIME (sec)	28318.73		
ELAPSED TIME (sec)	808.64	829.21	
NO. REFERENCES	50,157	50,157	
REF. THRUPUT (sec^{-1})	62.03	60.49	64.1
Percentage difference of thruputs		-2.47	3.33

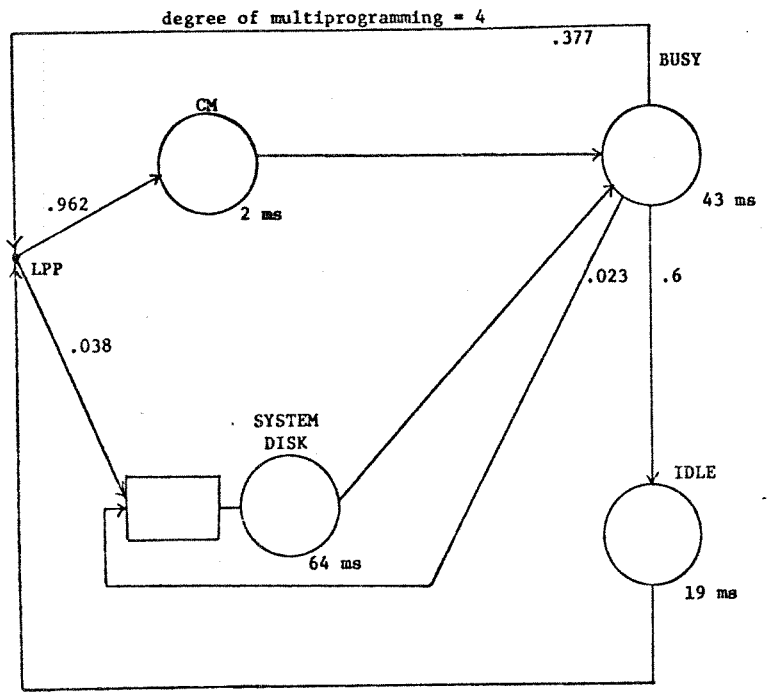


Figure 3.2: Analytical Model of the PP Library System

Confident in the validity of the analytical model, we now turn to an analysis of the dependence of the performance of the library system upon the fault probability. The model is slightly reconfigured to eliminate the idle phase of process activity in order to produce a saturation model. We use a saturation model because we cannot isolate the workload placed on the library system from the system's capacity to do work (which we define as its saturation reference thruput). Unable to determine this dependence (to do so would require constructing models of the entire computer system), we turn to a saturation model to evaluate the relative improvement of work capacity with a decreasing fault probability--realizing that the improvement to overall system performance must be less. The reference thruput as a function of reference fault probability is presented in table 3.4.

3.5 A Model of the Library File System with Dynamic File Allocation

The dynamic model differs from the static model in only two important respects: the presence of a memory manager and the fact that all peripheral programs are loaded from central memory to PP memory. The memory manager controls the allocation of CM for the peripheral program. If a peripheral program is not in CM ("LPP disk" in Figure 3.3), then storage for the program must be reserved by a request to the memory manager. The fixed CM partition for the resident portion of the library leads to the fact that if sufficient space is not available for the new program an inactive program must be deleted from CM. Which inactive programs are

TABLE 3.4: Work Capacity of Library System as a Function of Reference Fault Probability

REFERENCE FAULT PROBABILITY	REFERENCE THRUPUT (sec^{-1})
.01	82.55
.02	81.15
.03	79.70
.04	78.20
.05	76.67
.06	75.12
.07	73.55
.08	71.96
.09	70.37
.10	68.79
.20	54.04
.30	42.70
.40	34.58
.50	28.78
.60	24.52
.70	21.30
.80	18.80
.90	16.81

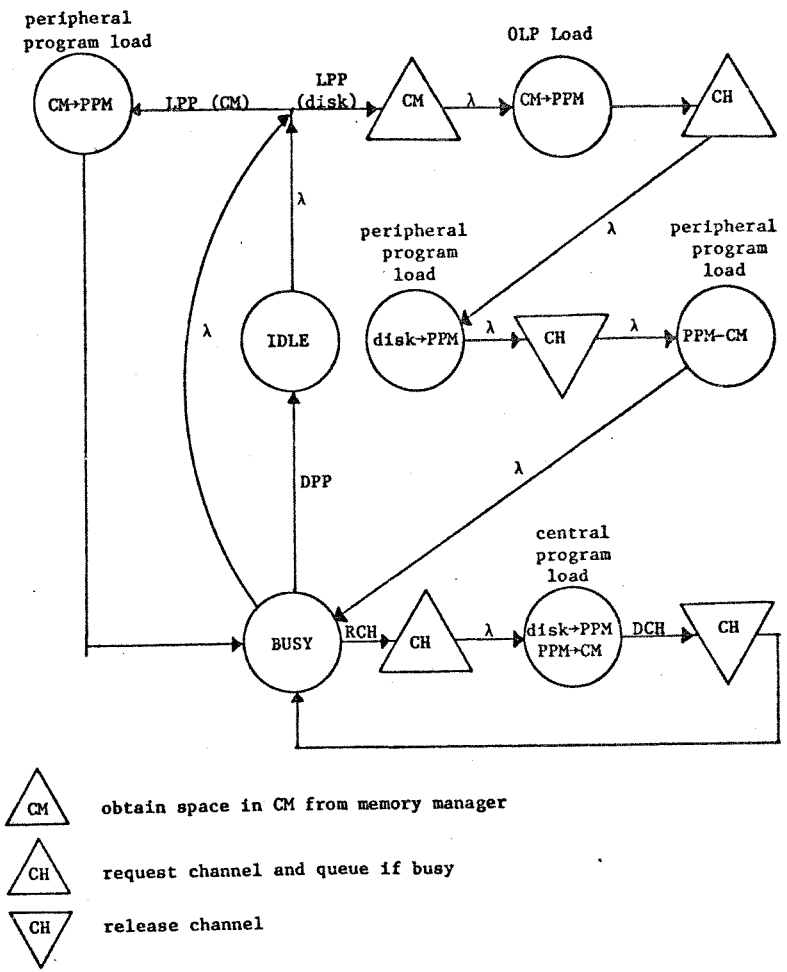


Figure 3.3: Dynamic Model Process Graph

selected for deletion is dependent upon the selection rule used.

The memory manager allocates space by the following algorithm:

Step 1. If there is a gap large enough for the program, reserve that gap; else go to step 2.

Step 2. If the summation of all gap sizes is as large or larger than the size of the program, then compact, reserving space for the program and (usually) leaving one gap; else go to step 3.

Step 3. Select a program for deletion according to the selection rule and delete. Go to step 2.

Both LFU and LRU selection rules are implemented in the model. Once memory is made available the PP is loaded with peripheral program OLP, the overlay which effects the disk load. (OLP is not an existing peripheral program, and hence its label is arbitrary.) The system channel is requested, and when allocated, the desired peripheral program is loaded from disk into CM through the PP's memory. Holding time distributions are the same as described in section 3.4.1. Compaction overhead is estimated at 5 ms/word moved. A memory overhead of 3076_8 words is assumed for peripheral programs which must be CM resident. These include OLP and the low-level disk drivers. The other process paths are so similar to those of the static model that they will not be discussed.

3.5.1 Validation

Direct validation of the dynamic model is not possible since a dynamic strategy is not implemented in the UT-2D system. Our confidence in the accuracy of the model must stem from the fact that the model is structured and parameterized similarly to the static model.

3.6 Comparison of Strategies by Simulation

In this section we compare the LFU and LRU strategies over a range of central memory sizes. Again a saturation model is assumed for the reasons presented in the previous section. An initial portion of the events in trace tapel was used to drive the simulations. The metric used for comparison was the simulation completion time for the 6,575 references. The completion time of the system is bounded by the total time required to perform the central library transfers. Results of the runs are presented in figure 3.4. The LFU quantum of infinity is given for illustration since LFU performance over the trace was insensitive to both the choice of quantum and method of count decrement. We see that LFU (∞) outperforms LRU at each memory size. The relative differences in performance decrease with increasing memory size. We conclude that LFU is the superior strategy (for trace tapel). The fact that file migration under LFU (∞) is observed to decrease steadily during a run (due to the ordering of files by frequency becoming fixed) infers that LFU (∞) provides a lower performance bound for the optimal static strategy. Unfortunately, the simulation is not a fair proving ground for this conjecture because the size of programs which must be specified resident is different for the static strategy due to realistic system constraints. The comparison of all three strategies is presented in section 3.8.

3.7 Reference Trace Analysis

The great expense of simulation leads us to seek other

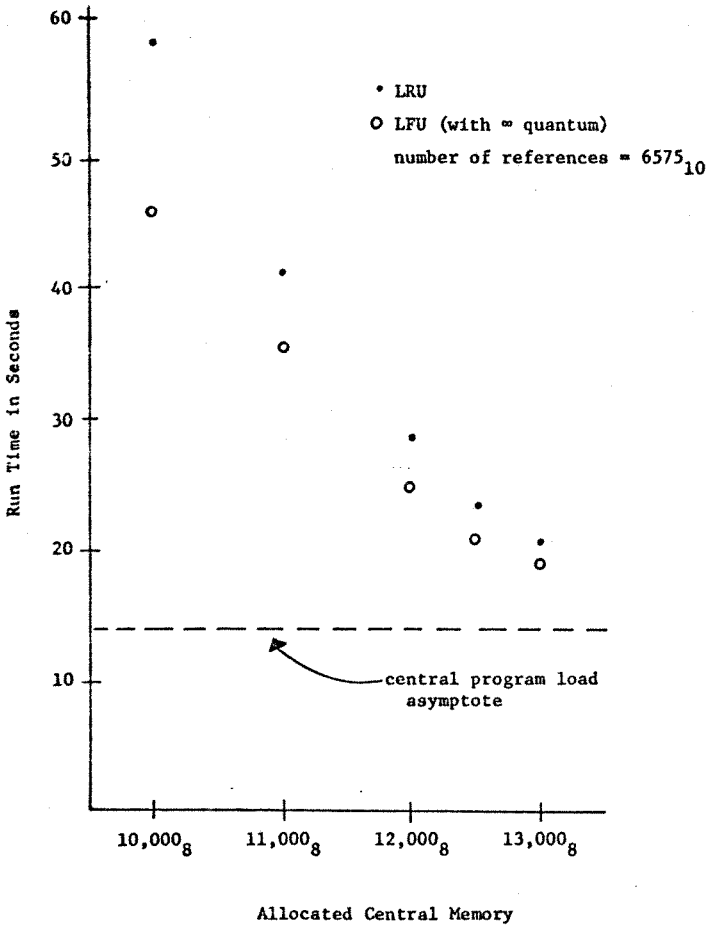


Figure 3.4: Comparison of Allocation Schemes by Simulation Run Time

methods for comparing different memory strategies. The singular importance of the reference fault probability has already been established for this comparison. The reference fault probability can be determined directly from the trace stream of references at lesser expense, allowing us to examine a greater number of traces. We shall use the term "reference string" to denote the string composed solely of PP program references extracted from an event trace.

As already stated, dynamic strategies outperform static strategies if the reference frequencies change in time; the relative improvement depending on the nature of the change. We will look to the nature of a user's program's behavior in the following sections to try to determine if we can use this behavior to the benefit of a dynamic strategy.

3.7.1 Locality in the Reference Trace

Program behavior has been investigated by a large number of researchers, who assume that programs in general demonstrate the property of locality in page referencing behavior. We quote Coffman and Denning [C8, p. 286]:

> Informally, the property of locality implies that a program tends to favor a subset of its pages during any time interval and that the membership of the set of favored pages tends to change slowly. . . . We can summarize the properties of locality in three statements:
>
> L1. During any interval of time, a program concentrates references nonuniformaly over its pages.
>
> L2. Correlation between immediate past and immediate future reference patterns tends to be high, and correlation between nonoverlapping reference substrings tends to zero as the distance between them increases.

L3. Taken as a function of time, the frequency with which a given page is referenced tends to change slowly.

The assumption that programs obey locality is the basis for the implementation of many dynamic allocation strategies in virtual memory machines. The LRU strategy is considered to be the most robust over a range of CM sizes allocated to programs [D2].

Let us now engage in some speculation over the probable behavior of the artifice of a "program" which generates file references to the PP library system instead of the page references considered above. The file reference string is incremented in actuality by PP function calls from user and system programs executing in CM. These functions are performed by chains of PP overlays (library files). For example, consider the clusters of possible calling sequences from PP program CIO illustrated in figure 3.5. The graph is taken from [A1, ch. 4]. The arc labels are the probability of taking that arc upon completion of the event state labeled in the circle. Each state marked by an asterisk generates a PP library file reference. Let us assume for argument's sake that the machine is running in monoprogrammed mode and we eliminate references generated by the system. Then the reference string would consist of blocks of references, each block corresponding to the reference string generated by a user job from beginning to end. We would expect to find the blocks comprised of reference "clusters" corresponding to behavior patterns as implied by the graphs of figure 3.5. Furthermore, the set of reference clusters generated by a user job may be a function of the job, and thus reference

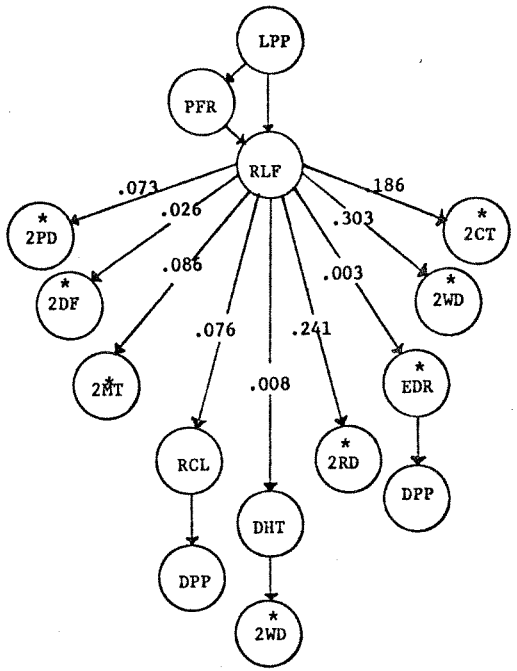


Figure 3.5: Event Trace Graph for PP Program CIO

frequencies would vary by block. If such is the case, then the string of blocks would satisfy L1, with L2 satisfied by the correlation between blocks of references being nil. In contrast to pages which are the "property" of individual programs in a virtual memory system a PP library file may be accessed by any user job (within any block), thus making L3 unlikely. However, this would not necessarily mean that a dynamic memory strategy would be less desirable than a static strategy.

We construct the above reference strings to determine if the LRU strategy can outperform the optimal static strategy. If the LRU performance is superior we might expect its superiority to hold for low degrees of multiprogramming in actuality. If the LRU strategy is not superior under the conditions of this artifice, then it is very unlikely to improve under actual conditions. The superiority of the LRU strategy over the optimal static strategy would imply the property of locality holds for PP library references. The metric we use for comparison is the reference fault probability. The comparison is made in section 3.9.

3.7.2 Construction of the Reference Trace

In its original state the event trace consists of the interleaved behavior of both system and user jobs. The multiprogrammed machine event trace is decomposed into user program event traces through methods developed by J. R. Haley. The interested reader is referred to [N1] for details of the method. The PP library related events are then extracted from the user program event trace, from which

in turn the reference string is extracted. The reference string consists of blocks of references, each block corresponding to a user program. Blocks are ordered according to the clock time the user program was initially detected. The references within each block are also ordered chronologically. We will refer to the user program reference string as "monostream" references and to the original event trace references (ordered chronologically) of interleaved programs as "multistream" references.

Three event traces (tape2, tape3, and tape4) were reduced to provide five monostream reference strings, labeled tape2a, tape2b, tape3a, tape3b, and tape4a. The "a" designates streams of interactive user jobs which began and ended within the time span of the trace. The "b" designates streams of interactive user jobs which were begun but did not complete within the trace interval. A summary of all traces is found in Appendix C.

3.8 Models of Reference Patterns

In order to better our understanding of the possible reference patterns (implied by the clustering discussed in the last section) we attempt accurate models of reference string sequencing. The probabilistic models are parameterized from statistics gathered from a reference string RS. The parameterized model is then used to generate reference string RS′. The reference fault probability (RFP) is determined for RS and RS′ under the LRU allocation strategy. The test of the model's validity is how closely the RFP for RS′ matches the RFP of RS over a range of memory sizes.

Two distribution driven models for reference generation are constructed. The first we label the "independent" reference model. It is parameterized by the probabilities $P = \{p_i, i=1,\ldots,N\}$, where p_i is the probability of next referencing file i. P is determined by normalizing the frequency counts of the references so that

$$p_i = \left(\sum_{i=1}^{N} f_i\right)^{-1} f_i.$$

A generated reference index is described by a random variable drawn from the probability mass function defined by P. The second model we label the "Markov" model. It is parameterized by determining the probability, p_{ij}, of next referencing file i given that file j was most recently referenced for i,j=1,...,N. The model simulates a Markovian system with N states. The probability of transiting from state i to state j is p_{ij}. The model is initialized to the state which will most likely be referenced the greatest number of times. The reference string is simply a track of the state of the system. See figure 3.6 for an example with N=3.

The reference probabilities for the strings generated by both models will stochastically converge to the (normalized) f_i of RS. Thus RS and RS´ wil result in the same RFP for both the LFU (∞) and the optimal static allocation strategies. This is the reason the LRU strategy is used to determine the accuracy of the models.

3.8.1 Parameterization

The models were tested against the four monostream reference

reference string: 1 1 2 1 3 2 1 1 3 2 1 1 2 1 3 2 1 3 3 1 1 2 .

frequency count of i following j

j \ i	1	2	3
1	4	3	4
2	5	0	0
3	1	3	1

p_{ij}

j \ i	1	2	3
1	4/11	3/11	4/11
2	5/5	0	0
3	1/5	3/5	1/5

Markov state graph

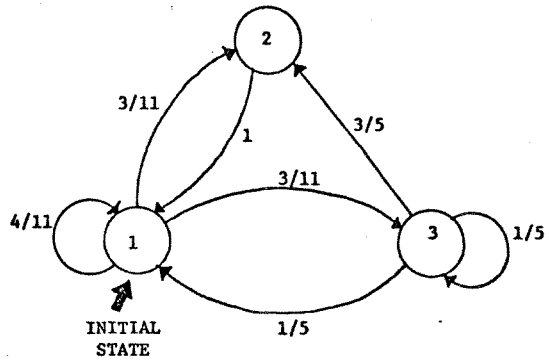


Figure 3.6: Example of Markov Reference Model

traces, tape 2a, tape2b, tape3a, and tape3b. In each case a model was parameterized by the original reference string RS and used to generate reference string RS′. An RFP and RFP′ were then obtained under the LRU strategy for both the independent and the Markov model. The results of the runs are displayed in figures 3.7-3.14. Figure 3.7, 3.9, 3.11, and 3.13 show the RFP and RFP′ as functions of CM size. For these cases the PP program (file) sizes are those given in tables 3.1 and 3.2. The other four figures show the RFP and RFP′ as functions of the number of peripheral programs, M, in CM. In the latter cases the number of programs in CM is held constant. This allows a comparison of characteristics independent of the empirical program sizes.

3.8.2 Validation and Conclusions

We find the RFP′ produced by the independent model to be consistently pessimistic. The inaccuracy of the model is especially severe (on the order of 2) for small CM and small M, and decreases with increasing CM size. These results clearly indicate that the assumption of reference independence does not hold. The RFP′ produced by the Markov model also displays the greatest error in the small CM, small program number range. The Markov model, while still inaccurate, is consistently better in predicting the RFP. Note that the RFP's displayed by both the Markov and independent model converge at the high ranges. We suspect the reason is that the LRU strategy is insensitive to the small degree of sequencing the Markov model enforces once a large percentage of the library can reside in CM.

We conclude that since both models are poor predictors of the RFP under a LRU strategy then neither the independence assumption holds for the references (no sequencing) nor does a simple Markov model (a "small" degree of sequencing) adequately capture program behavior. More complex Markov models rapidly become intractable.

3.9 Comparison of Strategies by Reference Trace Analysis

Seven reference strings were used in the comparison study. Tape2a, tape2b, tape3a, and tape3b are monostream traces taken from the CDC 6400 running an interactive workload. Tape5 is a monostream composite constructed by concatenating together these four strings. Tape4 is a multistream trace taken from the CDC 6600 running a batch workload. Tape4a is the monostream trace taken from the same event stream.

The comparison was kept deliberately simple. Each string was run against both the LRU and the optimal static strategy. Each strategy was evaluated at different CM memory sizes and for different values for M, the number of peripheral programs in CM. Tape5, the composite, was evaluated to gauge the impact of a "changing" workload on the strategies, since the two trace tapes from which the strings are extracted were taken on different days (although the same time of day).

3.9.1 Results

The results of the comparisons are found in figures 3.7-3.20. We see that for the interactive traces (tape2a, tape2b, tape3a, and tape3b) that the optimal static does worse for small CM and M and

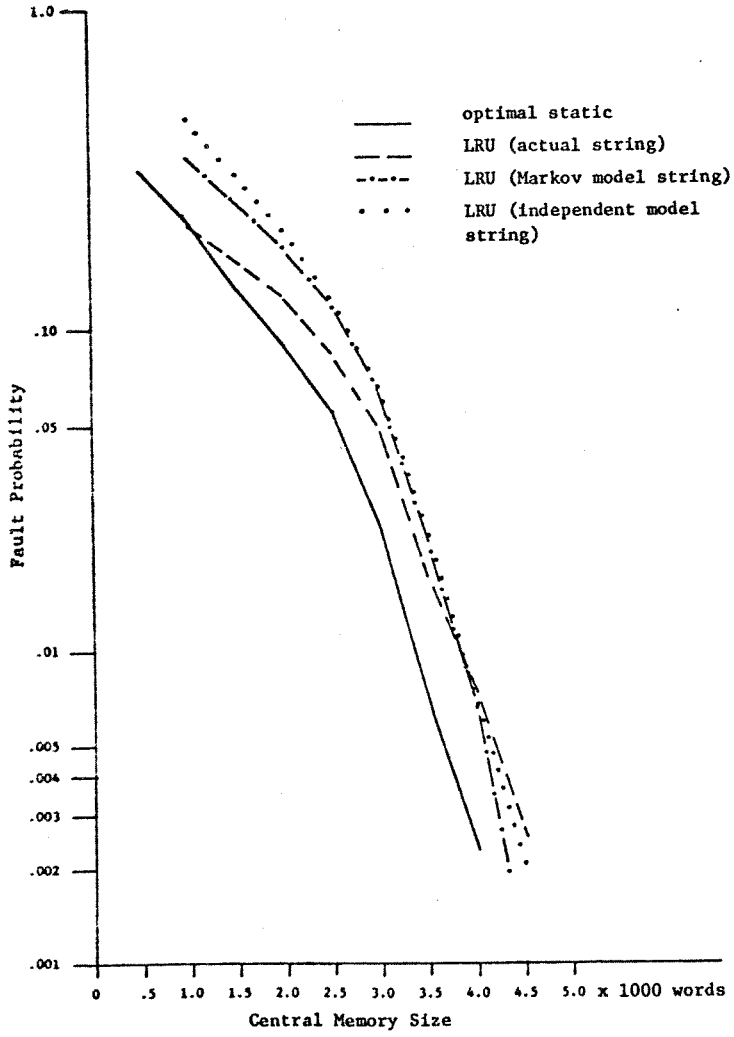


Figure 3.7: Reference Fault Probability vs. CM Size for Tape2a

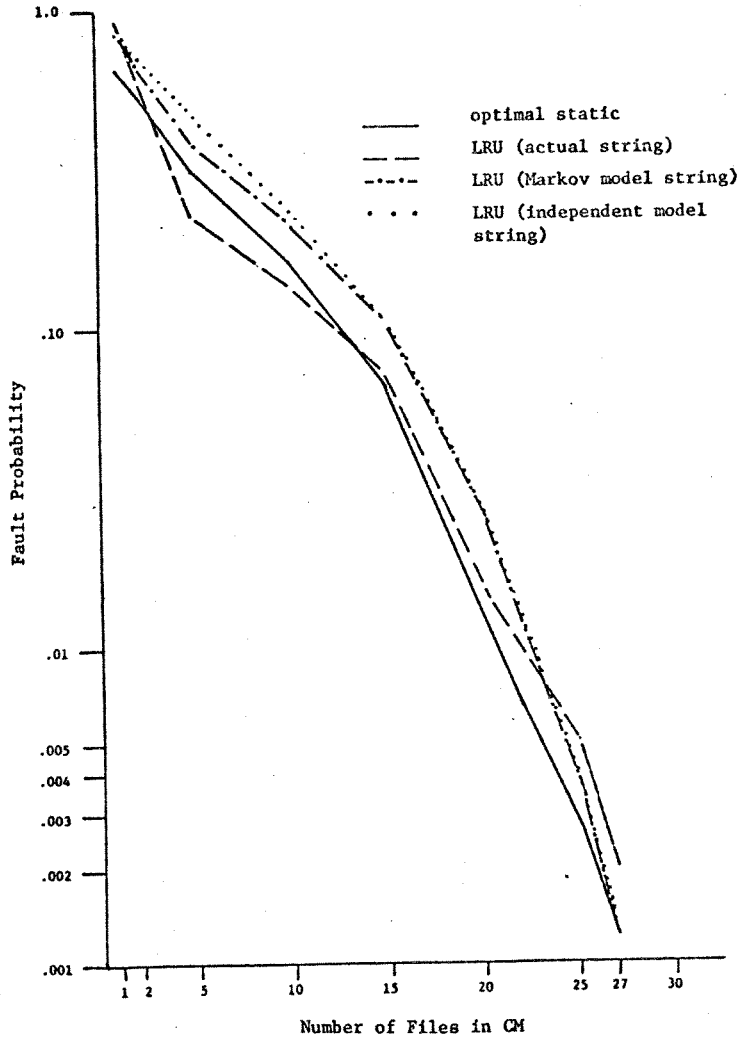


Figure 3.8: Reference Fault Probability vs. M for Tape2a

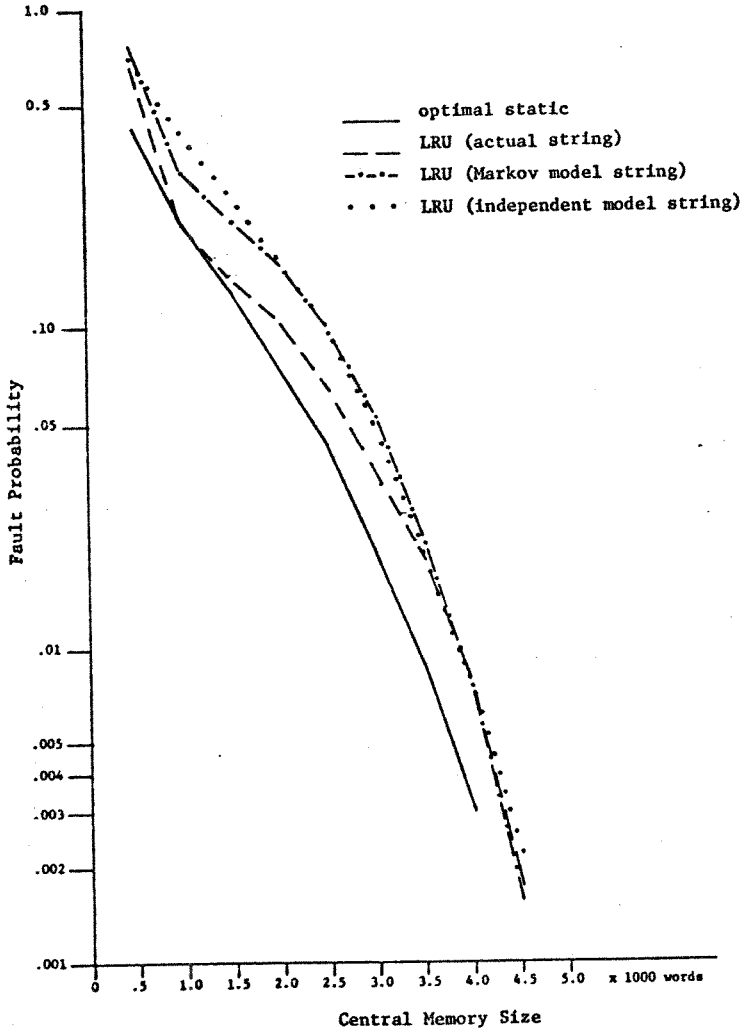


Figure 3.9: Reference Fault Probability vs. CM Size for Tape2b

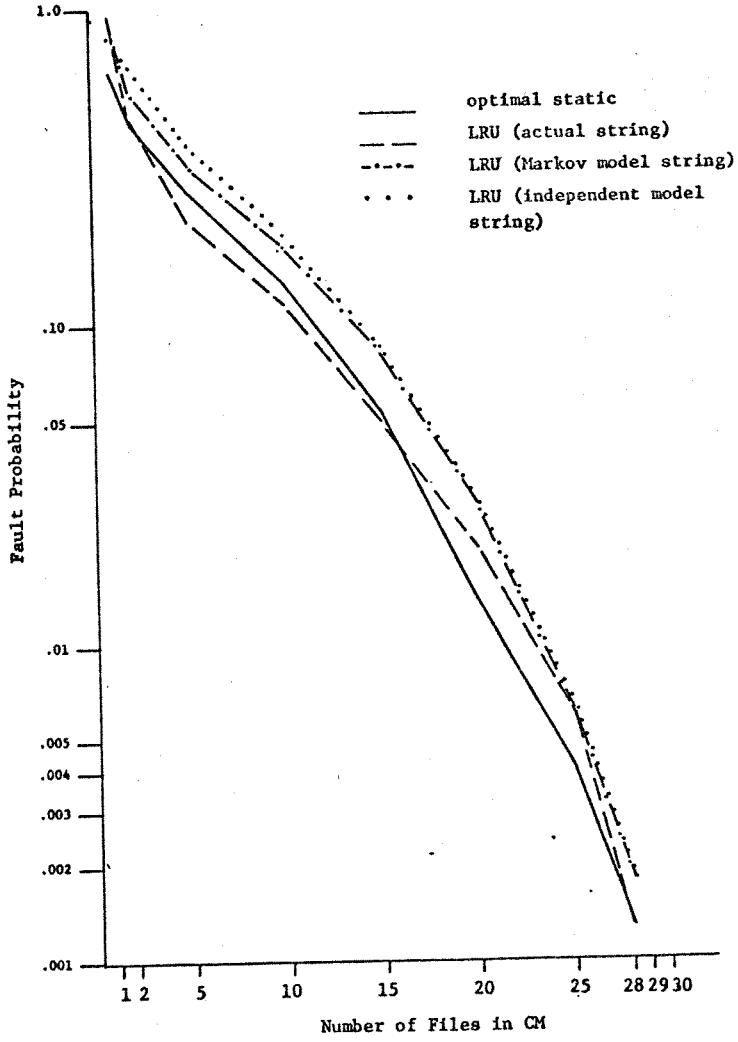


Figure 3.10: Reference Fault Probability vs. M for Tape2b

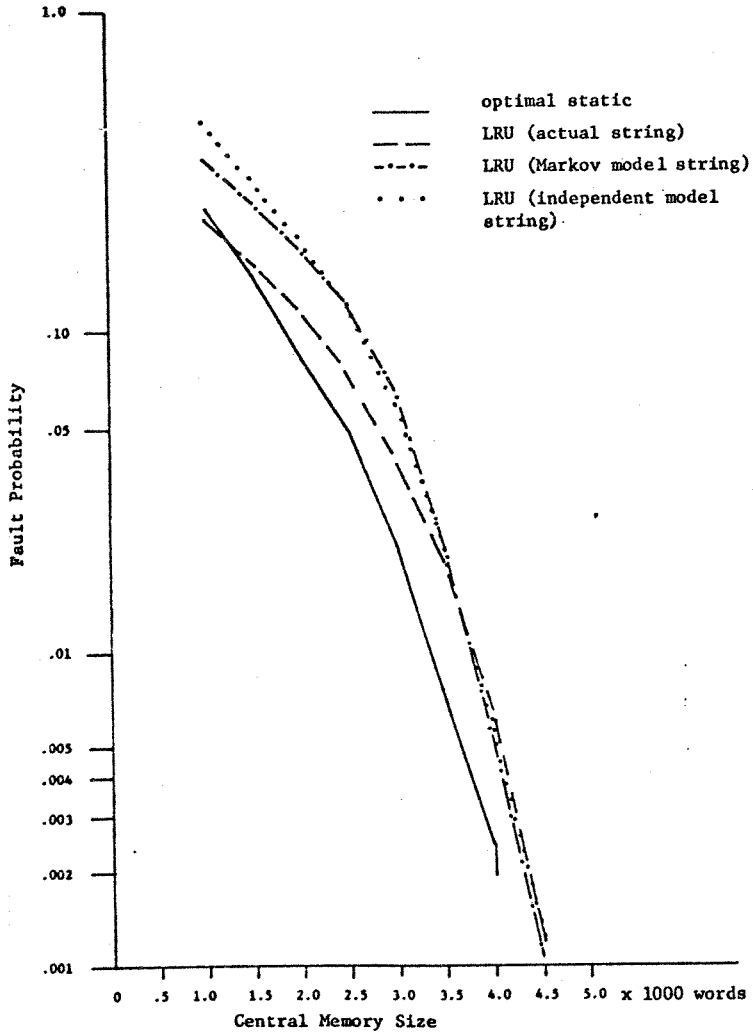


Figure 3.11: Reference Fault Probability vs. CM Size for Tape3a

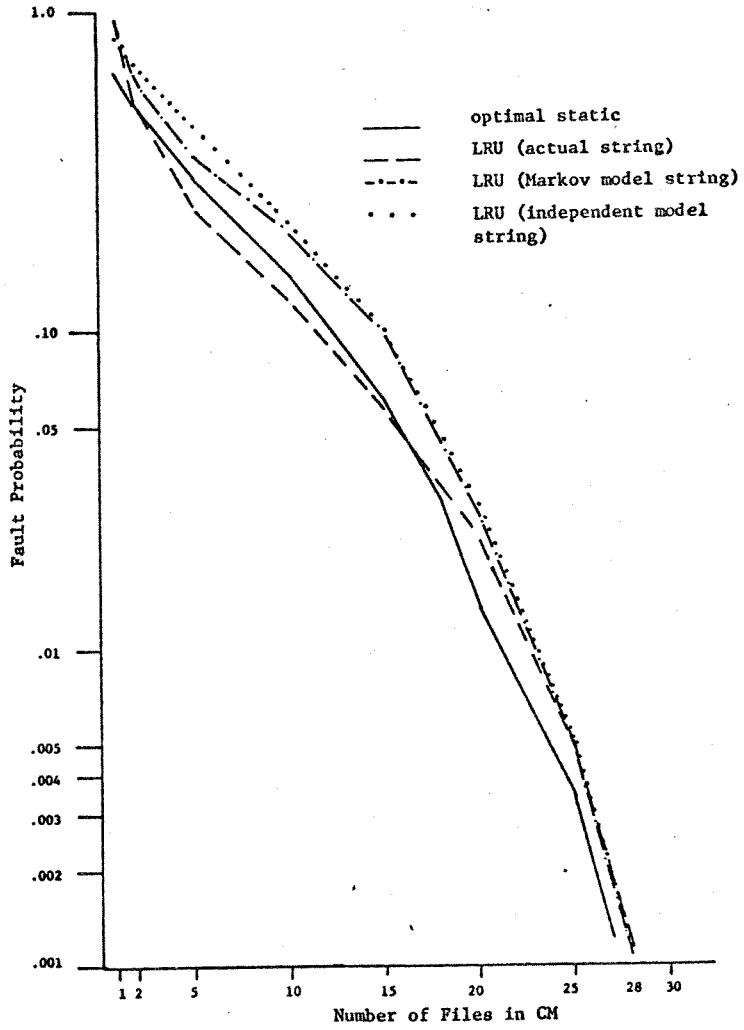


Figure 3.12: Reference Fault Probability vs. M for Tape3a

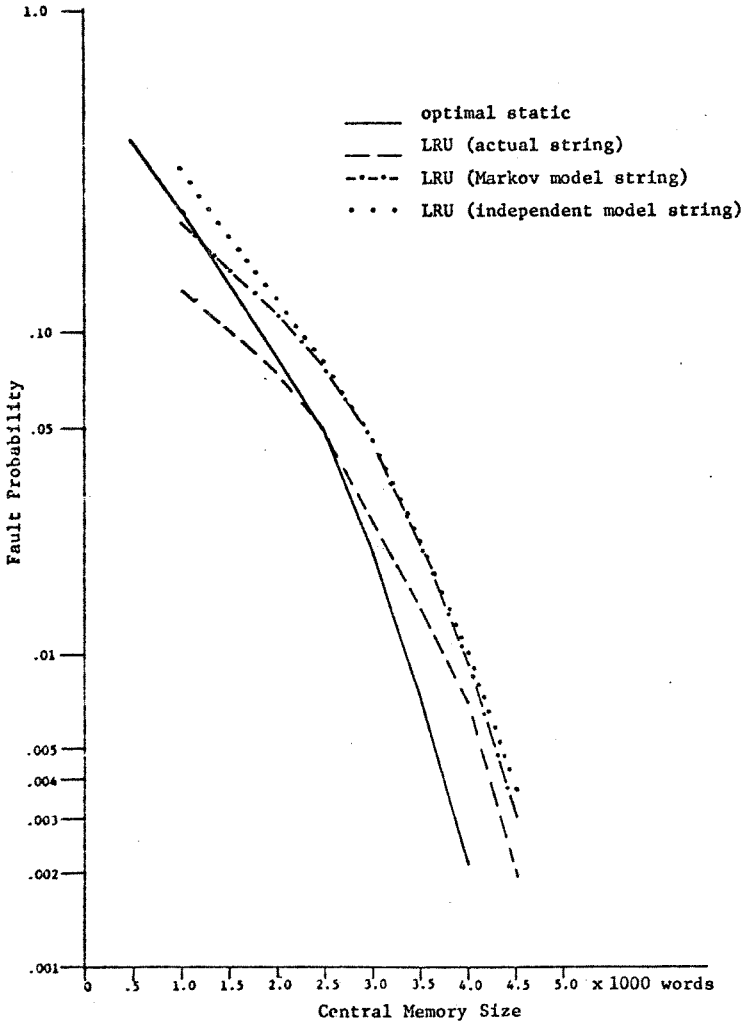


Figure 3.13: Reference Fault Probability vs. CM Size for Tape3b

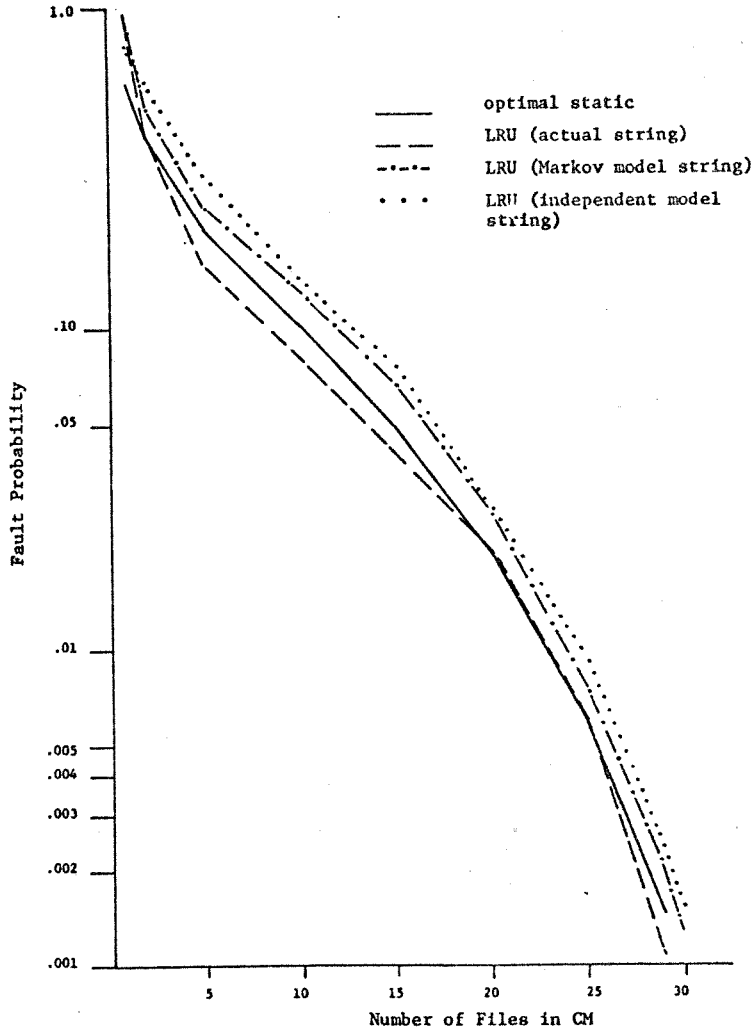


Figure 3.14: Reference Fault Probability vs. M for Tape3b