Configuring Local Area Network-based Distributed Systems

Tzong-yu Paul Lee

Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley

Ph.D. Dissertation

June 1984

The research reported here has been supported in part by the State of California and the NCR Corporation under MICRO Grant No. 1-532436-19900.

ACKNOWLEDGMENT

I wish to express my deep gratitude for the encouragement and support of my research advisor, Professor Domenico Ferrari, who introduced me to the topic discussed in this dissertation and provided me with an excellent example of a researcher and a teacher. I also benefited very much from discussions with members of the PROGRES research group. In particular, my friends Rafael Alonso, Doug Terry, and Bart Miller have always been sources of encouragement and technical advice.

I want to thank Professor Ronald Wolff and Professor C. V. Ramamoorthy for reading my dissertation and serving on my thesis committee. I also want to thank Professor Luis Felipe Cabrera for his valuable comments on my dissertation. Finally, I am indebted to Professor Jean Walrand for spending his time in discussing with me recent results in stochastic queueing networks.

The State of California and the NCR Corporation under the MICRO grant 1-532436-19900 have provided financial support for this research work. Special thanks go to both Susan Whitford and John Wakefield of the NCR Corporation, who have provided useful workload data and enlightening information on the subject of this dissertation.

Finally, I wish to thank my parents and Jeanet, for their continuous encouragement and support during my graduate work at Berkeley and the writing of this dissertation. My son, Joseph, has also been providing an unending and invaluable source of inspiration that enriches my life both inside and outside of my research work.

ABSTRACT

This dissertation studies the problem of designing the configuration of local area network-based distributed systems by a performance modeling approach. The study concentrates primarily on the interactive transaction-oriented computer systems connected by an Ethernet-like network. Major transaction types are chosen to represent the workload; each type is characterized by the demands on various computing resources. A two-step methodology is described that produces the initial configuration; this configuration is the one to which more detailed queueing network models are applied. Based on the queueing model results, the methodology then iterates to refine the configuration of the distributed system.

Various transaction types are first assigned to host systems to balance CPU utilizations. We then distribute the shared files among host systems to minimize total remote file accesses. Queueing network models for our distributed systems are constructed from a set of submodels of the host systems, of the local area network, and of the file servers if any. Model parameters are derived from our workload data, measured in an interactive transaction-oriented business system. Two examples are provided to show how the configuration of a local area network-based distributed system can be designed by using our workload data.

Attempts are next made to capture in our queueing network models the difference in resource demands during the preparation phase and the execution phase of a typical transaction. As a result, we introduce the special class of *phase-free* queueing network. It is shown that a general product-form queueing network can be reduced to an equivalent phase-free product-form queueing network. It is also shown that the per-class throughputs, mean queueing time, and mean queue length in the original network can be calculated from the values of the aggregate indices of the phase-free network.

Lastly, a special class of local area network-based distributed systems, that of the workstation-based systems, is examined in some detail. Workload clustering is proposed as a method for reducing the number of chains in the model to make it more tractable mathematically. File server design issues are investigated, and design guidelines are recommended based on workload data and performance goals.

TABLE OF CONTENTS

CHAPTER 1

Introduction

1.1. The Coming Era of Distributed Computing

Progresses made in microelectronics since the early 1970's have revolutionized the whole computer industry, and even created a new market, that of personal computers [Kay77]. The 16-bit microprocessors used in many systems today have a power equivalent to that of many early minicomputers. It will be only a few years before the 32-bit microprocessors are used extensively in new microprocessor-based products. The increasing density and speed of the semiconductor memory chips has also helped the development of small and powerful computer systems. It is very common to employ 64K-bit memory chips in today's computer systems. Meanwhile, the 256K-bit memory chips of the next generation are almost ready to come out of research laboratories. Technological advances in peripherals such as bit-mapped displays, high-density disks, and easy-to-carry floppy disks have all together made this revolution possible.

On the other front, the communication industry has paralleled the progress and development of the computer industry [Tan81]. The commercial use of satellites provides unprecedented bandwidth and speed for data communication, as well as for the more traditional voice and video communications, across large geographical distances. Packet switching networks built on top of the existing telephone networks provided by public carriers open up a convenient and inexpensive way for information exchange. More recently, the technology of local area networking [Cla78] has been introduced to provide high bandwidth and multiaccess capabilities to computer systems and users located within several thousand feet of each other. This advance is of great value to the office environment and to neighborhood communities.

The proliferation of small, inexpensive, yet very powerful computers together with new communication technology makes distributed computing a viable solution for data processing [Sch78, Zie79]. New sectors in the computer industry are also emerging : office automation with workstations [Bec82], and home and personal computing [Wil83, Tho84]. The trend towards computerization in both office and home will greatly influence the way of life in the coming century.

1.2. Our Focus - Configuring Local Area Network-based Distributed Systems

Technological progress in microelectronics and data communications has presented many new challenges to computer researchers. The structure and the behavior of distributed systems are considerably more complex than those of single-machine systems; hence, the need for design methodologies and evaluation tools is even more evident than it is in the latter case.

We are interested primarily in the design of local area network-based distributed systems configurations. The rest of this section describes the general environment that our research efforts have assumed. The communication medium that allows exchange of information between computer systems is a local area network [Cla78, Thu80]. There are many alternative technologies that can provide the basis for local area networking; we have only assumed for such network that the unit of information exchange at the network level is a packet or datagram, and that the network has some kind of multiaccess protocol to a broadcast medium [Eth80, Tan81]. The computer systems connected by this network are homogeneous in the sense that they have the same interface to the file system. In other words, we require all computer systems connected to the network to have the same mechanism for remote file access. Although the architectures and technologies of the CPUs in the various computer systems can be quite different, we assume that the CPU power of each computer system is known. Different implementations of a given architecture by the same vendor are often in this so-called homogeneous category since the rankings of their powers are usually readily available. A stricter definition of homogeneity is needed if we allow task re-assignment among computer systems in the network. Here, the same result must be obtained regardless of which computer system executed the re-assigned task. Machines with incompatible floating-point formats and machines with different representations of binary integers typically fail to meet this stronger homogeneity requirement. However, we have assumed that tasks are not reassigned in our system.

A general description of the workload within an interactive environment is of course required. Users are connected to computer systems through terminals and interact with the computer system by entering data and commands. Knowledge of the detailed nature of a task is not necessary; however, a description of the resource demands of each individual task is needed. These tasks share some data files. A simple characterization of file sharing is essential. Files are stored on disks which may be distributed among the computer systems connected to the network or may be grouped together in a file server [Isr78].

1.3. Related Work in Distributed Systems and Models

We shall review some of the related work in the area of distributed systems. The definition of distributed system adopted here is roughly that of Enslow's [Ens78]. A distributed system consists of multiple autonomous computer systems physically connected by a network medium. An operating system with some level of transparency manages the resources, possibly with some degree of decentralized control. We divide this section into two parts : experimental systems and models of distributed systems.

1.3.1. Experimental Systems

Many proprietary distributed systems have been built in prototype form and experimented with, but the detailed descriptions of these systems are not in the public domain. We can only survey some of the distributed systems described in the literature. The Xerox Palo Alto Research Center has engaged in many of the pioneering studies of, and experimentations with, distributed systems. The Xerox researchers have taken the client-server viewpoint based on a functional decomposition of a distributed system [Isr78, Mit82], and many interesting applications have been developed. Grapevine is a system

providing message delivery, resource location, authentication, and access controls [Bir82]. Violet is an experimental decentralized system that allows its user to build distributed applications [Gif81]. A calendar system was an early example of such applications. Clearinghouse is a decentralized agent for locating named objects in a distributed environment [Opp83].

The Cambridge file server employed in the Cambridge distributed system uses the Cambridge ring as the communication medium, and a comparison with the Xerox distributed file system is presented in [Mit82]. The FELIX file server built by Bell Northern Research is designed to support a variety of file systems, virtual memory, and database applications [Fri81]. A distributed version of UNIX based on the functional separation of file access mechanism and process execution domain is described by Luderer et al [Lud81]. This distributed system is a tightly coupled one with high speed virtual circuits as its communication vehicles. A simple star-shaped microcomputer-based distributed system running UNIX has been described by Pechura [Pec81]. SODS/OS, a distributed operating system for IBM Series/1 computers, offers a hierarchy of services provided by the local and network operating systems [Sin80].

Almost every university has some research activities in distributed systems. The LOCUS system built at UCLA is a distributed UNIX-based operating system with replicated files and a high degree of transparency [Pop81, Wal83]. The JADE project at the University of Calgary is essentially a prototyping environment for developing distributed system software and applications [Wit83]. The Eden project at the University of Washington has the same goal along with some hardware system development [Laz81]. The Athena project at MIT is noted for its eventual size and its application to undergraduate education [Bra84]. This is only a small sample of the research projects at various universities.

Experience with a large distributed banking system connecting 10,000 Bank of America terminals across the state of California has been reported by Good [Goo83].

1.3.2. Models of Distributed Systems

There are many aspects in the area of modeling that are related to the design of distributed systems. File placement algorithms have been studied within many different contexts, and such studies have been described by many authors; some of the best known are [Chu69, Cas72, Fos81, Wah84]. The survey by Dowdy and Foster covers and compares a dozen of models in this area [Dow82]. Variations on the theme include taking into account the number of file copies, the disk capacity constraints, and various cost models for file retrieval, update, and communication overhead. A variety of mathematical programming techniques are also covered in that survey. Strategies to perform task allocation in distributed environments have been studied quite extensively, and two recent investigations based on network flow graph models are reported in [Chu80, Wu80]. Many other studies combine task allocation, file placement, and other related resources allocation problems in one single model [Mah76, Mor77, Tri79, Den81]. These studies tend to be more comprehensive than those mentioned previously.

Realistic and comprehensive models addressing the configuration design problem are very few, especially those based on measurements of real workloads. Models constructed

specifically for existing systems are discussed by Acker and Seaman in the context of a feasibility study [Ack82] and by Deitch in a capacity planning context [Dei82]. Goldberg et al constructed a validated model for the LOCUS distributed system at UCLA and are reportedly working on the tools for the design of distributed systems configurations [Gol83].

1.4. The Contributions of this Research

The major emphasis in this research is on devising a practical methodology for configuring distributed systems subject to a given workload and given performance constraints. A characterization for the workload is proposed so that actual data can be collected from existing systems or can be extracted from some higher level workload description. This characterization and the performance indices adopted in our study are discussed in Chapter 2. The first part of the methodology allocates computing resources, and distributes the workload so as to balance CPU demands among various host systems and to minimize total remote file accesses. The second part of the methodology uses queueing network models to verify performance requirements and to estimate delays due to resource contention which are fed back into the first part of the methodology for possible iteration. Instead of formulating and solving a set of simultaneous equations or some mathematical models, the iterative methodology proposed in this dissertation allows the system designer to examine the intermediate configurations, to interpret the results, and to validate the assumptions made at the various steps of the methodology. The methodology can be easily modified to satisfy special constraints which do occur often in actual configuration design. The two parts of the methodology are described in Chapters 3 and 4 respectively.

We shall demonstrate through examples the tradeoffs the designer can make, and the insights that can be learned. For example, in Chapter 4 we show how to evaluate file system organizations, that is, how to choose between a distributed file system and a file server-based system, given a workload description and a general system description. More sophisticated models based on multi-phase workload characterizations are constructed in Chapter 5 to reflect somehow the distribution of resource demands of the workload. As a result of this queueing network modeling effort, we are able to characterize a special family of queueing networks having an equivalence relationship with some simpler queueing networks. This provides not only an intuitive explanation for some earlier work in this area, but also simplifies the computational efforts in getting the equilibrium state distribution for a queueing network. The notions of balanced file server and of system-wide load measure given in Chapter 6 for a workstation-based environment provide a system designer with a set of guidelines and hints as to where measurements should be collected and improvements should be attempted.

Our methodology should be applicable to distributed relational database systems, if a proper interpretation is given to the model parameters. Most relational database systems implement relations as files [Sto76, Wil81]. In this case, a database consisting of a set of relations can be viewed as a collection of files with record (tuple) structure. Applications built on top of the database are essentially well-defined transactions accessing and modifying a subset of relations, that is, of files. This description transforms the problem into the one we study in this dissertation, and makes our methodology perfectly applicable

to the distributed relational database system case in the appropriate environment.

We do not specifically address the case of heterogeneous systems in a local area network environment. This case is inherently more complicated from many functional viewpoints; however, in the context of configuration design, adding a set of transaction and file assignment constraints due to the heterogeneity of the distributed system is probably a simple approach to take in modifying the configuration design methodology proposed in this dissertation.

We are only able to validate models of single host system against measurement data from an existing system. Models of distributed systems similar to ours have been used in [Gol83], and the authors of that paper claim to have validated them. However, the queueing network models of distributed systems need substantially more validation and calibration work than the one done so far. With this premise, our models are better suited for a study of comparative nature, e.g., one comparing and selecting the best design among several configuration alternatives, than for a study of predictive nature. Before substantial validation work is performed, however, care must be exercised in using and interpreting directly the absolute values of the performance measures obtained from the models.

CHAPTER 2

Workload Characterization and Performance Indices

2.1. The Context of the Study

The workload to be used in our study was measured in a medium-size interactive transaction-oriented business system [Whi83]. Such systems are used mainly in the wholesale industry. A typical configuration consists of one central processing unit (CPU) with half a megabyte of memory, six terminals, two line printers, two cassette drives, and four disk platters. There are many applications that run on these systems and share a common set of files, typically two dozen in number. Commonly found applications are order processing, inventory control, and sales analysis.

In such systems, the notion of "user" which is found, for instance, in a program development environment, does not exist. Instances of the execution of application programs are called transactions. Transactions are equivalent to the "users" in that they demand and share various resources. The way transactions interact with each other is generally known in advance. In particular, we know the set of files they share and the degree of sharing that might exist. Since the set of shared files is rather static and is limited in its size, the directory of shared files is rarely changed, and, thus, very stable. It is likely that this characteristic will be carried over to the distributed version of such transaction-oriented business systems.

Unlike what happens in business systems, users in typical research and development environments show relatively little sharing of user files in their activities [Por82]. When a user file is shared, the sharing usually occurs across a much larger time span, e.g., several days. Notice that the set of common operating system files related to various kernel utilities are shared and used by all users, and is generally referred to as *system* files. The issue of directory structure is a rather complicated one in a research and development environment, where sharing is dynamic although a relatively infrequent phenomenon [Wal83]. In fact, the directory of files is a specialized name server in a distributed environment, and the issues involved in designing distributed name servers are not completely understood yet [Ter84]. The structure and the modeling of a file directory for our particular workload environment is discussed in Section 4.1.

2.2. Workload Characterization

We shall divide this section into two parts : basic resource demands, which can be satisfied by multiple instances of the same type of resource, and spatial characterization of file sharing.

2.2.1. Resource Demands

The workload of an interactive transaction-oriented business system can be characterized by the number of major transaction types, the resource demands by each major transaction type, and the relative throughputs of these transaction types during some typical interval of operation. The relative throughputs of these transaction types are assumed to be expressed in percentages, and sum up to unity.

We shall use this characterization of a transaction type as a template of resource demands for user jobs or transactions. A user job or transaction is an instance of a transaction type which is initiated from a user terminal. The sequence and the amounts of resource demands are those which characterize the corresponding transaction type. We shall use terminals, transactions, and user jobs interchangeably when no ambiguity will be possible. Each transaction uses a few temporary (private) scratch files during its execution while sharing some permanent files with other instances of the same and other transaction types in execution. It is this collection of shared files that are our main concern in this study.

The resource demands of a transaction of a given type can be further characterized by the average values of the following quantities :

(1) the number of user interactions (or commands);
(2) the user think/preparation time per interaction;
(3) the total CPU time demand;
(4) the number of physical disk accesses to be made to each shared file;
(5) the number of display outputs.

Five major transaction types are characterized and summarized in Table 2.1. For simplicity, we name these five types of transactions T1, T2, T3, T4, and T5; these symbols correspond to the following types : order entry, new order release, shipment processing, stock receipts, and invoice printing, respectively.

Note that a display output is the transfer of a character string by a transaction process through an input/output (I/O) channel, usually an asynchronous communication line, to a user display terminal. There are generally a number of display outputs during

Table 2.1 Workload Data

Transaction Type	Relative Throughput	Number of Interactions per Transaction	Think Time for each Interaction (seconds)	CPU Time (seconds)	Number of Physical Disk I/O's	Number of Display Outputs (characters)
T1	20%	47	1.0	16.78	179	217 (15)
T2	20%	11	10.0	36.20	834	196 (25)
T3	15%	6	1.5	28.46	100	27 (20)
T4	25%	12	2.0	8.72	42	85 (15)
T5	20%	7	0.5	34.00	1030	29 (15)

an interaction. Notice that we have separated the input from the output functions for a typical user terminal. This is very convenient for representing the user operations of the computer system as long as there is no overlap between user input and display output at each terminal. Disks are sectored; a physical disk I/O transfers one sector of data.

The data listed above can be collected by a suitable hardware monitor and/or by appropriate software instrumentation. The values of the characterizing parameters for each transaction type generally remain invariant with respect to changes in the underlying structure of the distributed system; for example, they are usually the same for a system with file servers as for a system with a distributed file organization. However, there are overheads involved in getting a resource demand serviced at a remote site. These overheads range from simple, easy to quantify ones, such as the times spent for network transmission and communication protocols, to more complicated ones, such as the queueing delays due to contention at remote sites. Overheads must be taken into consideration together with the resource demands of invariant nature when modeling a distributed system.

Some of the workload information discussed above is used in the next chapter to determine the distribution of transaction types and shared files over the hosts. Input parameters to the queuing network models described in Chapter 4 are also estimated from some of the quantities in our workload characterization.

2.2.2. File Sharing

There are twenty files that are shared among the five transaction types we have described in the previous subsection. For simplicity, we have named the set of shared files F1 through F20. Table 2.2 gives us the estimated probabilities that a particular file be accessed by a transaction of a certain type *given* a file access is required by the transaction type in question. This is a conditional probability, and the entries in each column sum up to unity. In order to use this table to compare rates of file access among shared files, the actual access rate to a shared file from a transaction type should be weighted properly by the total file access rate of the corresponding transaction type. An entry containing ϵ indicates that the corresponding file is used by the corresponding transaction type, but the (conditional) probability of accessing it is comparatively negligible. For the purpose of modeling, we can ignore these entries.

Locks are used to allow concurrent sharing of files while preserving the semantic correctness of multiple parallel transactions. In the measured system, lockable granules are the physical sectors of a file, so that file sharing will not drastically degrade transaction throughputs [Rie77].

In order to characterize the spatial aspect of the sharing phenomenon, we propose two metrics based on a conceptually simple user-object sharing graph. Characterizing the temporal aspect of sharing is much more difficult, and we will not attempt to do that here. Let us assume that there are k objects shared by n users in a computing environment. Given a fixed time interval, we represent the intensity of sharing between user i and object j by the edge label t_{ij}, which is the number of accesses by user i to object j during that interval. Without loss of generality, we can assume that the t_{ij}'s are between 0 and 1 by properly normalizing the intensities of all edges. One such example is

Table 2.2 Conditional Probabilities for File Access

Shared File	Transaction Type				
	T1	**T2**	**T3**	**T4**	**T5**
F1	ϵ	ϵ	ϵ	0	0
F2	0.033	0.104	0.081	0.052	0.075
F3	0.050	0	0	0	ϵ
F4	0.310	0.453	0.551	0	0.508
F5	0.084	0.081	0.163	0	0.113
F6	0.201	0.226	0.204	0.175	0.302
F7	ϵ	ϵ	ϵ	0.035	0
F8	ϵ	0	ϵ	0	0
F9	ϵ	0	ϵ	0	0
F10	0.042	0	0	ϵ	0
F11	0	0	0	ϵ	ϵ
F12	0	0	0	0.157	ϵ
F13	0.025	0	ϵ	0.052	0
F14	ϵ	0	ϵ	0	0
F15	0.252	0	0	0	ϵ
F16	ϵ	ϵ	ϵ	0	0
F17	0	0.133	0	0	ϵ
F18	0	0	0	0.140	ϵ
F19	0	0	0	0.333	ϵ
F20	0	0	0	0.052	ϵ

given in Figure 2.1 for a bipartite graph with two sets of nodes, users and objects, and edges exist only between pairs of nodes from both sets. We define two metrics, the mean m and the variance var, for the sharing graph. In order to apply them to a general class of problems, these metrics are normalized so that they lie between 0 and 1. Characterization of sharing and design decisions can use these metrics, as we shall show later.

We define the total normalized intensity S_j of access to an object j as the average of intensities from all users, as shown in (2.1). We can interpret S_j as a random variable.

$$0 \leq S_j = \frac{1}{n} \sum_{i=1}^{n} t_{ij} \leq 1 \tag{2.1}$$

Recall that the t_{ij}'s are between 0 and 1, and, thus, we have the inequalities in (2.1). The mean m and the (normalized) variance var are defined in (2.2) and (2.3).

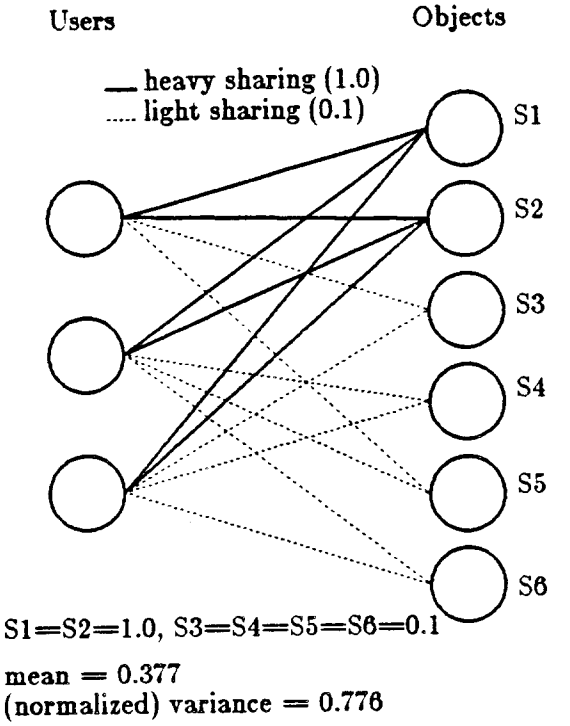

Figure 2.1 Spatial Characterization of a User-Object Sharing Graph

$$m = \frac{1}{k} \sum_{j=1}^{k} S_j \tag{2.2}$$

$$var = 4\,(\frac{1}{k} \sum_{j=1}^{k} S_j^2 - m^2) \tag{2.3}$$

It is obvious that m is (tightly) bounded by 0 and 1. We shall now show that var is also (tightly) bounded by 0 an 1. Notice that the factor 4 is present in (2.3) in order to normalize the variance var.

That var is (tightly) bounded by 0 and 1 will be shown in two steps. First, we show that "moving" to 0 one by one those S_j's which are less than the mean will increase the variance as a result. A similar argument applies to those S_j's that are larger than or equal to the mean if we "move" them to 1 one by one. Formally, let us assume that $0<S_i<m$, and move S_i to 0. For this new distribution, we have a different variance as well as a different mean. However, the new variance var_{new} is greater than the old variance var_{old} as shown below.

$$var_{new} = \frac{1}{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} S_j^2 - (\frac{1}{k} \sum_{\substack{j=1 \\ j \neq i}}^{k} S_j)^2$$

$$= \frac{1}{k}((\sum_{j=1}^{k} S_j^2) - S_i^2) - (\frac{1}{k}((\sum_{j=1}^{k} S_j) - S_i))^2$$

After some algebraic manipulation, we have

$$var_{new} = var_{old} + \frac{S_i}{k^2}(2(\sum_{j=1}^{k} S_j) - (k+1)S_i) \tag{2.4}$$

Since $(k+1)S_i = S_i + kS_i < S_i + \sum_{j=1}^{k} S_j < \sum_{j=1}^{k} S_j + \sum_{j=1}^{k} S_j = 2\sum_{j=1}^{k} S_j$, the term in parenthesis is greater than 0. Thus, we have shown that the variance increases when one sample point is moved to the extreme (0 in this case).

In the second step, we show that the maximum variance is achieved within this class of "polarized" distributions when we have an equal number of S_j's at both extremes, i.e., 0 and 1, and there are no S_j's assuming intermediate values. For the general "polarized" distribution, assuming that p S_j's equal 1 and the remaining $k-p$ S_j's equal 0, the variance $var_{polarized}$ is given by

$$var_{polarized} = \frac{1}{k^2} p(k-p) \tag{2.5}$$

It is easy to show that this variance $var_{polarized}$ is maximum when $p = \left\lfloor \frac{k}{2} \right\rfloor$. In particular, the value of this maximum is 1/4 when k is an even number. An example can be readily constructed with variance 1/4; therefore, we normalize it accordingly, as shown in (2.3).

We can apply the pair of metrics (m,var) to the problem of deciding how to distribute shared files to host systems so that local accesses to shared files are maximized by some criterion. If the mean m of a given sharing graph is large, we know that it is nearly impossible to allocate (single-copy) files to host systems such that remote accesses are minimized. Putting all shared files in a central depository such as a file server is probably a good solution. On the other hand, if m is small, we have a very good chance in allocating files so that most accesses are local to the host system. It is when m is neither large nor small that we need a second metric like var. When the variance var is small, the distribution of shared files to host systems could prove to be a viable solution. However, if we have a very large variance, a few files are shared heavily by almost all host systems and the others are barely shared. It will still be difficult to distribute these heavily shared files satisfactorily over host systems as we demonstrate in the second example of Chapter 4. A hybrid approach, consisting of some centralized files and some distributed files, may be appropriate with this particular type of sharing behavior. The thresholds for these metrics are application dependent.

2.3. Basic Analysis of the Workload Data

A simple analysis of workload data is carried out here with appropriate assumptions; some of the results will be used later as parameters for various models. Throughout this study, we have assumed that the devices involved have the following characteristics. Disks are sectored, and each sector contains 512 bytes of data; a disk access takes about 32.07 milliseconds, which is the sum of average seek time, mean latency time, and data sector transfer time. The bandwidth of the local area network is either 10 Mbit/sec [Eth80] or 24 Mbit/sec [Whi83]. Some kind of multiaccess protocol is assumed on this Ethernet-like communication medium [Met76]. Display output service times depend on the speed of the I/O channel which terminals are connected to. Normally, we assume a 9600 baud asynchronous line with 10 bits per character (byte). Therefore, transferring 15 characters requires about 15.62 milliseconds.

We first compute the elapsed time for each transaction type by assuming a uniprogramming environment, i.e., an environment without queueing delays for services rendered by the various resources and at various processing stages. The elapsed time includes user think/preparation time at the terminal. The inverse of this time will be the best possible throughput for each instance of the corresponding transaction type. These best elapsed times and throughputs are shown in Table 2.3.

It is also of interest to compute the resource demands on a per interaction basis. The best possible response times are then computed by the same uniprogramming assumption. The results are shown in Table 2.4. These constitute the lower bounds for the respective transaction types. Note that we have assumed in Table 2.4 that the resource demands for each interaction are the same on the average. In Chapter 5 we shall examine the effects on the queueing network models of the partial relaxation of this assumption.

In Table 2.5, we have assumed a simple model of process execution. The process initiated at the terminal goes to the CPU for a "burst" of CPU time, until it needs to perform an I/O to the disk subsystem or an output to the display device. After the I/O operation, the process resumes at the CPU to get another burst. After the last CPU burst, the process finishes its execution and returns to the user terminal for the next interaction. The various branching probabilities are calculated by assuming geometric

Table 2.3 Best Elapsed Time and Throughput

Transaction Type	Best Elapsed Time (seconds)	Best Throughput (per minute)
T1	72.905	0.822
T2	178.049	0.336
T3	41.229	1.455
T4	35.393	1.695
T5	70.984	0.845

Table 2.4 Resource Demands per Interaction

Transaction Type	CPU (seconds)	Disk I/O N_{disk}	Display Output N_{disp}	Best Response Time (seconds)
T1	0.357	3.80	4.61	0.551
T2	3.290	75.81	17.81	6.185
T3	4.743	16.66	4.50	5.371
T4	0.726	3.50	7.08	0.949
T5	4.857	147.14	4.14	9.640

Table 2.5 CPU Bursts and Branching Probabilities with Display Output

Transaction Type	CPU Burst (ms)	Probability to Disk	Probability to Display Output	Probability to Terminal
T1	37.93	0.403	0.489	0.108
T2	34.77	0.801	0.188	0.011
T3	214.03	0.751	0.203	0.046
T4	62.69	0.302	0.611	0.087
T5	31.89	0.966	0.027	0.007

distributions for the number of visits to each service center, so that the means of these distributions match the expected number of disk accesses and display outputs as obtained from the measurements. Specifically, the branching probabilities to disk, display output, and terminal are :

$$P_{disk} = \frac{N_{disk}}{N_{cpu}}, \; P_{disp} = \frac{N_{disp}}{N_{cpu}}, \text{ and } P_{term} = 1 - P_{disk} - P_{disp}, \tag{2.6}$$

where $N_{cpu} = 1 + N_{disk} + N_{disp}$, N_{disk}, and N_{disp} can be found in Table 2.4.

Table 2.6 CPU Bursts and Branching Probabilities without Display Output

Transaction Type	CPU Burst (ms)	Probability to Disk	Probability to Terminal
T1	74.37	0.791	0.209
T2	42.83	0.986	0.014
T3	268.57	0.943	0.057
T4	161.33	0.777	0.223
T5	32.78	0.993	0.007

There are cases in which we assume that terminals are locally attached to the computer system by a high speed communication line, and, thus, display service times can be ignored and not modeled. The mean durations of CPU bursts and the branching probabilities under these assumptions are shown in Table 2.6. The branching probabilities are calculated with equations similar to (2.6). We shall use this set of model parameters for the workstation-based distributed systems in Chapter 6.

2.4. Performance Indices

There are two specific types of performance indices of interest in this study. One is the overall system throughputs for the various transaction types, the other is the average response time to user commands/inputs for each transaction type. These two indices can be employed to compare designs with different configurations, and may be assigned as performance requirements to be satisfied by the design. The designer must properly configure the system to meet these requirements while keeping its cost at a minimum.

It should be pointed out that the throughput index and the response time index are actually vectors, in which each element corresponds to a transaction type. Since the comparisons of multiple indices or variables are somewhat cumbersome, a judicious choice of the presentation technique is essential.

Other system-related performance indices, such as the utilizations of critical resources, may be useful for bottleneck detection and capacity planning. It is important for a system designer and modeler to examine utilizations, queue lengths, queueing times, and throughputs of various resources. Such evaluation approaches cannot be easily automated, and the details of their implementation will depend on the expertise of the modeler and designer. More importantly, it is usually difficult to incorporate the cost structure of the proposed configuration into a performance model, and designers will have to bridge this gap to reach a cost-effective decision in the design and planning process. We shall demonstrate this point with a few examples in the ensuing chapters. For instance, the first example given in Chapter 4 shows that a moderate increase in CPU speed may be more cost effective than the addition of another host system to a distributed system. This tradeoff can be carefully evaluated using the simple methodology presented in this dissertation before a major procurement or upgrading decision is made.

CHAPTER 3

Allocating Computing Resources and Distributing the Workload

Numerous models have been proposed for the problem of allocating computing resources so as to optimize some performance-based (or performance-oriented) objective function [Tri80, Dow82]. These computing resources include computing power (CPU), storage capacity (Disk), or communication network capacity (Link). We have associated terminals with user jobs of various transaction types; thus, terminals can be viewed as representing the workload, and are subject to distribution as well. Intuitively, the goal of the allocation/distribution process in the context of distributed systems design is to allocate computing resources to match the needs of the workload, which is also distributed so that response times and throughputs are optimal, or, more simply, that the requirements in terms of such indices are met. It should be clear that the dollar cost is always either an objective function subject to minimization or a constraint. Without presenting a detailed cost model, we study and compare only similar types of configurations, so that costs are similar and can all satisfy the same kind of cost constraints.

The existing models are rarely comprehensive enough to be usable in configuring the local area network-based distributed systems as a whole; and, even if we do have such a model, the chances of its being computationally tractable are very remote indeed. For this reason, we propose an iterative step-wise methodology for configuring local area network-based distributed systems. This methodology makes the problem of configuration design mathematically tractable with various kinds of design constraints, and allows system designers to parameterize the design alternatives more easily for comparisons at various steps. The first half of the methodology deals with the allocation of resources and the distribution of the workload in the distributed system by mathematical or graph-theoretic models, so that CPU demands at various host systems are balanced and total remote file accesses are minimized. This first half is presented in this chapter. The second half of the methodology uses the proposed configuration as a basis for the queueing network models which, for the most part, factor in the stochastic aspects of the workload and the interactions and resource contentions among transactions. More realistic data or indices can be obtained from queueing network models. Specifically, the queueing network models provide an estimate for the queueing delays due to contention for resources among various transactions. This new estimation of queueing delays and transaction throughputs may require modifications to the computing resources allocation and workload distribution that was carried out in the first half of the iterative methodology if necessary. This part is discussed in Chapter 4.

3.1. Confining the Problem

The type of distributed system being studied here consists of various numbers of host systems interconnected by a local area network. It is sometimes advantageous to consider connecting terminals to more than one host system through the local area network or some dedicated high-speed data concentrator. Nevertheless, we confine the study to systems in which terminals are connected to specific host systems. Note that, however, we do investigate one type of system that connects all CPUs to the same disk subsystems, i.e., file server-based distributed systems [Mit83]. This type of configuration is based on the functional separation of computation and storage, with a high-speed local area network as the necessary interface.

For convenience, we divide the task into two major steps : the task of transaction type (workload) allocation and the task of (single-copy) shared file distribution. Before discussing the distribution of files, we need to investigate the problem of allocating transaction types, i.e., terminals, to host systems. The task of transaction type allocation is dealt with first, since each transaction type exhibits similar resource demands, and thus it is easier to capitalize on file access locality. We only have to consider the issue of shared file distribution in a more general type of distributed file system, i.e., that without dedicated file servers [Wal83]. Although distributing shared files over host systems is a problem which arises only in the design of a distributed file system, a similar problem arises in the design of a file server-based system if the distribution of files over several disk drives and/or over several servers is considered. However, the problem is at a different level in a hierarchical description of the system.

3.2. Step I - Allocation of Transaction Types to Host Systems

Since files have not been assigned to host systems yet, in this step of the design we assume conceptually that they are stored in a central file depository on the network (e.g., in a file server). Hence, the only load that can be meaningfully balanced at this point is the CPU load. Minimizing the remote I/O activity is the objective of the next step. Implicitly, by selecting this as the objective of the first step, we have assumed that the workload to be distributed is more CPU-bound than I/O-bound, i.e., the workload consists mostly of transaction types with relatively long CPU bursts between disk accesses like the workload we have in Tables 2.5 and 2.6.

Given the number of hosts that the distributed system to be designed will consist of, transaction types are assigned to host systems according to their expected CPU demands per unit time, properly weighted by the given relative throughputs. The CPU demand rate of a transaction type is obtained by dividing the total CPU demand by the uniprogramming execution time of one transaction. The uniprogramming execution time of a transaction is given by the sum of its total CPU time and its total I/O time, and is used here for initial placement instead of the multiprogramming execution time since this is unknown. The objective of this assignment is to balance the CPU loads among the various hosts as much as possible. The criterion we use is the minimization of the sum of squared differences between the CPU demands for each host system and the ideal average load. Specifically, if the CPU demand rate for transaction type i is d_i, and there are m transaction types to be allocated to n host systems, the ideal average load is

$$load_{ideal} = \frac{\sum_{i=1}^{m} d_i}{n} . \tag{3.1}$$

Note that this ideal average load may not be in the space of feasible solutions, and is a constant, given n, m, and d_i's. The objective function we choose to minimize is

$$\sum_{j=1}^{n} (\sum_{i=1}^{m} x_{ij} d_i - load_{ideal})^2 , \tag{3.2}$$

where the decision variable x_{ij} is 1 if transaction type i is allocated to host system j, and 0 otherwise. Other objective functions such as *maxmin* or *minmax* are possible; however, we choose the familiar least sum of squares objective function so that a larger deviation from the ideal average load would be penalized more heavily. By expanding the sum of squares, (3.2) becomes

$$\sum_{j=1}^{n} (\sum_{i=1}^{m} x_{ij} d_i)^2 - n \ (load_{ideal})^2 . \tag{3.3}$$

Recalling that $load_{ideal}$ is a constant, a less complicated but equivalent objective function is

$$\sum_{j=1}^{n} (\sum_{i=1}^{m} x_{ij} d_i)^2 . \tag{3.4}$$

Note that we generally try not to assign the same transaction type to more than one host system, since transactions of the same type exhibit the same file access pattern. This approach will ensure better locality of access to (single-copy) shared files. This would be irrelevant in file server-based systems.

3.2.1. Solution Approaches for the Transaction Type Allocation Problem

The decision problem version of the optimization problem we have just described is NP-complete even for the case $n=2$ [Cha75, Gar79]. The proof of NP-completeness involves a straightforward reduction from Karp's PARTITION problem [Kar72], and is omitted here. This problem is solvable in pseudo-polynomial time for any fixed n. We choose to employ a simple heuristic to solve the problem [Cha75], and evaluate the heuristic probabilistically. In order to do that, we present two approaches to obtaining the optimal solutions for the cases of $n=2$ and $n=3$, respectively.

Let us restate the problem and introduce some additional notation to be used in later discussions. The optimization problem consists of solving for the 0-1 decision variables x_{ij} so that (3.4) is minimized, given n, m, and the d_i's (where i ranges from 1 to m). We shall designate by S_j the total load allocated to host j.

$$S_j = \sum_{i=1}^{m} x_{ij} d_i . \tag{3.5}$$

We shall also designate by $C(z,P)$ the sum of squares obtained from a particular problem instance z and with algorithm P, that is,

$$C(z,P) = \sum_{j=1}^{n} \left(\sum_{i=1}^{m} x_{ij} d_i \right)^2 . \tag{3.6}$$

Intuitively, we can think of this problem as the one of placing a set of m items, each with weight d_i, into n bins so that the resulting weight in each bin is best "balanced" according to some criterion; in this case, so that the sum of squares is minimum.

3.2.1.1. Dynamic Programming Approach

For the case $n=2$, we adopt the dynamic programming approach [Bel62] to obtain an optimal solution. This approach results in a pseudo-polynomial time algorithm. Assume that the d_i's are integers, or have been scaled to integers if they are rational numbers. Let us define S and U as follows :

$$S = \sum_{i=1}^{m} d_i \ , \ U = \left\lfloor \frac{S}{2} \right\rfloor . \tag{3.7}$$

The function $t(i,j)$ we use in the recursion is defined as

$$\begin{aligned} t(i,j) &= \textit{True} \text{ if there is a subset in } \{d_1, d_2, \cdots d_i\} \text{ that sums up to } j \ , \\ &= \textit{False} \text{ otherwise} . \end{aligned} \tag{3.8}$$

It is clear that $t(i,j)$ can be recursively expressed as

$$t(i,j) = \textit{True} \text{ if } [\ t(i-1,j) = \textit{True}\ ] \textit{ or } [\ j \geq d_i \text{ and } t(i-1,j-d_i) = \textit{True}\ ] \tag{3.9}$$

with initial conditions

$$\begin{aligned} &t(i,0) = \textit{True} \ , \text{ and} \\ &t(1,j) = \textit{True} \text{ if } j = d_1, \textit{False} \text{ otherwise.} \end{aligned} \tag{3.10}$$

An example of this computation is shown in Table 3.1, where the boolean values in italics are obtained from the initial conditions. After the computation, we search the last row for the largest index j with a *True* entry. Taking advantage of the fact that for $n=2$ the least sum of squares is optimization-wise equivalent to

Table 3.1 An Example for the Dynamic Programming Algorithm

$\{d_1=5, d_2=4, d_3=2, d_4=1, d_5=3\}$, $U=7$								
	j=0	j=1	j=2	j=3	j=4	j=5	j=6	j=7
i=1	*True*	*False*	*False*	*False*	*False*	*True*	*False*	*False*
i=2	*True*	False	False	False	True	True	False	False
i=3	*True*	False	True	False	True	True	True	True
i=4	*True*	True	True	False	True	True	True	True
i=5	*True*	True	True	True	True	True	True	True

$$\sum_{j=1}^{2} | S_j - load_{ideal} | , \tag{3.11}$$

or simply the absolute value of the difference between S_1 and S_2 , we now have the optimal solution with the least sum of squares equal to $j^2+(S-j)^2$.

This algorithm is very easy to implement. Although generalizing it for $n>2$ is not too difficult, it is not obvious which entry in the multi-dimensional table will correspond to the optimal solution. A limited search for the optimal solution is necessary, and thus this approach may not be efficient for $n>2$.

3.2.1.2. Branch-and-Bound Approach

The branch-and-bound technique [Hor78] is employed for the case $n=3$. The basic idea of this approach is to carefully apply a backtracking algorithm such that the search tree is limited in its size during the computation. With a little care in planning, symmetrical cases, e.g., those which consist of interchanging all items between two bins, are not re-evaluated in our optimization problem. The bounding function, which guides and limits the search, is briefly discussed here.

Following the observation[1] that,

$$\text{given } \sum_{j=1}^{n} S_j = K, \quad \sum_{j=1}^{n} S_j^2 \text{ is minimum if } S_j = \frac{K}{n} \text{ for all } j,$$

we derive a bounding function for the partial sum S_j. For the case $n=3$, suppose we have a solution, e.g., from a heuristic algorithm, such that C is the sum of squares. If there is an optimal solution $C^*<C$, the following inequality must hold for any given partial sum $S_j \geq S/3$ at any point of the computation :

$$S_j^2 + 2\left(\frac{S-S_j}{2}\right)^2 \leq C^* < C, \tag{3.12}$$

where S is the sum of the d_i's as defined in (3.7) above. This leads to the simple bounding inequality (3.13) for any partial sum S_j at any point of the computation :

$$S_j < \frac{S+\sqrt{6C-2S^2}}{3} \tag{3.13}$$

Using the best available solution C, this bounding function can be easily computed and tested when S_j is changed to see whether further search on the subtree is necessary. If a leaf of the search tree has been reached, the sum of squares is computed, and the currently best solution is replaced if necessary. Obviously, the bounding functions needs be updated if a better solution is found, so that the search can be tightened and converge quickly.

The generalization of this approach to $n>3$ is straightforward. Another advantage of this approach is that the search for the optimal solution can be terminated at a pre-determined time or at any "sign" of convergence to the optimal solution.

---

[1]This result can be easily proved by using an interchange argument, or the Lagrangian's method.

3.2.2. Probabilistic Evaluation of the Heuristic Algorithm

The heuristic algorithm [Cha75] for the allocation problem works as follows :

(1) For each bin j=1 to n, initialize the partial sum S_j to zero.

(2) Allocate the largest d_i in the input set to the bin j with the smallest partial sum S_j.

(3) Update partial sum S_j , and remove d_i from the input set.

(4) Repeat steps 2 and 3 until the input set becomes empty.

In our evaluation, we assume a uniform distribution for the input sizes d_i between 0 and 2000. The results are summarized in Tables 3.2, 3.3, 3.4, and 3.5, with n=2,3 and m=5 to 13. We compare the solutions of 200 problem instances obtained from the heuristic algorithm and the optimal algorithm for each entry corresponding to an (n,m) combination.

Table 3.2 gives the estimated probability that the heuristic algorithm provides an optimal allocation solution. It agrees with our intuition that, as m increases, the chance of getting an optimal solution decreases. Particularly for smaller m's, the probability for n=2 is smaller than that for n=3 because there are generally more ways to rearrange the items, and, intuitively, a simple heuristic could deviate farther from the optimal solution.

In order to get some insight into the distribution of the "deviation" from optimality, we define such deviation $\delta(z)$ as follows :

$$\delta(z) = \frac{C(z,\text{Heuristic Algorithm})}{C(z,\text{Optimal Algorithm})} - 1 \tag{3.14}$$

It should be noted that the distributions of $\delta(z)$ in all of our evaluations lie well within the worst-case upper bound for $\delta(z)$ given in [Cha75], and are generally very narrow indeed. Tables 3.3 and 3.4 shows the fraction (in percentage) of the problem instances whose $\delta(z)$'s are smaller than 0.2% and 0.4% respectively. Notice that in Tables 3.3 and 3.4, when m is small and is *not* an integral multiple of n for n=3, the fraction of the problem instances that is within the given δ from the optimal solution tends to be small. Overall, there is little dispersion and, thus, small variance, in the distributions we have obtained.

Table 3.5 shows the minimum deviation δ, in percentage, that will include 90 percent of the problem instances. This value is also referred to as the 90%-fractile or 90-th percentile value. Generally speaking, as m increases, the deviation measure decreases; thus, the minimum δ in Table 3.5 decreases.

Table 3.2 Probability of Getting an Optimal Solution by the Heuristic Algorithm

	m=4	m=5	m=6	m=7	m=8	m=9	m=10	m=11	m=12	m=13
n=2	1.000	0.630	0.455	0.215	0.240	0.070	0.080	0.045	0.025	0.025
n=3	1.000	1.000	0.905	0.550	0.330	0.150	0.065	0.030	0.010	0.005

Table 3.3 Fraction (in Percentage) of the Problem Instances with δ Less Than 0.2%

	m=4	m=5	m=6	m=7	m=8	m=9	m=10	m=11	m=12	m=13
n=2	100.0	79.0	78.0	79.0	89.5	89.5	97.0	99.0	99.0	99.5
n=3	100.0	100.0	95.0	81.0	67.5	75.0	82.0	79.0	91.0	96.0

Table 3.4 Fraction (in Percentage) of the Problem Instances with δ Less Than 0.4%

	m=4	m=5	m=6	m=7	m=8	m=9	m=10	m=11	m=12	m=13
n=2	100.0	84.0	88.0	90.0	95.5	98.0	99.5	100.0	99.5	100.0
n=3	100.0	100.0	97.0	86.5	79.0	89.0	93.5	97.0	97.5	98.5

Table 3.5 Minimum δ (in Percentage) Covering 90% of the Problem Instances

	m=4	m=5	m=6	m=7	m=8	m=9	m=10	m=11	m=12	m=13
n=2	0.000	0.822	0.472	0.400	0.206	0.210	0.095	0.086	0.058	0.044
n=3	0.000	0.000	0.018	0.556	0.705	0.433	0.280	0.290	0.18	0.152

3.3. Step II - Modeling File Distribution

After allocating the transaction types, we are now ready to distribute the shared files to various host systems. This step does not apply to the file server-based distributed systems since all shared files will be stored in the file server. We assume that temporary and private files are stored locally, i.e., on the host where a transaction is actually executed. In the context of local area network-based systems, we assume that there is complete freedom in allocating the total storage capacity required to various host systems, so that there is no explicit capacity constraints associated with each host system in the file distribution problem. Nevertheless, capacity constraints on the algorithm complexity are taken into account in Section 3.3.2.1 for completeness.

3.3.1. Graph-theoretic Model

As shown in Figure 3.1, we construct a bipartite graph model with two types of nodes; a node on the left-hand side represents a transaction type, and a node on the right-hand side represents a shared file. The edge between a transaction type node and a shared file node is weighted by the rate of physical file (disk) accesses from the transaction type in question to the corresponding shared file. The rate of file accesses is obtained by dividing the number of file accesses by the uniprogramming execution time of an average transaction of that type. The uniprogramming execution time of a transaction is used here for initial placement instead of the multiprogramming execution time since this is unknown. The goal of shared file distribution/clustering is to minimize total intercluster, i.e., remote, file accesses.

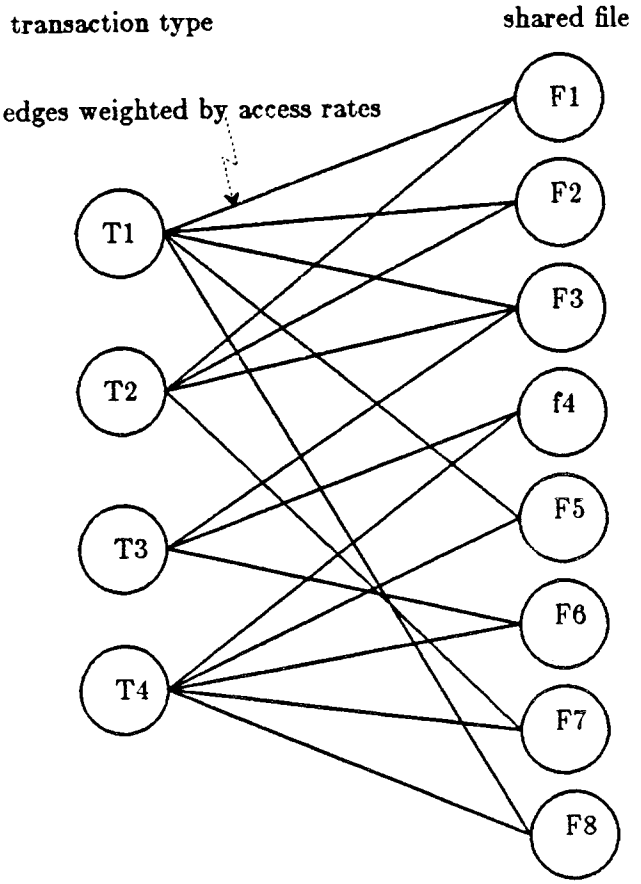

Figure 3.1 A Bipartite Graph Model of File Access

Now, we can use the assignment of transaction types to host systems determined in the previous step to coalesce some of the transaction type nodes and their associated edges. For example, if transaction types 1 and 4 are assigned to the same host system, we combine nodes T1 and T4 in the graph in Figure 3.1, and similarly for nodes T2 and T3. The weights of the edges from this new node to the shared file nodes are computed by summing the individual weights in the original graph model. A possible clustering for shared files F1 to F8 is shown in Figure 3.2. We need, however, to retain the original graph model in order to be able later to calculate remote file access probabilities on a per transaction type basis.

3.3.2. Graph Algorithms and Their Mathematical Interpretations

The clustering problem for the general graph-theoretic model is NP-complete for $n > 2$ [Gar79], where n is the number of clusters (or host systems, in our case). For $n = 2$, we can apply the max-flow min-cut theorem of network flow theory to get the optimal solution [Law76, Sto77]. For $n > 2$, we have to rely on heuristic algorithms for the

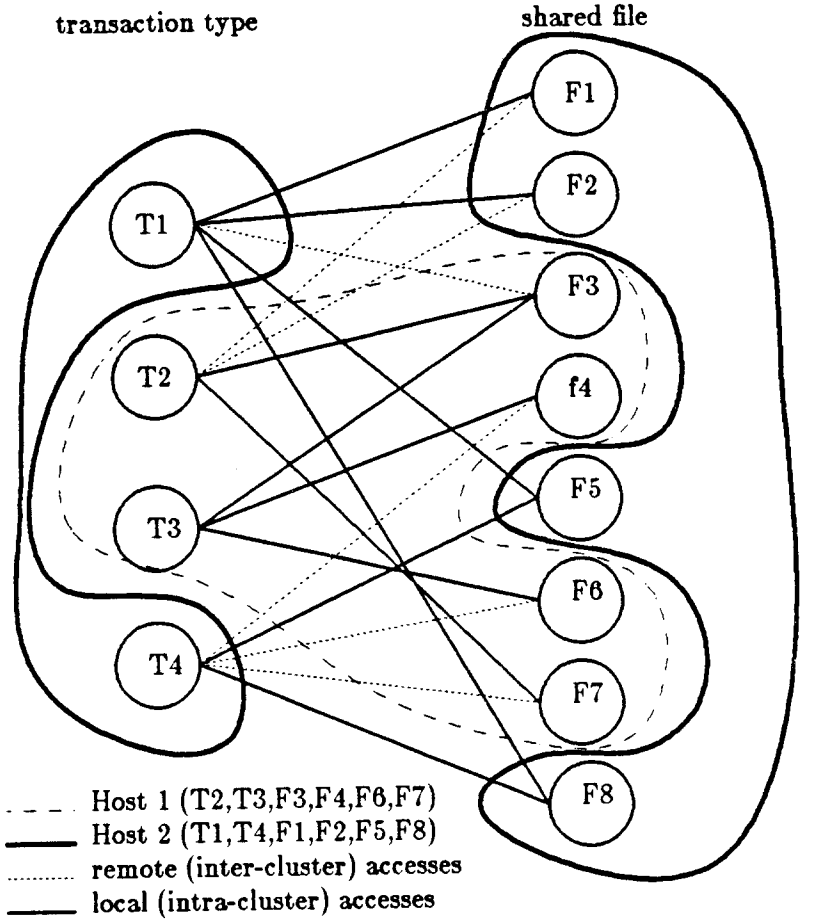

Figure 3.2 File Assignment

solutions. For instance, it has been shown in [Wu80] that a heuristic based on successive applications of the max-flow min-cut theorem yields reasonable clusters.

However, for bipartite graphs, there is a simple polynomial time algorithm to do the clustering. We shall now describe that algorithm.

3.3.2.1. A Polynomial Time Algorithm

Formally, let us assume that the number of host systems is n, the number of shared files is k, and the weight of the edge from node i to node j is c_{ij}. The objective of the assignment of shared files to the n host systems is to cluster the host systems-files graph so that each cluster includes one and only one host system, and the sum of all inter-cluster edge costs, i.e., the total access rate to remote files, is minimum. Mathematically, we can formulate this as a standard integer programming problem [Sal75]. The decision variable x_{ij} is 1 if file j is assigned to host system i, and 0 otherwise. Notice that the sum of both inter and intra-cluster edge costs is a constant for a given graph. It is easier in this case

to formulate the problem in terms of the sum of intra-cluster edge costs. Thus, the objective function to be maximized is

$$\sum_{i=1}^{n}\sum_{j=1}^{k} c_{ij}x_{ij} \text{ , with implicit constraints } \sum_{i=1}^{n} x_{ij}=1 \text{ for all } j, \text{ and } x_{ij}\in\{0,1\}. \quad (3.15)$$

This problem can be solved by the following simple polynomial-time algorithm. The objective function to be maximized can be rewritten as

$$\sum_{j=1}^{k} (\sum_{i=1}^{n} c_{ij}x_{ij}) , \quad (3.16)$$

and we can take advantage of the fact that each term within the parentheses can be independently maximized, since the assignment of one file does not depend on the assignments of the other files. The term in parentheses subject to the constraint $\sum_{i=1}^{n} x_{ij}=1$ is maximized by assigning file j to the host i with the largest weight c_{ij}.

If there are capacity constraints on some host systems, then we can apply a standard (binary) integer programming solution technique [Sal75] to this problem, which is NP-complete. The capacity constraints can be represented as follows :

$$\sum_{j=1}^{k} x_{ij}s_j \leq L_i \text{ for all } i \text{ , } \quad (3.17)$$

where s_j is the size of file j, and L_i is the capacity of host system i.

After clustering the shared files, we can use the original model (see Figure 3.1) to compute the probabilities of local and remote file access (one for each remote host system) for each transaction type. A numerical example is given as part of the second example in Chapter 4.

3.3.2.2. A Simple Extension : Replication of Shared Files

It is often desirable to replicate shared files to allow fast access and to increase availability at the expense of the overhead involved in concurrency and consistency controls [Koh81, Stu80, Par82, Wal83]. As a simple extension to the above algorithm, we can easily allocate shared files with a replication factor of two, i.e., two copies are stored for each shared file in the distributed system. In this case, it seems very attractive to have one copy of the shared file stored in a file server, and the other copy stored on a host system according to the file distribution algorithm we have outlined. We have, on the one hand, maximized the probability of local accesses for the shared file, and, on the other hand, we route the necessary remote accesses to a file server which is less sensitive to the congestion that may exist in various host systems. The "centralization" of the second copies in the file server also eases somewhat the thorny issues of concurrency and consistency controls.

If we are distributing two copies of each file to two host systems, it is not obvious how to represent these two access alternatives in a graph-theoretic model. However, in mathematical terms, we simply modify the constraints in (3.15) to

$$\sum_{i=1}^{n} x_{ij}=2 \quad \text{for all } j, \text{ and } x_{ij}\in\{0,1\}. \tag{3.18}$$

Arguments and algorithms similar to those we have discussed above can then be applied to this replication problem. That is, the first copy is allocated to the host system that accesses it most frequently, i.e., to the host system i with the largest c_{ij}, and the second copy goes to the host system with the second largest c_{ij}.

If capacity constraints exist, then we have again an integer programming problem, and any standard solution technique for such problems can be employed.

CHAPTER 4

Performance Requirement Verification by Queueing Network Models

In the second half of the design methodology, queuing network models are constructed for the distributed system being configured. These models allow us to gain insights into the interactions and the contention for resources among transactions in the proposed distributed system configuration. More specifically, they offer a relatively fast and inexpensive way of verifying whether the given performance requirements can be satisfied by the system being configured. Bottleneck detection and capacity planning can also be done on the basis of the queuing network models to be introduced, as demonstrated in this chapter.

Our models will be product-form queuing networks [Bas75], that can be analyzed quite rapidly and inexpensively by any of the several solution packages now available commercially. The models are constructed hierarchically, by first modeling the various subsystems, and then integrating these submodels into a complete system model. Model parameters are derived from our workload data by introducing appropriate assumptions. We conclude this chapter with a simple example on modeling file system organizations by employing the methodology described here and in the previous chapter, and a more comprehensive one illustrating the design process for the configuration of a local area network-based distributed system.

4.1. Queueing Network Models

We model a local area network-based distributed system in the same way as it is actually constructed, i.e., connecting subsystems together by a local area network. Well known models of a single-machine environment [Boy75, Buz78] are used as models for the subsystems. These submodels are then integrated into a complete distributed system model. This composition method has been "validated" to some extent when a model of the LOCUS distributed system was built [Gol83].

We have already discussed in Chapter 2 the file sharing characteristics of our workload, and of business applications in general. As far as directories for the distributed file system in this study are concerned, we assume that each host has a complete directory of all permanent shared files, and that, because of the stability of these files, directory updates are so infrequent that the network traffic and the overhead caused by them have negligible effects on performance.

In a general distributed and fully or partially replicated directory structure, we could introduce the probability of directory look-up or update, and represent the resource demands of this operation by some CPU and network service times appropriate for the directory structure in question. This would clearly require some measurements or at least estimates of the directory accessing overhead and of the probability of incurring it.

4.1.1. Components of Models

4.1.1.1. Host System

We assume that in our system each CPU is multiprogrammed, and has one or more disk drives and several terminals connected to it. The host system model includes a number of active user terminals which initiate interactions by entering commands or data. Each interaction typically requires some CPU bursts, a few disk accesses, and some display outputs before ending by returning to the user at the terminal. Figure 4.1 illustrates the central-server model [Buz71] of a host system. The CPU is modeled as a PS (processor sharing) service station, which approximates a time-sharing multiprogrammed processor. The disk drive is modeled as a FCFS (first come first served) service station, and both the display output and the user terminals are modeled as IS (infinite servers) service stations.

A typical process will use the CPU, do a disk access or a display output, use the CPU again, then the I/O subsystem again, and so on, until it returns, after its last visit to the CPU, to the user terminal. We generally assume that the number of active terminals is fixed, and that each host can process more than one transaction type. All transactions of a given type have statistically identical behaviors, and constitute a single routing chain in the model. Each terminal, in our model, keeps entering transactions of the same type, but a host can have terminals corresponding to various transaction types.

4.1.1.2. File Server

A file server is just a specialized host system which has no user terminals, hence, no display outputs. In reality, a file server can achieve economies of scale by using high-density disk packs and high-speed disk channels, and distributing the cost of this equipment over all host systems. However, these considerations are beyond the scope of the models described here, which are intended to provide approximate predictions of performance. The length of the CPU service time in the file server is usually much shorter than that of typical CPU burst in a host. It generally depends on the efficiency of

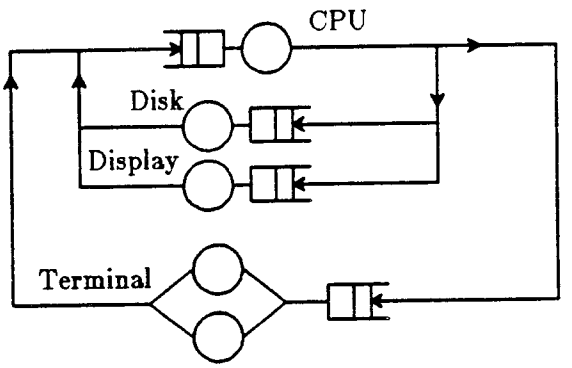

Figure 4.1 A Queueing Model for a Host System

the communication protocols that are executed in the file server in order to satisfy requests from various host systems. The total CPU requirement at the file server is bounded from below by the time it takes the file server to unpack a request datagram [Tan81], to initiate the I/O request, and to pack a return datagram which presumably contains the file sector requested. Figure 4.2 illustrates the model of a file server.

4.1.1.3. Local Area Network

The backbone of the distributed system we are considering is the local area network. It is assumed that the local area network uses a multiaccess protocol [Met76]. This component does not necessarily behave as a FCFS server; however, this approximation will generally be appropriate when the network is not heavily loaded [Alm79, Sho80]. If we make this assumption, the local area network can simply be modeled as a FCFS server with arrivals from all hosts and departures towards them. However, if the network is fairly loaded, say 20 percent utilized, we should use a more accurate model for it, since it is quite likely that we will underestimate the queueing time through the network by ignoring the delay due to collisions.

4.1.2. File Server-based Distributed System Model

Models for the type of distributed systems being studied can be obtained by simply integrating a number of host system models with a local area network model, and a file server model. Slight modifications of the host system model are needed, however. Specifically, in the file server-based model to be studied in this chapter, there is no local disk drive within each host. Nevertheless, it is straightforward to add a disk server in the host system model if there is a scratch disk platter in the proposed host system for storing temporary files and caching permanent files or sectors. All disk drives are in effect grouped together under the control of the file server's CPU. Therefore, a disk I/O operation would go through the network node to the CPU of the file server to access a file server's disk drive, and back through the network node again. Disk I/O's from all hosts are essentially "remote" accesses, and they must compete for resources in the file server among themselves. There is no need to distribute files over various host systems in these

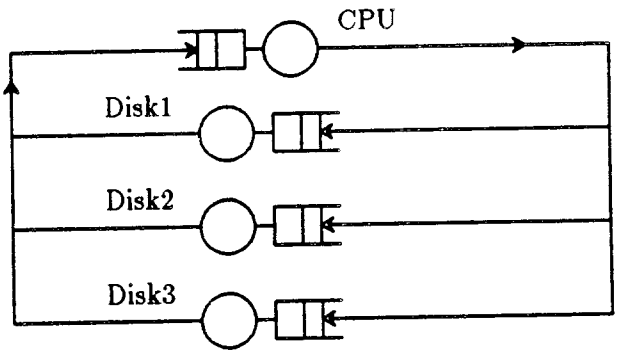

Figure 4.2 A Queueing Model for a File Server

configurations. The file server-based model becomes less complicated since all file accesses are "symmetrical", in the sense that the overhead involved in the file server is independent of where the request originates. A model for this type of distributed system is shown in Figure 4.3. A sample path of a complete user interaction is given by the dotted line in the same figure. Note that jobs/transactions will always return to their "home" host because of their belonging to the routing chain that corresponds to their transaction type, and of the assignment of each transaction type to a single host. Note also that, if a method were used in the first half (see Chapter 3) which assigned transactions of the same type to more than one host, that type would have to be subdivided into subtypes, each consisting of the transactions of that type assigned to a particular host, and a distinct routing chain should be used to model each subtype in the queueing network model in order to ensure the return of each process to its home host after a remote access.

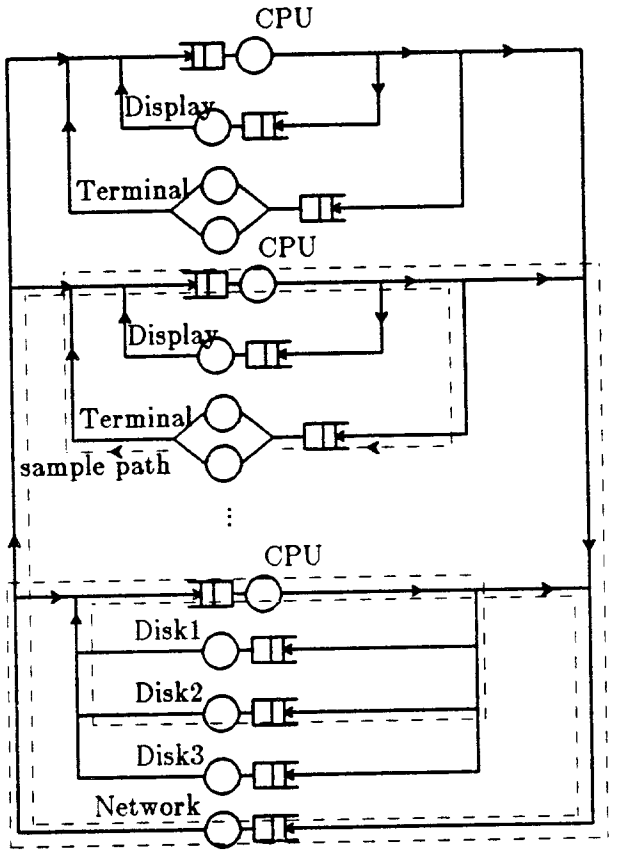

Figure 4.3 A Model of a File Server-based Distributed System

4.1.3. Distributed File System Model

A model for a distributed system in which shared files are stored in various hosts can be similarly obtained by integrating a number of host system models with a local area network model. Remote disk/file accesses in a distributed file system model are routed in a way similar to that used in the file server-based model by viewing a remote host system as a file server for the purpose of a particular disk access. A remote disk access will go through the network node to the remote host's CPU first, then to the remote disk, then to the remote CPU again before coming back to the local CPU through the network node. A remote access must compete with processes running on the remote host for both the remote CPU and the remote disk drive. A model for this type of distributed system is shown in Figure 4.4. A sample path of a complete user interaction is given by the dotted line in the same figure.

4.1.4. Computing Model Parameters

Most of the model parameters can be obtained from the basic analysis of the workload presented in Chapter 2, i.e., in Tables 2.1, 2.2, 2.5, and 2.6. The branching

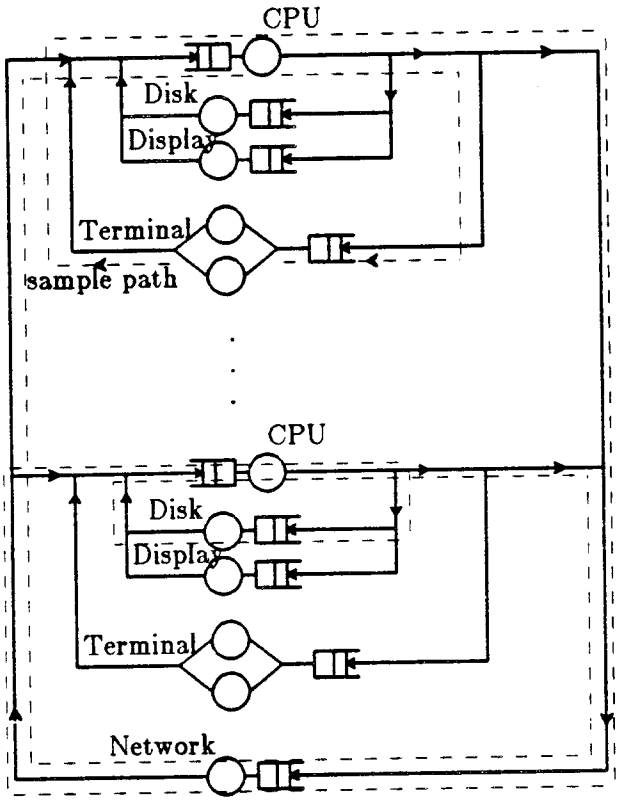

Figure 4.4 A Model of a Distributed File System

probabilities for disk I/O's are invariant with respect to the file system organization. It is necessary, however, to break down these numbers further in an environment with a distributed file system. This breakdown depends entirely on where the shared files are stored as a result of applying the first part of the iterative methodology presented in Chapter 3. The example in Section 4.3 shows how this can be done.

4.2. A Simple Example Modeling File System Organizations

We have already illustrated in the last section how queueing models of the two basic file system organizations, file server-based and distributed file systems, can be constructed, and we now show how they can be used to compare the performances of these organizations.

4.2.1. The Example

In this example, we make the following simplifying assumptions. All hosts have the same number of active terminals, and all terminals are used to input the same transaction type. In the model of a distributed file system, the probability of remote disk access by a transaction/job is the same at all hosts; furthermore, if the disk access is remote, all other hosts are equally likely to be accessed.

All transactions in the system are assumed to be of Type 1 (see Chapter 2). The characterization of that type (T1) is summarized here for convenience as follows :

(1) the mean number of user interactions in a transaction is 47;

(2) the mean total CPU time demand is 16.7 seconds;

(3) the mean number of physical disk accesses per transaction for all shared files is 179;

(4) the mean number of display outputs per transaction is 217.

We assume that each interaction of a transaction uses on the average the same amounts of resources. The CPU bursts and the branching probabilities to disk, terminal, or display server are calculated by assuming geometric distributions of the number of visits to the CPU. Other important parameters are the user think time, the disk service time, the network service time, the display output service time, and the file server's CPU service time. These times are assumed to be 1 second, 32.07 milliseconds, 0.17 milliseconds, 14 milliseconds, and 5 milliseconds, respectively.

Most of the above simplifying assumptions were made to reduce the number of model parameters that could be varied. Three control variables were considered in this experiment, for both the file server-based model and the distributed file system model. The total number of terminals in the proposed distributed system ranged from 2 or 3 to 36, the number of hosts in the distributed system was either two or three, and the probability of remote disk access was either 0.05 or 0.30. Only numbers of terminals that are integral multiples of the number of hosts were actually used in the experiment.

The experiment was carried out by using the RESQ2 queuing network analysis package [Sau82]. We show the mean response time versus the total number of terminals in Figure 4.5, and the total system throughput in transactions/second versus the total number of terminals in Figure 4.6.

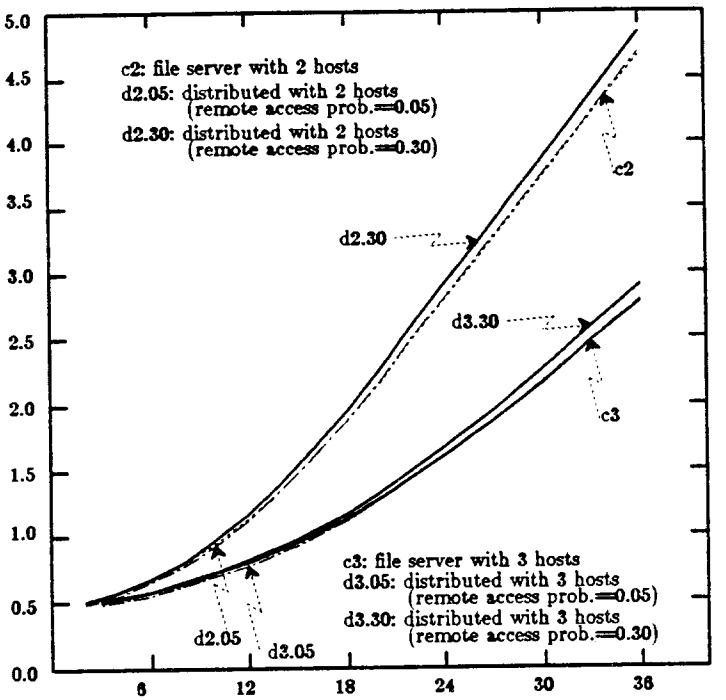

Figure 4.5 Response Time (s) Versus Number of Terminals

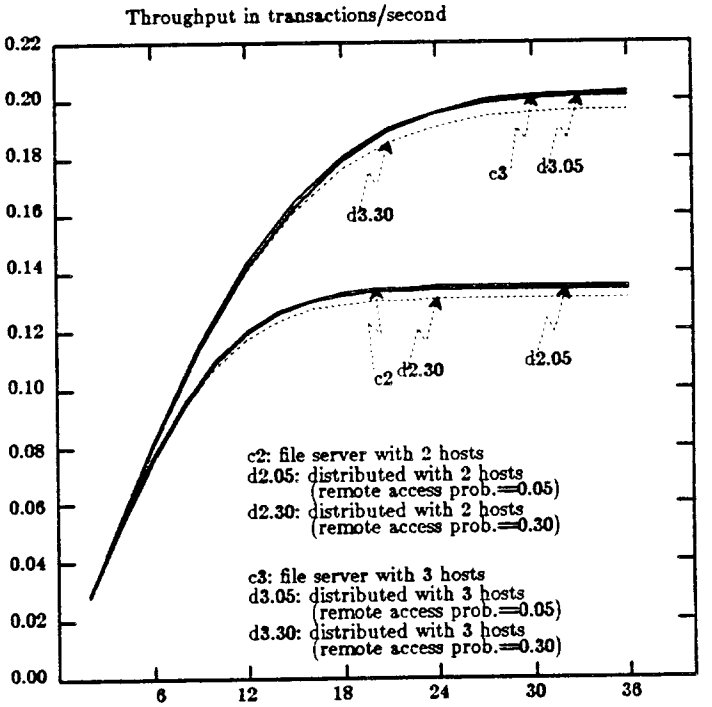

Figure 4.6 Throughput Versus Number of Terminals

4.2.2. Analysis of the Results

On the basis of our approximate models and under all of the assumptions we made, the distributed file system is performance-wise better than the file server-based system when the system is not very congested, i.e., when the number of terminals is low. The converse is true when the system is sufficiently congested. However, the differences between the two organizations in our experiment are not substantial, as shown in Figures 4.5 and 4.6. The cross-over point depends on the probability of remote disk access. The lower this probability, the higher the congestion a distributed file system can tolerate without losing in the comparison with a file server-based one. Both response time and throughput exhibit this phenomenon.

Results of this type are very useful in configuration design as well as in choosing a proper file system organization for a given workload. We can use curves like those presented in Figures 4.5 and 4.6 to select a minimum configuration that meets both the throughput requirement and the response time requirement by choosing a suitable number of hosts. For an existing distributed system, we can also easily check whether adding a few user terminals can increase the throughput enough to meet new demands while still keeping the response time below a reasonable limit.

In the particular system we experimented with, the bottlenecks were the host CPUs. Another experiment was carried out by hypothetically improving CPU speed by twenty percent in a file server-based system. The improvements to be expected of this proposed system are presented in Figures 4.7 and 4.8. It is interesting to observe that a 20% speed-up of host CPUs may allow the system designer to satisfy the new performance requirements simply by upgrading the CPUs without adding new hosts. The eventual decision will really depend on the relative costs of CPU upgrading and of the addition of a new host.

4.3. A More Comprehensive Example and Its Analysis

This section presents an application of both parts of our methodology to a realistic configuration design problem. Models of distributed systems with two host systems are constructed with and without a file server using the first four types of transactions described in Chapter 2, i.e., T1 , T2, T3, and T4. Transaction type T5 is not included in the example since this type of transaction mainly involves output printing and always runs with a much lower priority with respect to the other transaction types in the system. The shared files used by these transactions, to be distributed when appropriate, are the twenty files listed in Table 2.2.

The numbers of terminals corresponding to various transaction types are chosen to closely match the relative throughputs requirement. Configuration design experiments were run with two different file system organizations as well as two CPU speeds. The values of mean CPU bursts shown in Chapter 2 were used for the "regular" type of CPU, while a 40 percent reduction of those values was assumed to reflect proposed technological and architectural enhancements in the CPU's design. This new CPU type is referred to in our experiment as the "enhanced" CPU. We experiment with three terminal population sizes. The number of terminals/transactions for each transaction type is increased approximately linearly to maintain the relative throughputs of the transaction types.

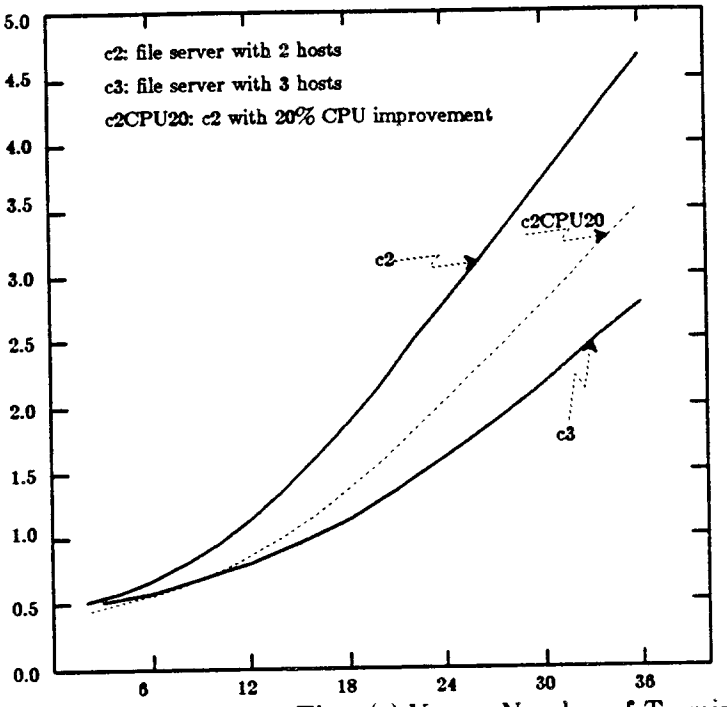

Figure 4.7 Response Time (s) Versus Number of Terminals

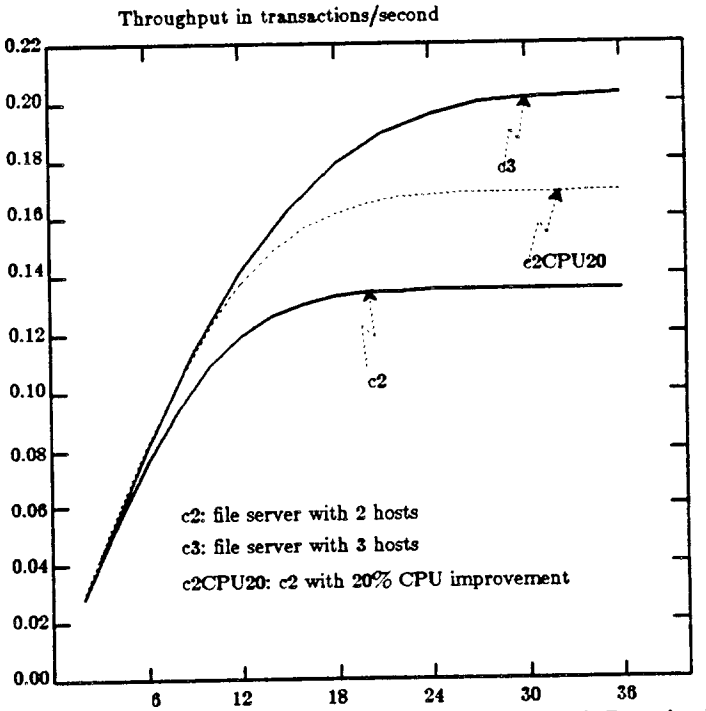

Figure 4.8 Throughput Versus Number of Terminals

4.3.1. Configuring a Hypothetical 2-Host Distributed System

We first derive the relative number of terminals corresponding to each type of transaction needed to match the relative throughputs in Table 4.1. Some relevant data from Chapter 2 are listed again in Tables 4.1, 4.2 and 4.3 for clarity. Notice that the relative throughput requirement among the four transaction types in Table 4.1 is re-normalized since T5 is not included in the configuration design. The initial requirement for the (relative) number of terminals for each transaction type is the ratio of the relative throughput requirement to the transaction throughput rate (per terminal) under the uniprogramming assumption. Without loss of generality, the terminal requirement for T3 is chosen as the base (1.00) for the relative number of terminals required by each transaction type.

In Table 4.2, we compute the relative CPU demands for each transaction type and assign transaction types to host systems by balancing their CPU demands. The CPU demand rate per transaction type is the product of the CPU demand rate per terminal and the relative number of terminals per transaction type. Since terminals of the same transaction type are assigned to the same host system, it is easy to see by using the heuristic algorithm in Section 3.2.2 that transaction types T2 and T4 are in the same host system H1, and transaction types T1 and T3 are in the other host system H2. It is conceivable in practice to assign terminals of the same transaction type to different host systems so that CPU demands can be better balanced; however, this is not done in this example.

Table 4.1 Relative Number of Terminals

Transaction Type	Relative Throughput	Throughput (/minute/terminal)	Relative Number of Terminals
T1	25%	0.822	2.36
T2	25%	0.336	5.75
T3	18.75%	1.450	1.00
T4	31.25%	1.695	1.43

Table 4.2 Relative CPU Demands

Transaction Type	Elapsed Time (s)	CPU Demand Rate (/s/terminal)	Relative Number of Terminals	CPU Demand Rate (/s/type)	Host Assignment
T1	72.905	0.230	2.36	0.542	H2
T2	178.049	0.203	5.75	1.167	H1
T3	41.229	0.690	1.00	0.690	H2
T4	35.393	0.246	1.43	0.350	H1

Table 4.3 Relative Disk Access Demands

Transaction Type	Elapsed Time (s)	Disk Access Demand Rate (/s/terminal)	Relative Number of Terminals	Disk Access Demand Rate (/s/type)
T1	72.905	2.455	2.36	5.793
T2	178.049	4.684	5.75	26.933
T3	41.229	2.425	1.00	2.425
T4	35.393	1.186	1.43	1.695

Table 4.4 Relative File Access Rates

Transaction Type	Shared Files				
	F2	F4	F5	F6	F13
T1	0.191	1.795	0.486	1.164	0.144
T2	2.801	12.200	2.181	6.086	0.000
T3	0.196	1.336	0.395	0.494	0.000
T4	0.088	0.000	0.000	0.296	0.088
H1 (T2+T4)	2.889	12.200	2.181	6.372	0.088
H2 (T1+T3)	0.387	3.131	0.881	1.658	0.144

With this assignment, we next attempt to allocate shared files in order to minimize total remote file accesses. We first compute the relative file access rates in Table 4.3. The disk access demand rate per transaction type is the product of the disk demand rate per terminal and the relative number of terminals per transaction type. As seen in Table 2.2, files F1, F8, F9, F11, F14, and F16 are used very infrequently by transactions of types T1, T2, T3, and T4; hence, these files can be arbitrarily assigned without having significant impact on the file access performance of the distributed system being configured. We can further take advantage of this situation by assigning them to host systems to meet capacity constraints if there are any, or simply to balance the disk storage size at each host system if this is desirable. With our simple algorithm described in Section 3.3.2.1, we quickly identify files F3, F10, and F15 as being made non-trivial use of only by transaction types T1 and T3 (recall that T1 and T3 are assigned to H2); thus, they are assigned to host system H2 without further calculations. Similarly, files F7, F12, F17, F18, F19, and F20 are assigned to host system H1. Table 4.4 shows the breakdown of relative file access rates for the rest of the unassigned files, calculated by using the conditional probabilities in Table 2.2 and the data in Table 4.3. Since transaction types have been already assigned at this point, relative file access rates for each host system can be computed as well.

With Table 4.4 and our simple file assignment algorithm described in Section 3.3.2.1, files F2, F4, F5, and F6 are assigned to host system H1 and file F13 is assigned to host

system H2. In summary, shared files F2, F4, F5, F6, F7, F12, F17, F18, F19, and F20 are assigned to host system H1; shared files F3, F10, F13, and F15 are assigned to host system H2; shared files F1, F8, F9, F11, F14, and F16 can be assigned to either host system. At this point, we can easily compute the conditional probabilities of local file access and remote file access by each transaction type. This is shown in Table 4.5. In order to reflect the existence of those negligible remote file accesses, we prefer not to have zero entries in the table; thus, a small number 0.001 is used instead.

This constitutes the initial configuration for a hypothetical 2-host distributed system with a couple of major design alternatives, i.e., file system organizations and CPU speed improvement. With the results of queueing network models, we can then iterate the steps of the basic methodology to obtain a configuration that will better meet the performance requirements.

4.3.2. Analysis of the Results

Three sets of user population sizes are assumed in this example. They will be referred to as the "minimum", "medium", and "maximum" user populations. Specifically, the minimum user population vector is (2,6,1,1), and corresponds roughly to the relative number of terminals in the last column of Table 4.1; this means that there are 2 terminals running transactions of type T1, 6 of type T2, 1 of type T3, and 1 of type T4. Similarly, the medium and maximum user population vectors are (5,12,2,3) and (7,17,3,4), obtained by multiplying by 2 and by 3, respectively, the relative number of terminals in the last column of Table 4.1. Notice that we round off these relative numbers to the nearest integers. The medium and maximum population sizes are approximately twice and thrice the minimum user population vector, respectively.

The utilizations of the CPUs, the disks, and the network are plotted in Figures 4.9 and 4.10. It is clear that these proposed systems are CPU-bound even for the minimum user population. As we increase the user population, the utilization of the file server only increases slightly due to the severe congestion at the CPUs in the host systems. Also notice that the disk utilizations of the two host systems in the distributed file system case are not well balanced. This is not too much of a surprise since the pattern of file accesses shown in Table 4.4 is very skewed. No matter how we assign the most heavily shared (single-copy) files, significant remote file accesses cannot be eliminated; thus, the usage of the disks in the two host systems is bound to be uneven.

Table 4.5 Probabilities of Local/Remote File Access

Transaction Type	$Prob_{local}$	$Prob_{remote}$
T1	0.369	0.631
T2	0.999	0.001
T3	0.001	0.999
T4	0.948	0.052

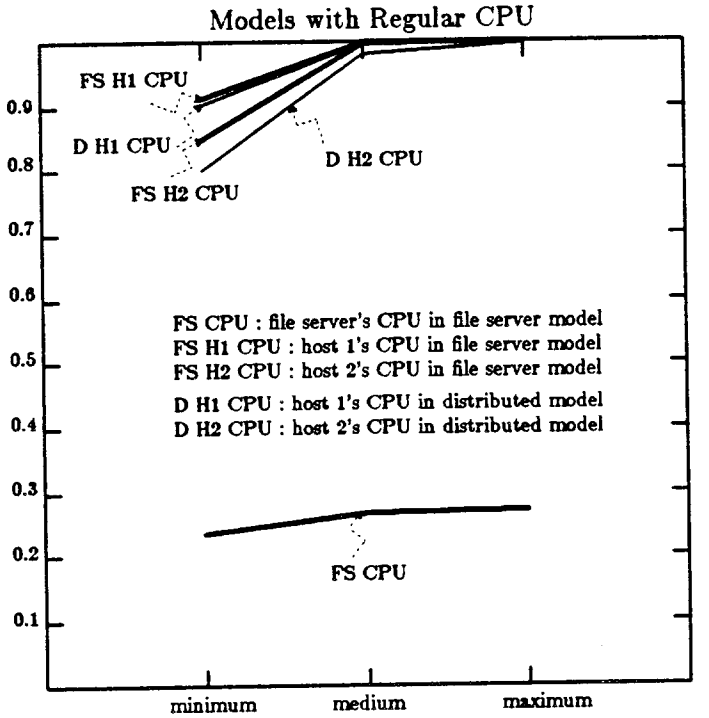

Figure 4.9 Utilizations of CPUs

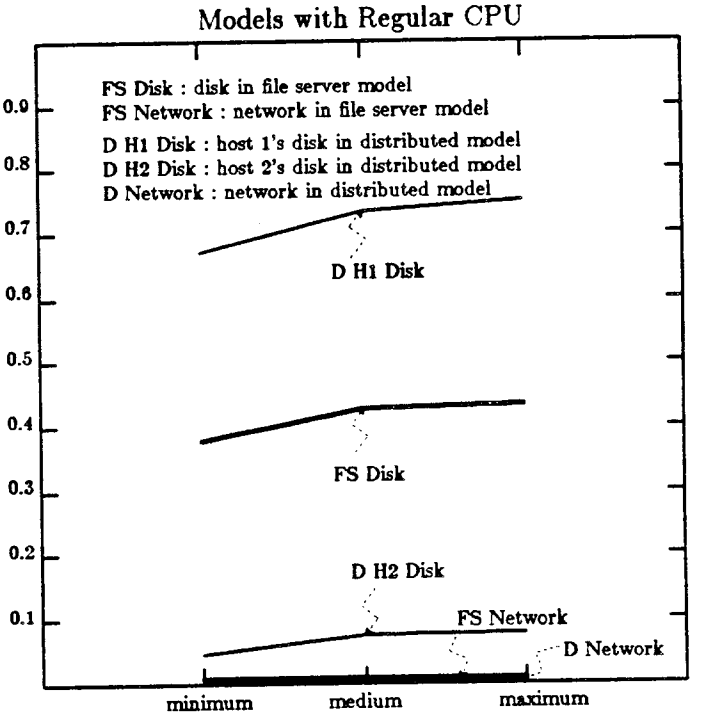

Figure 4.10 Utilizations of Disks and Network

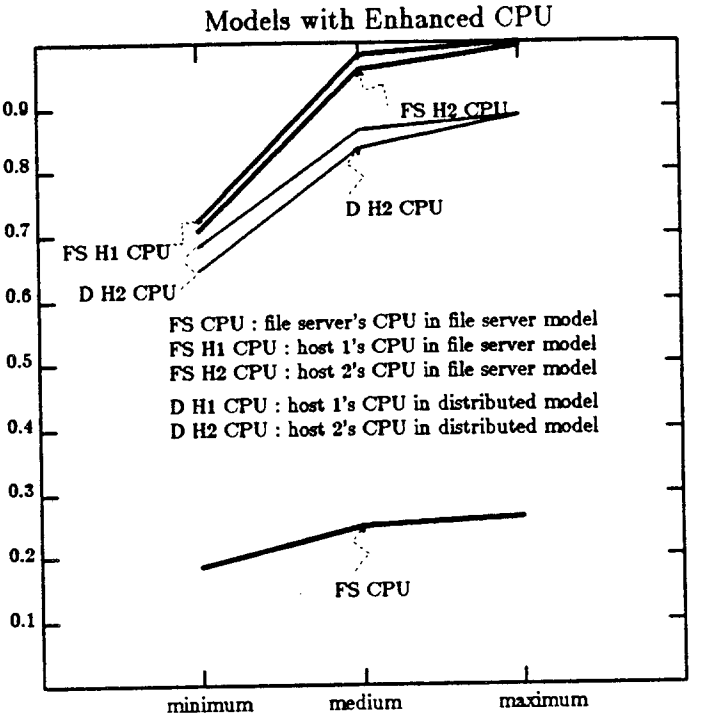

Figure 4.11 Utilizations of CPUs

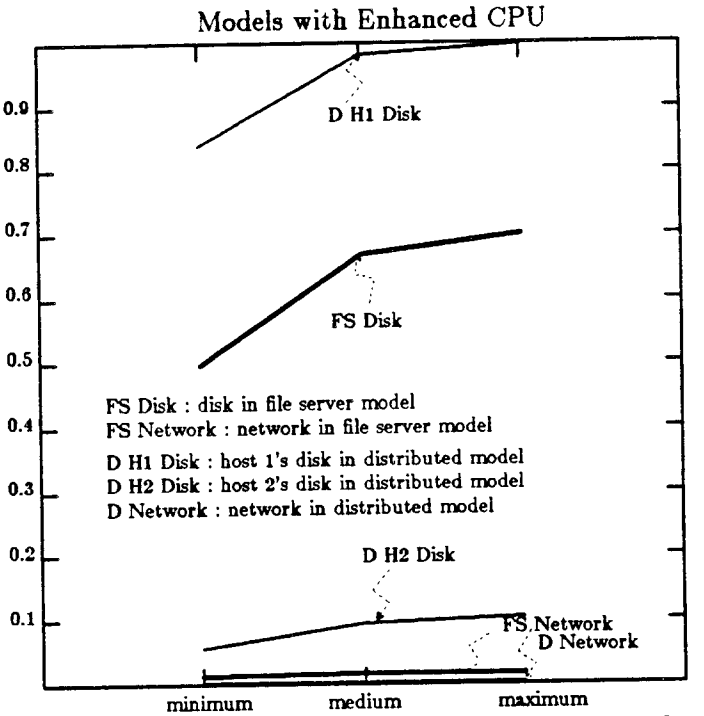

Figure 4.12 Utilizations of Disks and Network

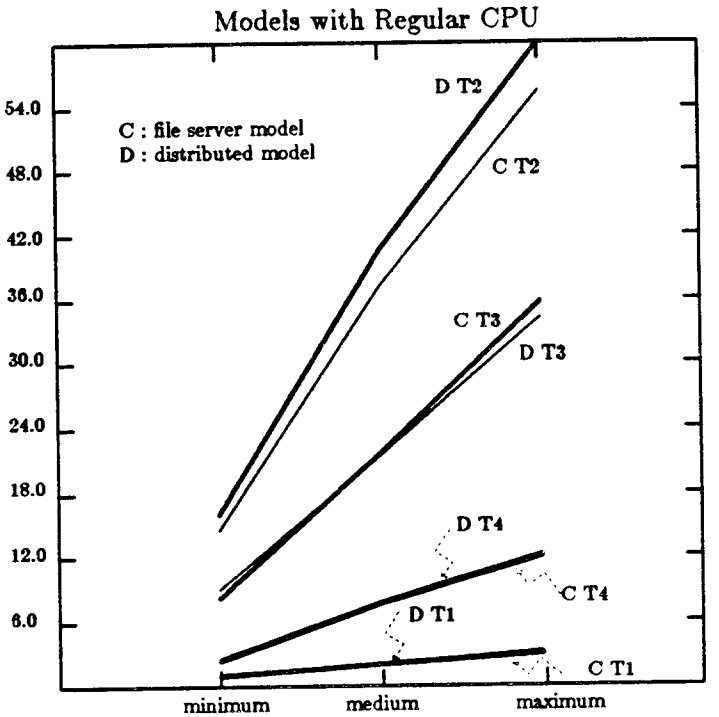

Figure 4.13 Transaction Response Times (s)

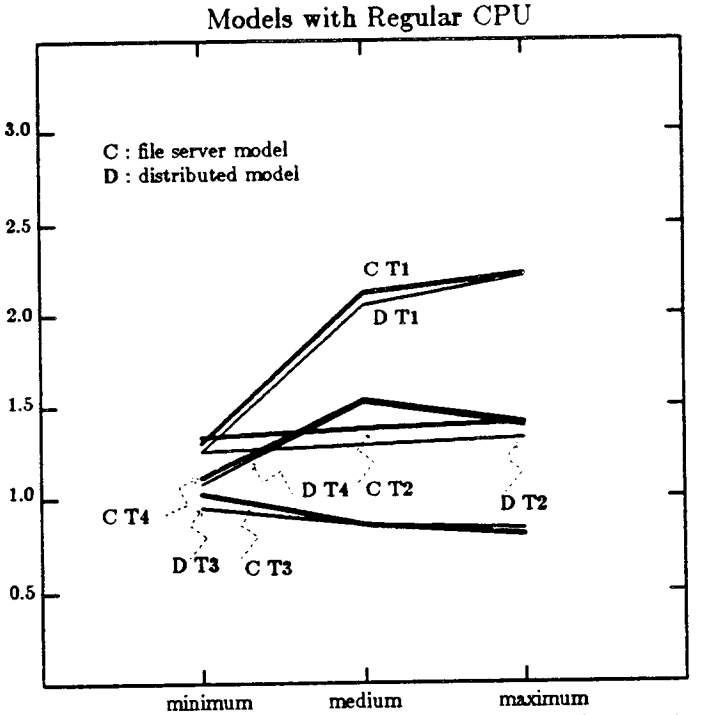

Figure 4.14 Transaction Throughputs (/minute)

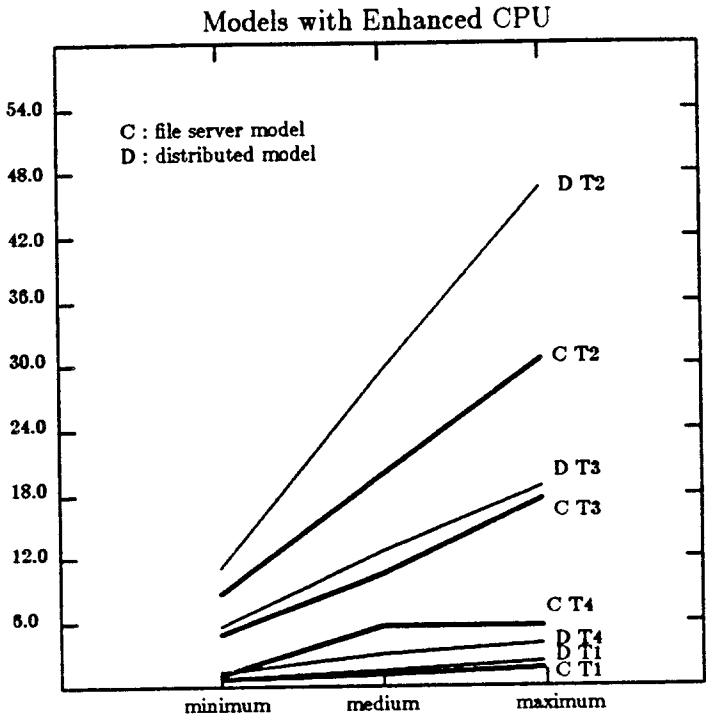

Figure 4.15 Transaction Response Times (s)

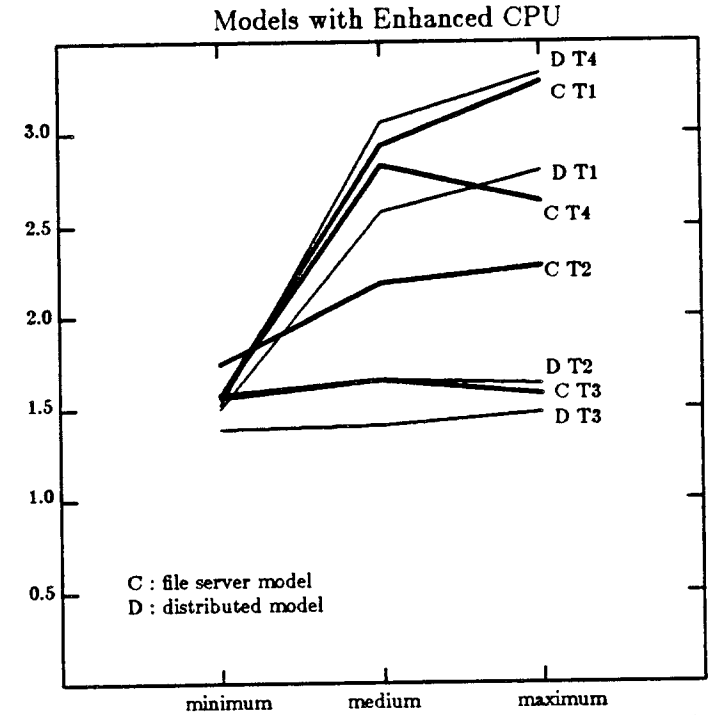

Figure 4.16 Transaction Throughputs (/minute)

Similar results are shown in Figures 4.11 and 4.12 for the same configurations with an "enhanced" CPU. In this case, the file server is better utilized as the user population grows from the minimum to the medium size. Increases in utilization are much better spread out across the various resources. The utilizations of disks in the case of the distributed file system organization are still uneven.

Response times and throughputs for these four transaction types are shown in Figures 4.13 and 4.14. The throughputs of these four transaction types with the "regular" CPU for the minimum user population actually match the given relative throughput requirement quite well. It is not necessary to modify the user population in order to obtain a better match with the requirement. When the user population is maximum, severe contention in the CPUs and disks affects unevenly the transactions of various types; thus, adjusting the user population of each type in the proper direction appears to be necessary. If we are in the design or planning stage, this is probably not the first operation to be done, since the system is completely saturated. In these conditions, the throughputs of *all* transaction types suffer significantly; tuning the system to match relative throughput requirement is indeed not very interesting. Response times as well as throughputs are insensitive to the file system organization in Figures 4.13 and 4.14, since both of these indices are basically limited by the speed of the host CPUs.

In Figures 4.15 and 4.16, we show the response times and throughputs we expect to obtain with the enhanced CPU. The improvements in both the response times and the throughputs are very significant and not surprising. In order to match the relative throughputs in the case of the maximum user population, iterations on the number of terminals for each type appear necessary. The iterations to fine-tune the configuration are part of the methodology, but are not carried out here explicitly. Note that the utilizations of the CPUs in both host systems are very high with the maximum user population, as shown in Figure 4.11. That the throughputs of some transaction types (e.g., T3 and T4 in Figure 4.16) actually decrease at the maximum user population is partially due to the non-linear (although approximately linear) increase in the user population for each transaction type when we go from the medium user population to the maximum user population.

With enhanced CPUs in the system, both the response times and the throughputs of all transaction types exhibit some sensitivity to the file system organization, as shown in Figures 4.15 and 4.16. When the host systems are more congested, the file server-based system exhibits better performance. As far as the remote file access of a transaction is concerned, it is performance-wise preferable to "travel" through the least congested path to get the requested service done. This path includes both the remote CPU and the remote disk. It is not unreasonable to expect that a file server in a heavily loaded distributed system will actually serve as a relief path for remote file access activities. The file server in this case will be a better choice from a performance viewpoint when it is properly designed and configured, like the file server-based 2-host distributed system we have described in this section with an "enhanced" CPU in the case of maximum user population.

CHAPTER 5

Characterizing Multi-phase Workloads in Queueing Networks

The resource demands of the interactions within a transaction are generally not uniform. This is not too surprising, since the earlier phases of a transaction often involve the preparation of data and the retrieval of records, and only later do the computation and record updates actually occur. For example, the last user interaction of a type 1 transaction (see Chapter 2) consumes on the average approximately 25% of the total CPU time demanded by the transaction and accounts for about 50% of the total disk accesses [Whi83]. This observation has led us to the construction of models that reflect this finer-grained characterization of the workload. In a queueing network model, this can be accomplished by allowing transactions to go through class changes which reflect changes in resource demands.

A simple example of a 2-phase workload characterization for transaction type 1 described in Chapter 2 is given in Tables 5.1, 5.2, and 5.3. In this example, we have divided the 47 interactions of transaction type 1 in Table 2.1 into two phases : the first phase includes the first 46 interactions; the other consists of the final interaction. Each phase is characterized by its measured resource demands, as shown in Table 5.1. We do not consider display outputs (see Chapter 2) in this study. In Table 5.2, we have summarized the resource demands per interaction for each of the two phases. The model parameters, CPU bursts and branching probabilities, are presented in Table 5.3.

Interestingly enough, the performance measures of interest, i.e., throughput, mean waiting time, and mean queue length, turn out to have the same values as those produced by models with the single phase assumption on resource demands. It is obvious, however, that a multi-phase workload characterization will provide some detailed performance measures that cannot be captured from the single phase workload characterization. This observation has led us to a formal characterization of *multi-phase* queueing networks. Earlier evidence of an equivalence relationship between these two types of workload characterization can be found in [Fer82]. We shall demonstrate the existence of this relationship in Section 5.1; furthermore, we shall show in Section 5.2 how to obtain detailed performance measures corresponding to a multi-phase workload characterization directly from the results of a queueing network model based on a single phase workload characterization. A simple example in Section 5.3, which generalizes the 2-phase workload characterization given in Tables 5.1, 5.2, and 5.3, concludes this chapter.

5.1. Reducing Multi-phase Queueing Networks

Although some definitions and results introduced in this chapter are applicable to general queueing networks, we shall restrict our discussion to product-form queueing networks [Bas75, Kel79], whose global equilibrium state distribution can be expressed as a product of the equilibrium state distributions of individual service stations. The notation

Table 5.1 Workload Data for Transaction Type 1

Phase	Number of Interactions	Think Time (sec)	CPU Time (sec)	Number of Disk I/O's
I	46	1.0	12.585	89.5
II	1	1.0	4.195	89.5

Table 5.2 Resource Demands per Interaction

Phase	CPU (second)	Disk I/O
I	0.273	1.945
II	4.195	89.5

Table 5.3 CPU Burst and Branching Probabilities

Phase	CPU Burst (msec)	Probability to Disk	Probability to Terminal (class change)	Probability to Terminal (no class change)
I	92.699	0.661	0.332	0.007
II	46.353	0.988	0.012	0.000

and terminology used in this chapter are similar to those used by Baskett et al [Bas75].

It is common practice to represent the routing behaviors and the patterns of resource demands of various customers/jobs by different classes and class transitions [Moo71, Bas75, Fer82]. The networks in which this is done may be called multi-phase queueing networks. Different phases are encoded by different classes, and the phase transitions are governed by the transition probability matrix for classes changes. We define a queueing network to be *phase-free* if there is *at most* one class for each (open and closed) chain in each of the service stations. In essence, the routing behavior of a customer in a phase-free queueing network is probabilistic or "memoryless".

In the rest of this section, we describe a two-step reduction procedure that can be used to obtain, from a general multi-phase queueing network, an equivalent phase-free queueing network. This equivalence is called *state-equivalence* in the sequel. We use the term *aggregate* state description to refer to a state description that is based on the number of customers at each service station of a given chain and does not specifically identify their individual classes. The term *detailed* state description is used to refer to a description which distinguishes customers by their classes. It is true that a multi-phase queueing network can provide detailed (i.e., per class) performance indices; however, we shall show in the Section 5.2 that important detailed performance measures can be readily derived from the corresponding measures at the aggregate state level in the phase-free equivalent network. These results are not only intuitively appealing, but also useful in

some computational aspects of the convolution algorithm that may be used to solve product-form queueing networks [Lam83].

When obtaining the phase-free network equivalent to a given multi-phase network, the general topology, the queueing disciplines of the service stations, and the routing structure at the level of the service stations do not have to be changed. As shown below, only the two following sets of network parameters are to be derived for the equivalent phase-free counterpart : the mean aggregate (composite) service time at each service station for each chain, and the branching probabilities (the transition matrix) at each service station for each chain.

Let the pair (i,r) represent the state of a customer of class r at service station i. The transition from state (i,r) to state (j,s) can be characterized by the probability $P_{i,r;j,s}$. In essence, a customer of class r at end of service at station i goes to station j and changes to class s with probability $P_{i,r;j,s}$. An open or closed chain C can be defined as a collection of (i,r) pairs which a customer of this chain may go through, i.e., $C = \{(i,r)\}$. Particularly, if $(i,r) \in C$ and $(i,s) \in C$ implies $r = s$ for each chain in a queueing network, the queueing network is phase-free.

5.1.1. Step I : Mean Aggregate Service Time

Step I : For each (open or closed) chain C, the mean aggregate service time $1/\mu_i$ at service station i is set equal to

$$\sum_{r,(i,r)\in C} \frac{e_{ir}}{\sum_{r,(i,r)\in C} e_{ir}} \frac{1}{\mu_{ir}} \tag{5.1}$$

where e_{ir} is the (relative) throughput for class r customers at service station i, and $1/\mu_{ir}$ is the corresponding mean service time.

Essentially, the aggregate service time of a single phase, i.e., phase-free, queueing network at each service station is computed as the weighted sum of individual service times of each class, where the weight is the relative throughput of each class. We sometimes refer to $e_i = \sum_{r,(i,r)\in C} e_{ir}$ as the total (relative) throughput at service station i to be attributed to chain C. We do not mention symbol C in e_i explicitly for brevity.

The equivalence relationship between a single phase queueing network and a multi-phase queueing network can be demonstrated rather easily from the work by Kelly [Kel79][1]. Kelly introduced the notions of stochastic reversibility, network of symmetric queues, and network of quasi-reversible queues. The quasi-reversible queues described by Kelly include all four types of queues that can be found in a BCMP network [Bas75]. His work encompasses a class of the product-form queueing networks slightly more general than that of product-form networks described in [Bas75]; nevertheless, his treatment of the product-form results is much more coherent and concise than that in [Bas75], because of his use of the powerful paradigm of reversibility argument. The proof of state-equivalence stems from Theorems 3.8 and 3.12 in [Kel79]. The relevant parts of these

[1]For some obscure reason, Kelly's work is not well known in the computer science community.

theorems are combined and re-stated below as Theorem 5.1, with the notation and terminology used in [Bas75]. The proofs of these theorems can be found in [Kel79].

Theorem 5.1 [Kel79]

A network of quasi-reversible queues has the following properties :

(i). The probability that a queue in the network contains n customers is of the form

$$\frac{{a_i}^n}{\prod_{l=1}^{n} \phi_i(l)} \tag{5.2}$$

subject to normalization in the probability space, where $\phi_i(l)$ is the service rate of service station i when there are l customers in the queue, and $a_i = \sum_{r,(i,r)\in C} \frac{e_{ir}}{\mu_{ir}}$.

(ii). Given that there are n customers in the queue, the classes of customers are independent, and the probability the customer in a given position in the queue is of class r is $\frac{e_{ir}/\mu_{ir}}{a_i}$.

For example, for an infinite server station, the service rate $\phi(l)$ is defined by $\phi(l)=l$. Part (ii) of Theorem 5.1 is particularly significant, as it states that, given the aggregate state distribution, the conditional detailed state distribution is *multinomial*.

Given a multi-phase queueing network, applying step I of the reduction procedure we have

$$1/\mu_i = \sum_{r,(i,r)\in C} (e_{ir}/e_i)\, 1/\mu_{ir} \ , \ \text{hence} \ e_i/\mu_i = \sum_{r,(i,r)\in C} e_{ir}/\mu_{ir} \ ,$$

and, from part (i) of Theorem 5.1, a state at the aggregate level of the multi-phase queueing network must have the same probability as that of the phase-free queueing network.

That this is the case can also be demonstrated algebraically from the results in [Bas75] by summing up all the relevant probabilities and using the multinomial expansion identity

$$(\sum_{r=1}^{R} a_{ir})^n = n!\,(\sum_{\sum_{i=1}^{R} n_i = n} \prod_{i=1}^{R} \frac{{a_{ir}}^{n_i}}{n_i!}) \ . \tag{5.3}$$

This procedure was in fact applied to prove a less general result in Ferrari's work [Fer82].

5.1.2. Step II : Branching Probabilities

Step II : For each (open or closed) chain C and each service station i, we define the aggregate state $(i) = \{(i,r) \mid r,(i,r)\in C\}$. The new entries of the transition matrix $\left[P_{ij}\right]$ for chain C are of the form

$$P_{ij} = \frac{\sum_{(i,r)\in(i),(j,s)\in(j)} e_{ir}\, P_{i,r;j,s}}{\sum_{(i,r)\in(i)} e_{ir}} \tag{5.4}$$

The denominator in (5.4) should be interpreted as the normalization factor for each row of the new transition probability matrix. It is fairly straightforward to show that the relative throughputs e_i (for all i) derived from this new matrix are exactly $\sum_{r,(i,r)\in C} e_{ir}$ for an open chain, and differ from $\sum_{r,(i,r)\in C} e_{ir}$ only by a multiplicative constant for a closed chain. This property is referred to as ***throughput-equivalence*** for the two queueing networks involved in this reduction procedure of the transition matrix. In general, state-equivalence does not imply throughput-equivalence, as shown by the simple example in Figure 5.1, where all service stations are operating with a FCFS queueing discipline. The corresponding throughputs of these two similar networks at all service stations cannot be the same simply because the branching probabilities are not the same. However, the relative loadings at all corresponding service stations, i.e., the a_i's in Theorem 5.1, are the same. Thus, the equilibrium state distributions are the same for both networks.

It will become much clearer later that the reduction in step II is not essential for the derivation of the equilibrium state distribution, or for that of the performance measures

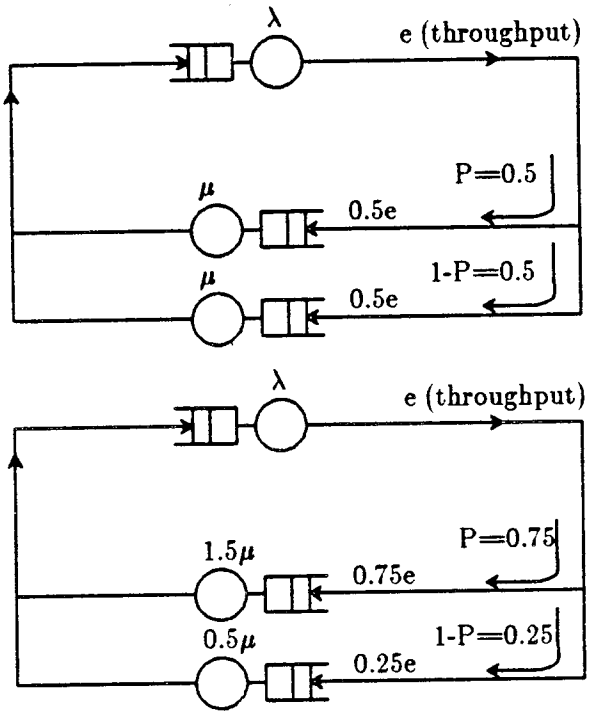

Figure 5.1 State-equivalent, not Throughput-equivalent Networks

associated with the detailed state space. It is needed only when the transition probability matrix of the phase-free queueing network in question is explicitly called for.

5.2. Obtaining Detailed Performance Measures

In this section, we shall show that, given a multi-phase workload characterization, detailed performance measures of multi-phase queueing network models can be easily derived from equivalent phase-free network models without solving directly the more complicated multi-phase queueing network models. As mentioned in the previous section, the detailed state distribution for the multi-phase queueing network can be constructed from the phase-free result by using the property of conditional multinomial distribution in part (ii) of Theorem 5.1. In general, we are interested only in a few performance measures such as throughput, mean waiting time, and mean queue length. These performance measures can be readily derived on a per class basis from the performance measures at the aggregate level. Specifically, we are going to show how to calculate throughput λ_{ir}, mean queue length L_{ir}, and mean waiting time W_{ir} for each class r from the corresponding aggregate measures λ_i, L_i, and W_i.

5.2.1. Throughput

Using the reduction procedure described in Section 5.1.2, we preserve the throughput-equivalence relationship. The throughput λ_{ir} for class r at service station i in a multi-phase queueing network is expressed as

$$\lambda_{ir} = \frac{e_{ir}}{\sum\limits_{r,(i,r)\in C} e_{ir}} \lambda_i \quad \text{or} \quad \lambda_{ir} = \frac{e_{ir}}{e_i} \lambda_i \ , \tag{5.5}$$

where λ_i is the aggregate throughput at service station i for chain C. This result is trivial for open chains since the e_{ir}'s are absolute measures of throughputs. The result in (5.5) is straightforward also for closed chains, since the e_{ir}'s are relative throughputs, and their ratios equal the ratios of the absolute throughputs.

5.2.2. Mean Queue Length

Given the mean (aggregate) queue length L_i for chain C at service station i, the mean queue length L_{ir} for class r for the general multi-phase queueing network is

$$L_{ir} = \frac{e_{ir}/\mu_{ir}}{\sum\limits_{r,(i,r)\in C} e_{ir}/\mu_{ir}} L_i \quad or \quad L_{ir} = \frac{e_{ir}/\mu_{ir}}{e_i/\mu_i} L_i \ . \tag{5.6}$$

The validity of this result comes directly from part (ii) of Theorem 5.1, which states that the expected number of class r customers in station i is $n\ (e_{ir}/\mu_{ir})/(e_i/\mu_i)$. Applying the general definition of mean queue length, i.e., $L = \sum nP(n)$, it is easy to show that relation (5.6) holds.

5.2.3. Mean Waiting Time

The mean waiting time (including service time) W_{ir} for a customer of class r at service station i is obtained by using Little's formula, $L = \lambda W$. Thus,

$$W_{ir} = \frac{L_{ir}}{\lambda_{ir}} = \frac{1/\mu_{ir}}{1/\mu_i}\frac{L_i}{\lambda_i} = \frac{1/\mu_{ir}}{1/\mu_i} W_i \,, \tag{5.7}$$

where W_i is the mean aggregate waiting time in the simplified phase-free queueing network.

It is interesting to note that the mean waiting time ratio $\frac{W_{ir}}{W_i}$ equals the mean service time ratio $\frac{1/\mu_{ir}}{1/\mu_i}$ for class r customers. Also, the mean waiting time W_{ir} has the form $K_i \frac{1}{\mu_{ir}}$, where $K_i = \frac{1/\mu_i}{W_i}$ is invariant with respect to classes of the same chain. Note that we use Little's formula to obtain the results for the mean waiting time in this section; these results will hold if and only if we have both throughput equivalence and state equivalence between two networks.

5.3. An Example of Equivalence Between Two Models

For the queueing network model of a host system in Figure 5.2 (see also Figure 4.1), we shall derive parameters from the workload data. We shall do this both for a two-phase version and for a phase-free version of the model, and demonstrate the equivalence between these two models. Note that we only show throughputs and mean service rates for the phase-free version in Figure 5.2.

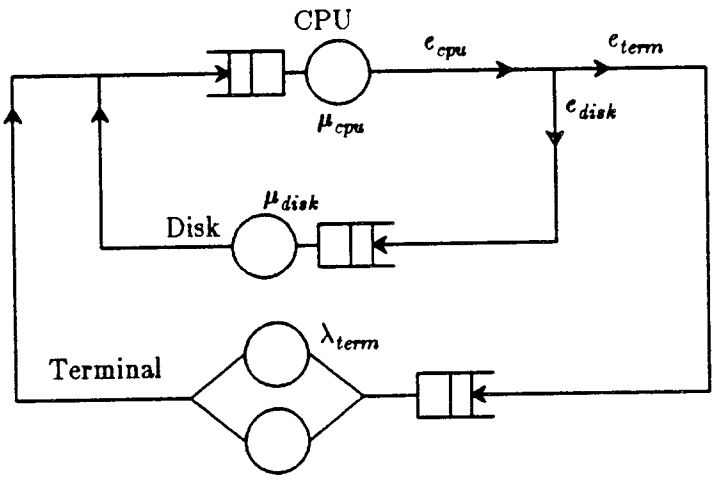

Figure 5.2 A Model for a Host System

Let N, Q, and D be the average number of interactions, the mean CPU time used, and the mean number of disk accesses for a transaction, respectively. Let us also assume that a transaction can be meaningfully broken down into two phases, and let the detailed breakdowns of these quantities for these two phases be denoted as follows :

$$N = N_A+N_B\ ,\quad Q = Q_A+Q_B\ ,\quad \text{and}\quad D = D_A+D_B\ , \tag{5.8}$$

where subscripts A and B denote the portions attributed to phase A and phase B, respectively.

We assume that the mean think time $1/\lambda_{term}$ and the mean disk service time $1/\mu_{disk}$ are the same for both phases. Other parameters are derived according to the assumption of uniform resource demands within each phase, and the geometrical distribution assumption for the number of visits to the various service stations.

For the one-phase or phase-free model, we have the following model parameters :

$$\frac{1}{\mu_{cpu}} = \frac{\frac{Q}{N}}{1+\frac{D}{N}}\ ,\ P_{disk} = \frac{\frac{D}{N}}{1+\frac{D}{N}}\ ,\ \text{and}\ P_{term} = 1-P_{disk}\ . \tag{5.9}$$

For the two-phase model, the first phase parameters are expressed as follows :

$$\frac{1}{\mu_{cpuA}} = \frac{\frac{Q_A}{N_A}}{1+\frac{D_A}{N_A}}\ ,\ P_{diskA} = \frac{\frac{D_A}{N_A}}{1+\frac{D_A}{N_A}}\ ,$$

$$P_{termAB} = P_{AB}\,(1-P_{diskA})\ ,\ \text{and}\ P_{termAA} = (1-P_{AB})\,(1-P_{diskA})\ , \tag{5.10}$$

where $P_{AB} = 1/N_A$ is the probability of a class (phase) change from A to B.

Similarly, the parameters for the second phase are expressed as follows :

$$\frac{1}{\mu_{cpuB}} = \frac{\frac{Q_B}{N_B}}{1+\frac{D_B}{N_B}}\ ,\ P_{diskB} = \frac{\frac{D_B}{N_B}}{1+\frac{D_B}{N_B}}\ ,$$

$$P_{termBA} = P_{BA}\,(1-P_{diskB})\ ,\ \text{and}\ P_{termBB} = (1-P_{BA})\,(1-P_{diskB})\ , \tag{5.11}$$

where $P_{BA} = 1/N_B$ is the probability of a class (phase) change from B to A.

In order to show that the performance measures of interest to us are identical in the two models, it is sufficient to show that we have both throughput equivalence and state equivalence at the aggregate level. This is done in the next two subsections.

5.3.1. Throughput Equivalence

We need to write for both models a set of (relative) throughput equations, and solve this set of equations. The set of throughput equations for the one-phase model is as follows :

$$\begin{cases} e_{cpu} = e_{disk} + e_{term} \\ e_{disk} = P_{disk}\, e_{cpu} \\ e_{term} = (1-P_{disk})\, e_{cpu} \end{cases}$$

One simple solution for the e's is $e_{cpu}=1$, $e_{disk}=P_{disk}$, and $e_{term}=1-P_{disk}$. Similarly, the set of throughput equations for the two-phase model is as follows.

$$\begin{cases} e_{cpuA} = e_{diskA} + e_{termA} \\ e_{diskA} = P_{diskA}\, e_{cpuA} \\ e_{termA} = (1-P_{diskA})(1-P_{AB})\, e_{cpuA} + (1-P_{diskB})\, P_{BA}\, e_{cpuB} \end{cases}$$

$$\begin{cases} e_{cpuB} = e_{diskB} + e_{termB} \\ e_{diskB} = P_{diskB}\, e_{cpuB} \\ e_{termB} = (1-P_{diskB})(1-P_{BA})\, e_{cpuB} + (1-P_{diskA})\, P_{AB}\, e_{cpuA} \end{cases}$$

One solution for the e's is

$$e_{cpuA}=1 \ , \ e_{diskA}=P_{diskA} \ , \ e_{termA}=1-P_{diskA} \ ,$$

$$e_{cpuB}=\frac{1-P_{diskA}}{1-P_{diskB}}\,\frac{P_{AB}}{P_{BA}} \ , \ e_{diskB}=P_{diskB}\,\frac{1-P_{diskA}}{1-P_{diskB}}\,\frac{P_{AB}}{P_{BA}} \ , \text{ and } \ e_{termB}=(1-P_{diskA})\,\frac{P_{AB}}{P_{BA}} \ .$$

It is straightforward algebra to show that the ratio between e_{cpu} and $e_{cpuA}+e_{cpuB}$ is $(N+D)/(N_A+D_A)$; this can be done by plugging the definitions of the model parameters derived from the workload data, i.e., relations (5-8) to (5-11), into the solutions for e_{cpu}, e_{cpuA}, and e_{cpuB}. Similarly, the ratio between e_{disk} and $e_{diskA}+e_{diskB}$ and the ratio between e_{term} and $e_{termA}+e_{termB}$ are both equal to $(N+D)/(N_A+D_A)$. Since we have the same ratios for all three throughput measures, the two networks are throughput equivalent.

5.3.2. State Equivalence

The mean aggregate service rate for the two-phase model at the terminal service station is λ_{term} , since the mean service rates of the terminal stations are the same in both phases. A similar argument applies to the disk service station. The only mean aggregate service rate for the two-phase model that needs to be checked against the phase-free network is that of the CPU service station. Let us denote by $1/\mu_{aggr}$ the weighted sum of $1/\mu_{cpuA}$ and $1/\mu_{cpuB}$ according to (5.1) :

$$1/\mu_{aggr} = \frac{1\ \dfrac{1}{\mu_{cpuA}} + \dfrac{1-P_{diskA}}{1-P_{diskB}}\ \dfrac{P_{AB}}{P_{BA}}\ \dfrac{1}{\mu_{cpuB}}}{1 + \dfrac{1-P_{diskA}}{1-P_{diskB}}\ \dfrac{P_{AB}}{P_{BA}}} .$$

After plugging in the definitions of the model parameters, i.e., relations (5.8) to (5.11), we obtain

$$1/\mu_{aggr} = \frac{Q}{N+D} = \frac{Q/N}{1+D/N} = 1/\mu_{cpu} .$$

Thus, we have shown that the mean aggregate service time at every service station is the same for both the two-phase model and the one-phase (i.e., phase-free) model. This concludes the proof that the two networks we are considering are state equivalent.

Thus, since they are also throughput-equivalent, all performance measures of interest, i.e., throughput, mean queue length, and mean waiting time, are identical.

CHAPTER 6

Configuring Workstation-based Distributed Systems

The rapid advances in microelectronics technology and the consequent reduction of computer systems' costs have led to the development of a special type of distributed system, i.e., the workstation-based distributed systems. A workstation is a single user desktop computer system typically with a bit-mapped display, a powerful CPU, and sufficient memory to execute most user jobs [Tha82, Bec82]. The workstations usually have limited non-volatile storage capacity for the user's permanent data files. These workstations thus rely on remote file servers for most of their needs in file storage and retrieval. Often, a file server is a specialized workstation equipped with high density disks as well as proper software for serving its client workstations. Workstations are usually connected together by a high speed local area network [Cla78] which facilitates communications among users as well as allowing access to file servers for file storage and data retrieval through some suitable mechanism such as a network operating system. File systems of this type are sometimes called "remote" file systems as opposed to distributed file systems. Notice that the amount of storage capacity at each workstation is likely to increase in the future; hence, investigations of file/sector/record caching from both the functional and performance viewpoints provide interesting areas of research with useful future applications.

In our study, a multiaccess protocol at the data link layer like that used in the Ethernet is assumed [Eth80]. Packets at this level containing file blocks or disk sectors are transferred back and forth between workstation and file server. High level file transfer protocol can be built on top of this data link protocol.

Within the framework we have developed so far, such workstation-based distributed systems can be easily characterized. These distributed systems correspond to the file server-based distributed systems shown in Figure 4.4, with one user per host system; however, it is often the case that in these distributed systems there are many such single-user "host systems", or workstations. Large numbers of host systems imply large numbers of closed chains in the corresponding models; these multi-chain models present computational difficulties in the solution techniques for the queueing network models. We shall discuss this issue in the next section. The rest of the chapter discusses the notion of balanced file server design and a simple workload characterization for various types of workstation jobs; based on this characterization of the workload, we propose a system-wide load measure for the prediction of performance indices of various workload types and for planning the capacity of a workstation-based distributed system.

6.1. A Solution Approach for Multi-chain Queueing Networks

The convolution algorithm for product-form queueing networks was first described by Buzen [Buz73] for single-chain networks, and was extended by Reiser and Kobayashi

[Rei75] to multi-chain networks. When the number of network states is large, the normalization constant of the convolution algorithm may be too large or too small, thereby causing a floating-point overflow or underflow. The computational complexity, in time and space, of the convolution algorithm makes it practically intractable for a large number of chains; more specifically, both complexities are proportional to $\prod_{k=1}^{K}(N_k+1)$, where N_k is the number of users in chain k, and K is the number of chains. The mean value analysis algorithm of Reiser and Lavenberg [Rei80] bypasses the evaluation of the normalization constant, and computes the mean queue lengths and chain throughputs directly. It avoids the problem of floating-point overflows, but floating-point underflows may still occur [Rei81]. Nevertheless, time and space requirements of the mean value analysis algorithm still grow exponentially with K. Although the tree convolution algorithm recently proposed by Lam and Lien [Lam83] reduces the computational cost, in time and space, it only applies to phase-free queueing networks, and works best only when the queueing chains are sparse and local. This algorithm certainly allows us to model comparatively more chains in the queueing networks and get solutions for such networks.

Approximation methods have been used to solve multi-chain queueing networks. A set of non-linear equations for throughputs were derived and solved in [Gol83]. Mean queue lengths can be expressed in terms of these throughputs; mean response times are then derived from Little's formula. Iteration methods are also used to approximate the solution of multi-chain queueing networks [Rei79, Rei80]. It is often the case that chain populations have to be large in order for the approximation methods to work well. In terms of our modeling efforts, we have one user, the minimum possible, per chain, and this is not too compatible with the use of these approximation methods.

We propose to turn to the heart of the problem, i.e., the number of closed chains, by examining the workload characteristics for this particular type of models. It is expected that the number of closed chains can be significantly reduced by clustering the workload [Fer78] properly. In the workstation-based distributed system models, the workload for a particular user (or a transaction type) can be characterized by the tuple (think time, CPU burst, probability of file server access). The probability of going to the user terminal after a CPU burst is the complement of the probability of going to the file server; thus, it is not part of the tuple. Although this tuple is very simple indeed, we shall show in the sequel another characterization for each workload type, which is a simple function of this tuple and of the service demands at each station. We choose not to impose the semantic interpretation of "transaction" on the sequence of user interactions at the workstations for these systems, since such interpretation does not affect the solutions of the models, and the relevant measures at the so-called transaction level can be easily derived from those at the interaction level.

After the workload is clustered, users are classified according to their workload type; therefore, the number of chains in the model is reduced. The number of users in a chain then corresponds to the number of users that get clustered into the same workload type. We need not have separate CPU service stations and terminal/think service stations for the same type of users. Instead, a single infinite-server station can be used for the CPUs and similarly for the terminals of all users of the same type. This reduces the number of

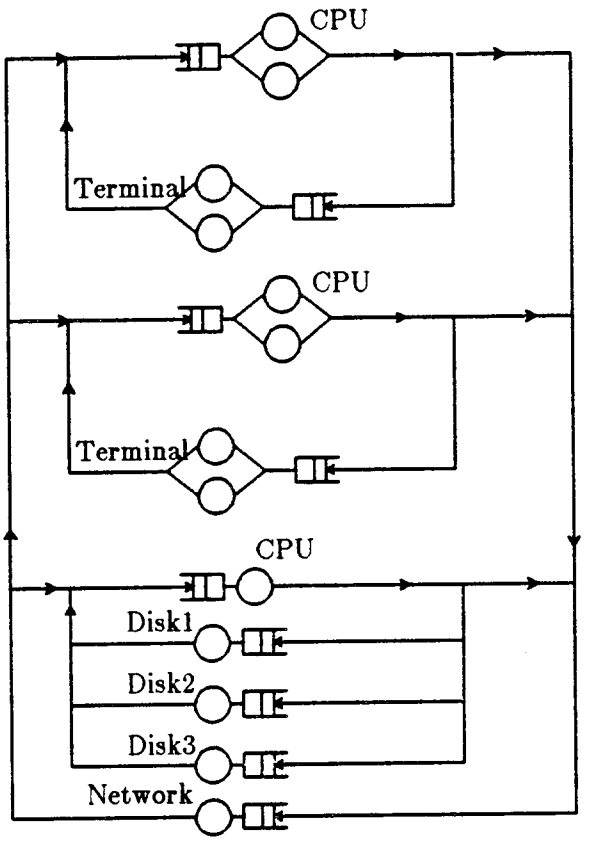

Figure 6.1 Two-chain Workstation-based Distributed System Model

explicit service stations as well as the number of chains. For convenience and clarity in later discussions, we transform the model in Figure 4.4 into the workstation-based model in Figure 6.1, in which two routing chains are represented. As in the previous models, a remote disk access goes through the network first, the file server's CPU, one of the disks, back to the file server's CPU, then back to the CPU of the workstation through the network again. Notice that we still assume the same (exponential) service time distribution for the FCFS stations, i.e., the local area network and file server's disks, for all workload types. Thus, it is possible, though too compact conceptually and less clear in the presentation, to coalesce users of all workload types into single infinite-server service stations corresponding to terminals and CPUs, respectively, while using different classes to reflect different service demands and to distinguish among workload types in the local area network and file server, so that they can be routed properly in the queueing network.

6.2. Workload Data and Model Parameters

The workload data used in the queueing models are drawn from Chapter 2. Since the workload has supposedly been clustered, an I/O-bound transaction type (A) and a CPU-bound transaction type (B) are selected to reflect simple clustering of workload

Table 6.1 Workload Types Used

Workload Type	Transaction Type in Chapter 2	Think Time (s)	CPU Burst (ms)	Probability of File Server Access
A	T5	0.5	33	0.99
B	T4	2.0	161	0.77
C	T1	1.0	74	0.79

characteristics. In some experiments, a third intermediate type (C) is used. Except for some numerical round-offs, their characteristics are taken directly from Chapter 2 and summarized in Table 6.1. We assume the same 32 millisecond disk service time and a 5 millisecond mean file server CPU time. The user terminal in a workstation is connected directly to the CPU so that there is no need to model the display output server. The network is assumed to have 10 Mbit/s capacity, and an average packet length of 512 bytes is used. This leads to a network service time of 0.45 milliseconds. The expected service time in the file server and local area network is thus a total of 42.9 milliseconds, since the user job travels through file server's CPU and network exactly twice, and the file server's disk once. If there is more than one disk in the file server, we assume equal probability of accessing any one of them for each user job.

6.3. Issues to Be Investigated

We first study issues in the design of the file server and the local area network from the performance viewpoint. Since these are the only resources shared and contended among workstations, a balanced design of them will provide good performance as well as permitting smooth growth in the number of workstations. A balanced design in this context means relatively equal utilizations among the shared resources. It is convenient to view the file accesses as arrivals at the file server subsystem depicted in Figure 6.2. Notice that we use two separate network servers to deal with arrivals and departures *only* for clarity in the exposition. Even though the local area network could consist of two Ethernet-like cables, one for going to the file servers and the other for returning from them, this is not the case in our model, where arrivals and departures use the same network server. Arrivals are generated by workstations, and each chain (workload type) has its own arrival rate λ depending on the interactions among all user jobs and on the characteristics of their resource demands. The arrival process cannot be easily characterized;[1] nevertheless, we can deduce a relationship among the relative utilizations of the network, the file server CPU, and the disks. Assuming that there are three chains as shown in Figure 6.2, and there are k disks in the file server, the utilizations of file server CPU, each disk, and network can be expressed as (6.1), (6.2), and (6.3).

[1]The arrival process is Poisson for a link in a single chain open network if and only if the link is not part of a loop [Wal82]; the arrival process is not Poisson for any link in a closed network.

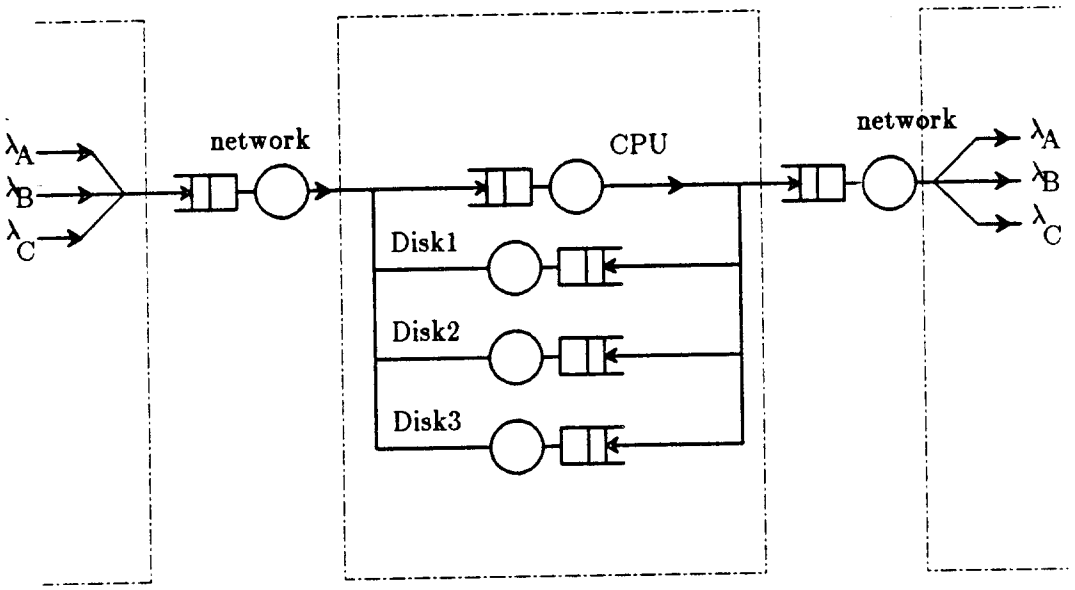

Figure 6.2 Arrivals at the File Server Subsystem

$$\rho_{cpu} = 2(\lambda_A+\lambda_B+\lambda_C)\ \frac{1}{\mu_{cpu}} \tag{6.1}$$

$$\rho_{disk} = \frac{1}{k}(\lambda_A+\lambda_B+\lambda_C)\ \frac{1}{\mu_{disk}} \tag{6.2}$$

$$\rho_{network} = 2(\lambda_A+\lambda_B+\lambda_C)\ \frac{1}{\mu_{network}} \tag{6.3}$$

Since we have the same factor $\lambda_A+\lambda_B+\lambda_C$ in all the expressions above, we can compare the relative utilizations of these resources without having to solve the model explicitly. With our model parameters, the utilization of the local area network is very small in comparison to that of the CPU, and can therefore be ignored. In order to equalize the utilizations of the CPU and the disks, k needs to be 3 with our set of model parameters. Experiments were run with different numbers of disks for the 2-chain (2 workload types) models. Similar experiments were carried out for 3-chain models as well in order to examine the effect on the results when there are more chains or workload types. Performance measures are analyzed as functions of the total number of workstations from 2 or 3 up to 24. When we add one workstation for one type of workload, we add one workstation for each of the other workload types as well in order to simplify the presentation of the results.

Intuitively, the variation of the performance measures, i.e., of the response time and of the throughput of a workstation, should depend heavily on the characteristics of its file

access behavior, since this is the only type of resource demand that needs to contend with those of other workstations in the distributed system. This leads to the idea of characterizing a "heavyweight" or "dominant" workload type; we investigate the impact of performance measures by eliminating CPU-bound (lightweight) workstations in the corresponding 2-chain models. In essence, by doing so we reduce ourselves to a one-chain model with the same number of heavyweight workstations without the less file I/O-demanding workstations. Performance measures will be computed and compared with and without the presence of the lightweight workstations in a distributed system model with a balanced file server. In particular, we will examine carefully the file server arrival rates in these two configurations.

The third area of investigation is concerned with devising a simple system-wide characterization of the workload for the workstation-based distributed systems. It is desirable and often necessary in practice to define a load measure related to the overall file server arrival rates, so that the file server and the local area network can be properly configured and designed. For instance, given a file server's characteristics and workload, how many workstations can we support before performance drops below a certain level? Based on the notion of balanced file server design and the characterization of file server arrival rates, we propose a system-wide load measure ρ defined as the sum of individual load measures; an individual load measure for a workstation is in turn defined as the utilization of the resource in the file server which is most utilized by that workstation (i.e., the most critical resource) when it is the only workstation in the system. Thus, ρ is expressed as

$$\rho = \sum_{i=1}^{n} \rho_i \ , \tag{6.4}$$

where n is the number of workstations, and $\rho_i = \dfrac{\lambda_i}{\mu_{critical}}$.

Notice that λ_i is the arrival rate of requests made to the most critical resource in the file server by workstation i. If disks are the most critical resources, λ_i also depends on the number of disks in the file server. The ρ_i's are different since λ_i's are different. In the models we experiment with, disks turn out to be the the critical resource throughout. Because of the single-workstation contention-free assumption, these individual load

Table 6.2 Load Measures for Single Workstation

Workload Type	File Server Arrival Rate (/ms)	Load Measure (one disk)	Load Measure (two disks)	Load Measure (3 disks)
A	0.0123	0.394	0.197	0.131
B	0.00118	0.0378	0.0189	0.0126
C	0.00251	0.0803	0.0402	0.0268

measures can be easily calculated by hand, and are summarized in Table 6.2 for our model parameters.

We now try to relate the system-wide load measure ρ to the delay in the file server and in the local area network. If we have an open queueing network, the sum of the individual load measures will be *the* utilization of the critical resource in the file server complex (which includes the local area network). For a closed queueing network, this sum will be only a "measure" of the load on the system. We expect that the delay in the file server complex can be approximated as a simple function of this measure. Since the workstation part, CPU and terminal, is privately owned, it is easy to calculate its (terminal) response time and its throughput when the delay in the file server and in the local area network is available.

Recall that, for a simple M/M/1 queue with arrival rate λ and service rate μ, the average waiting time (service time plus queueing time) of a customer is $\frac{1}{\mu}\frac{1}{1-\rho}$, where $\rho=\frac{\lambda}{\mu}$ is the utilization of the server [Kle75]. We can view this $\frac{1}{1-\rho}$ as a dimensionless normalized waiting time, or a "stretch" factor due to contention at the server. In a closed queueing network, there is no "absolute" arrival rate at the file server subsystem (see Figure 6.2), since the delay in the subsystem will be fed back to the workstation subsystem, and the arrival rate will be automatically reduced to achieve equilibrium. Although we do not anticipate that the stretch factor will have as simple a form as in the M/M/1 queue, we attempt to use the system-wide load measure to capture and approximate the effect of delay in the file server and local area network of the distributed system.

We shall plot, as functions of the proposed system-wide load measure, normalized waiting times based on the file server waiting time data obtained from the experiments carried out in this study. A simple approximation of normalized waiting time as a function of the system-wide load measure ρ will be proposed so that first-order capacity planning in the workstation-based distributed system can be carried out. Although this functional approximation is not immediately transportable to other configurations, the idea of the system-wide load measure and the methodology are very general and are expected to be applicable to many similar systems.

For convenience, we summarize the relevant single-workstation contention-free performance measures in Table 6.3. Similar to those load measures in Table 6.2, these

Table 6.3 Single-Workstation (contention-free) Performance Measures

Workload Type	Response Time (s)	Workstation CPU Time (s)	File Server and Network Time (s)	Throughput (/s)
A	7.547	3.300	4.247	0.124
B	0.844	0.700	0.144	0.351
C	0.514	0.353	0.161	0.660

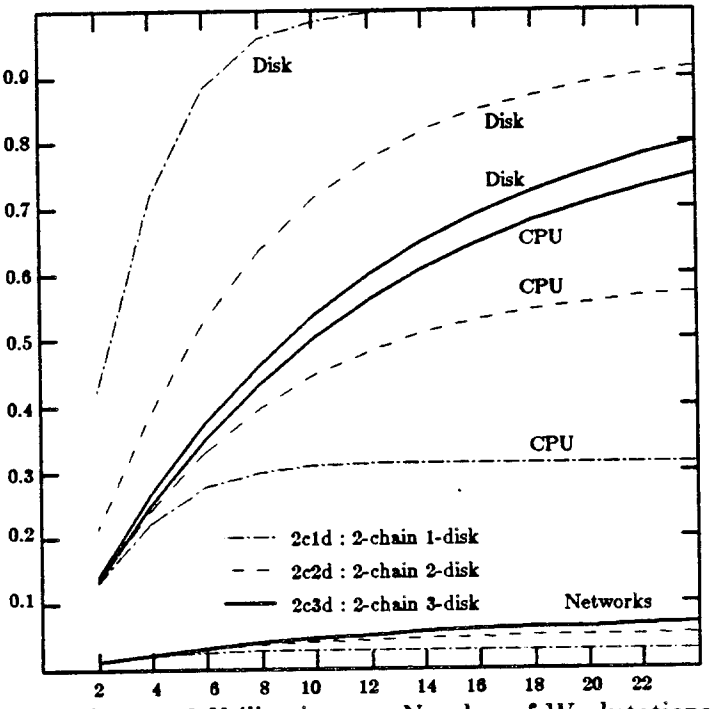

Figure 6.3 Utilizations vs. Number of Workstations

response times are in seconds

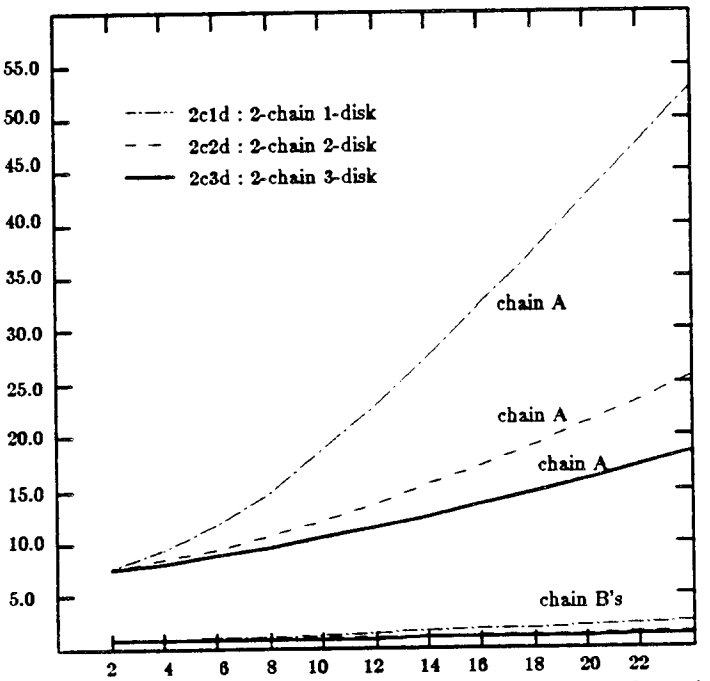

Figure 6.4 Response Times vs. Number of Workstations

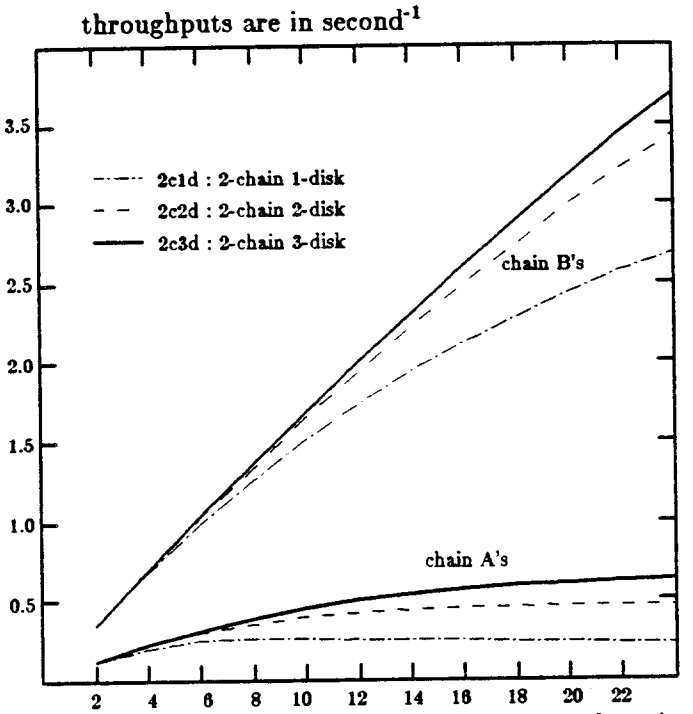

Figure 6.5 Throughputs vs. Number of Workstations

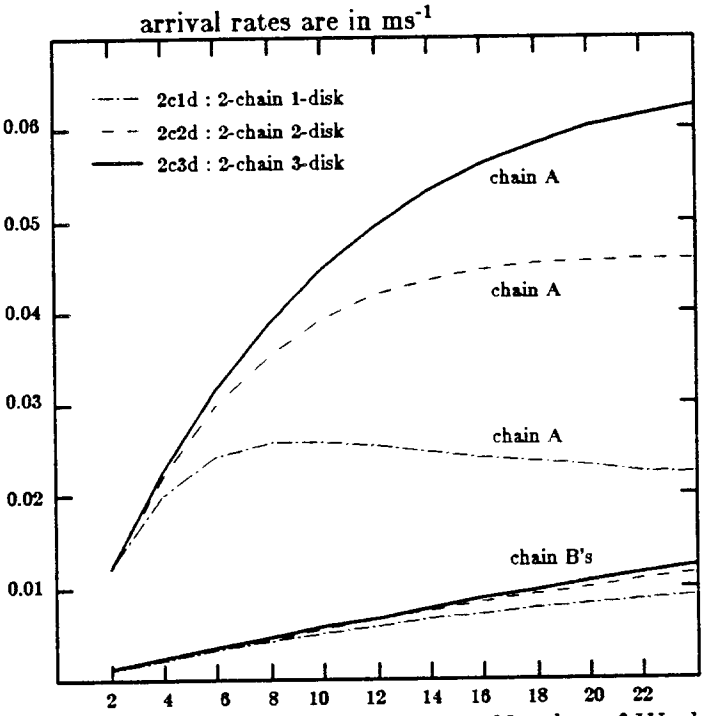

Figure 6.6 File Server Arrival Rates vs. Number of Workstations

performance measures can be easily derived from the parameters in Table 6.1 and from other model parameters. The normalized waiting time is then computed by dividing the actual time spent in the network and in the file server by the minimum (single-workstation) service time needed, that is available from Table 6.3. This will generally be some complicated function of our load measure, but we hope to be able to approximate it with some simple form.

6.4. Analysis of the Results

6.4.1. Balanced File Server

Experiments were run with one, two, and three disks as well as with different numbers of workstations for the 2-chain models. The utilizations of various resources are shown in Figure 6.3. Response times and throughputs are shown in Figures 6.4 and 6.5. It is apparent that balanced utilizations among resources in the file server are important for both of these performance measures. The file server arrival rates are important characterizations for the workload types. They are plotted in Figure 6.6. Notice that the file server arrival rates can actually drop as the total number of workstations increases due to contention from *different* types of workstations.

In order to mix in more workload types, a third intermediate type (C) was added to the model to study the effect of finer workload clustering on the performance measures. The functional behaviors of the performance measures versus the total number of workstations and versus the number of disks in the file server were found to be similar to those of the 2-chain models. They are shown in Figures 6.7, 6.8, 6.9, and 6.10. The extremely unbalanced one-disk case is not included in these figures to avoid overcrowding. The values of the performance measures corresponding to the one-disk case are not difficult to determine by extrapolation.

6.4.2. Dominant Workload Type

In Figures 6.11, 6.12, 6.13, and 6.14, we compare the utilizations, response times, throughputs, and file server arrival rates between models with 2 types of workstations and models with only heavyweight, i.e., type A, workstations. For the purposes of presentation and comparison, we report on the x-axis the total number of workstations in the 2-chain models. The number of workstations in one-chain models is only half of the corresponding value on the x-axis. For instance, if the number of workstations for the 2-chain models is 10, we mean that there are 5 type A workstations and 5 type B workstations. Corresponding to the same label 10, in the one-chain model we have 5 type A workstations and no type B workstations.

This simple experiment confirms our earlier argument that the dominant type of workload or workstation can be well characterized by its file I/O demand behavior. In particular, the dominance is reflected in the file server arrival rates shown in Figure 6.14. This is also evident in Figures 6.6 and 6.10. This observation led us to the idea of defining a simple system-wide measure related to these file server arrival rates for the workload as a whole, the "load measure" introduced in Section 6.3.

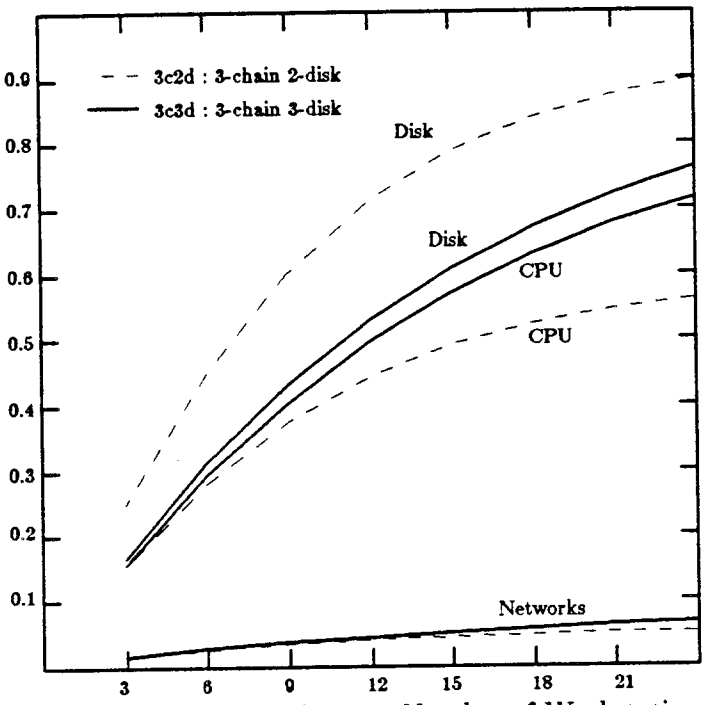

Figure 6.7 Utilizations vs. Number of Workstations

response times are in seconds

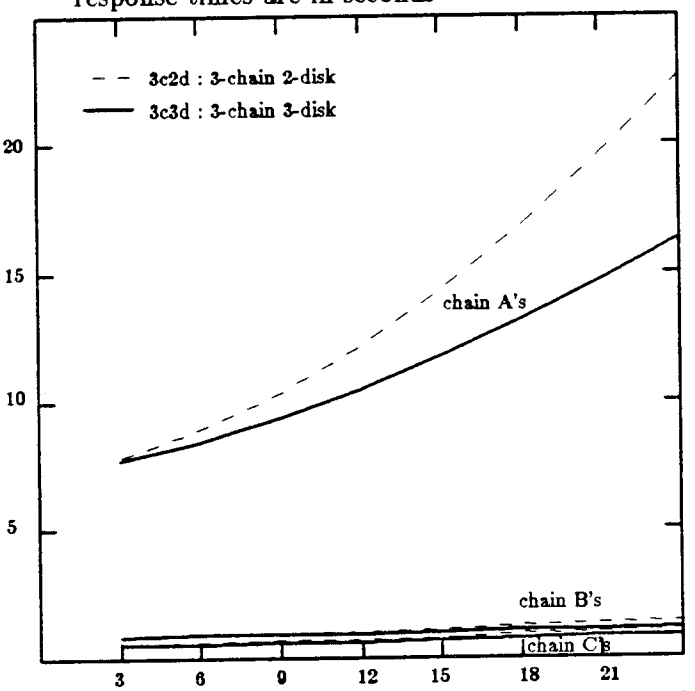

Figure 6.8 Response Times vs. Number of Workstations

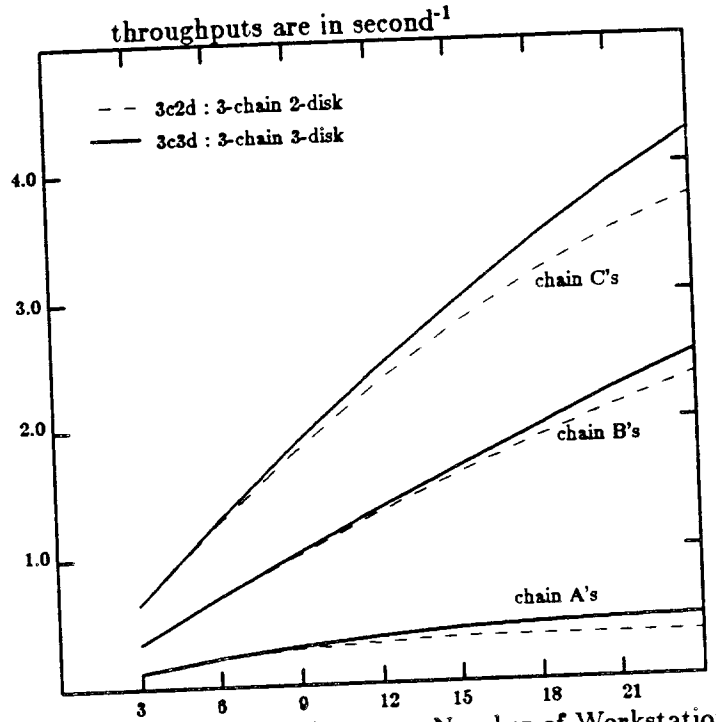

Figure 6.9 Throughputs vs. Number of Workstations

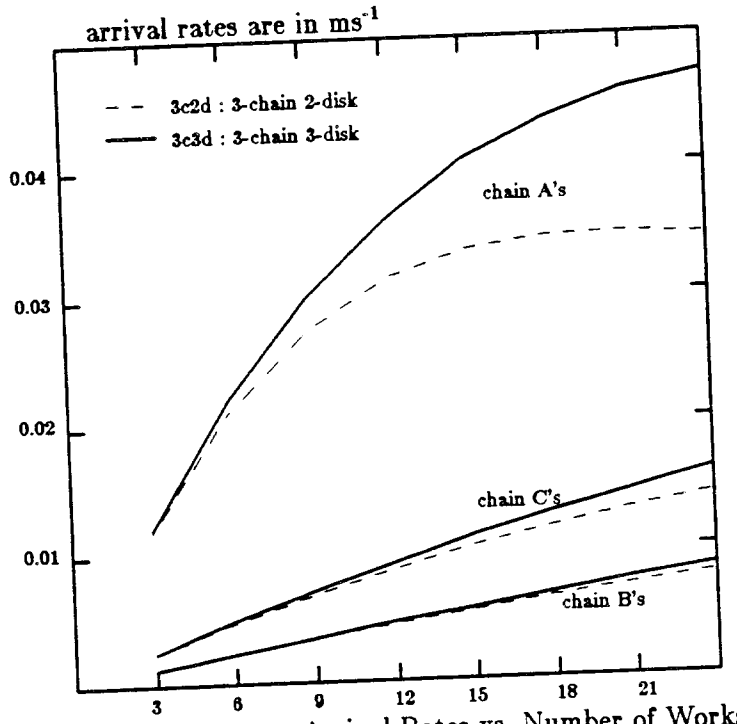

Figure 6.10 File Server Arrival Rates vs. Number of Workstations

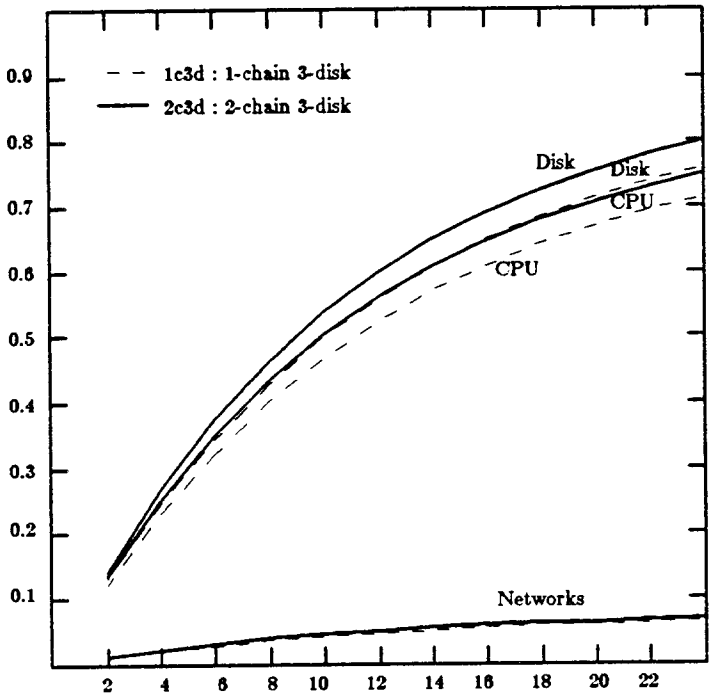

Figure 6.11 Utilizations vs. Number of Workstations

response times are in seconds

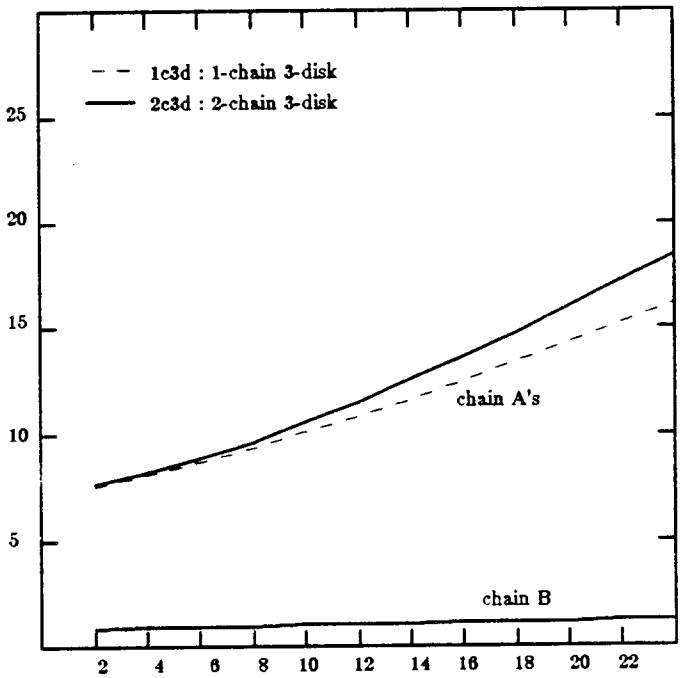

Figure 6.12 Response Times vs. Number of Workstations

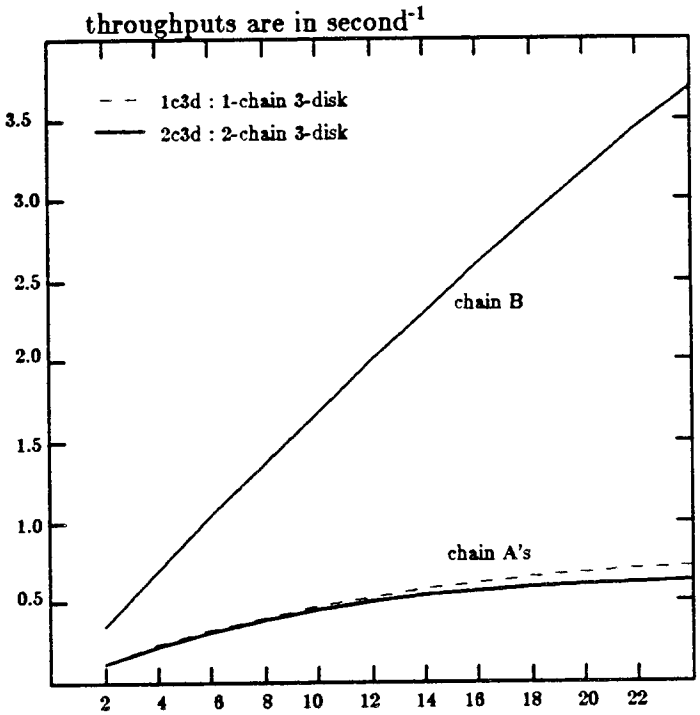

Figure 6.13 Throughputs vs. Number of Workstations

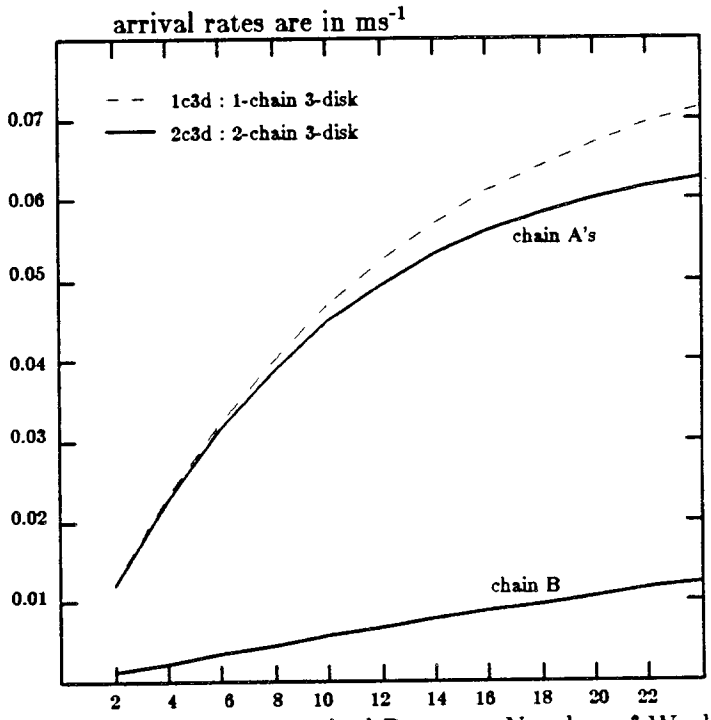

Figure 6.14 File Server Arrival Rates vs. Number of Workstations

6.4.3. A Simple Characterization of the Workload

The file server waiting times are plotted as functions of the system-wide load measure in Figures 6.15 and 6.16. The curves in those diagrams possess similar *forms*; when the system-wide load measure is slightly above one, the values of waiting times reported there are about twice the minimum service times required in the file server. We choose to consider a value of load measure near 1 as the typical operating load[2]. In systems with load measures significantly greater than one, the (terminal) response times and throughputs will become completely unacceptable. For load measures much less than one, there will be no performance problem since the file server is under-utilized.

In order to characterize the throughput and predict (terminal) response times for other workload types, it is necessary to consider the normalized (file server) waiting time as a function of the load measure. We plot the normalized waiting time as a function of our load measure in Figures 6.17 and 6.18. Two figures are used to facilitate the presentation of many curves. We also show on both diagrams the functions $1+\rho$ and $1+\rho^2$. The function $1+\rho^2$ approximates the normalized waiting time for the 3-chain models quite well. The accuracy of this approximation is very reasonable also near the typical operating load for the 2-chain and 1-chain models. The over-estimation beyond a load measure of 1.5 should be acceptable to conservative system designers. We can employ this simple rule of thumb to calculate the response time and throughput for various workload types, and plan a workstation-based distributed system accordingly.

Figure 6.19 is the plot of disk utilization as a function of the load measure. (Recall that disks are the most critical resource in our models.) For the models with balanced or near-balanced file server, the functional relationship $1-e^{-\rho}$ seems to provide a reasonable approximation and a close lower bound function. Only in the models with extremely unbalanced file server design this functional relationship underestimates the disk utilization, or overestimates the load measure corresponding to a given device utilization. This large deviation is anticipated, and the unbalanced configuration in the file server which causes this large deviation should probably be the first thing to correct in the design and operation of such systems.

It is in practice very easy to obtain device utilizations in the running system. This allows a simple but conservative estimation of the load measure, which in turn can be used to approximate file server waiting times and to calculate terminal response times and throughputs. To this extent, this system-wide load measure may be very useful in planning workstation-based distributed systems.

---

[2]The $\rho < 1$ condition is referred to as "normal usage" in [McK82].

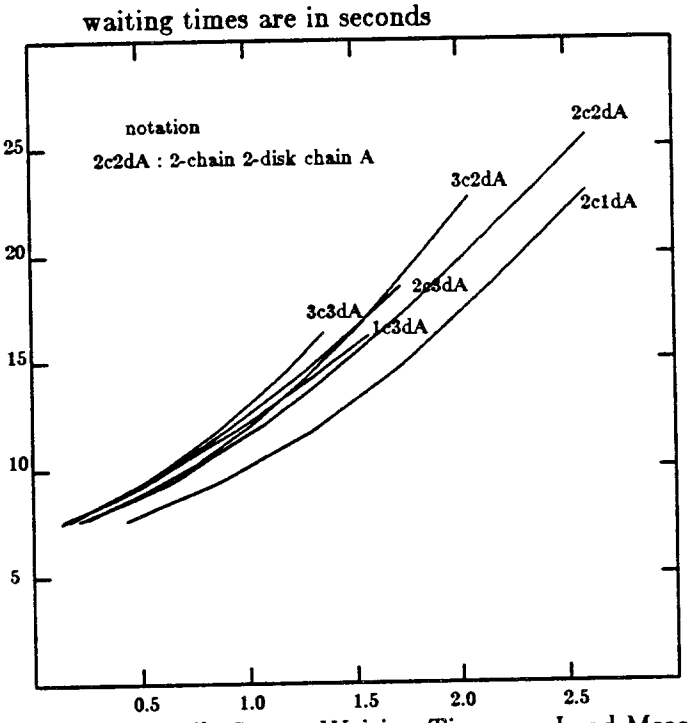

Figure 6.15 File Server Waiting Times vs. Load Measure

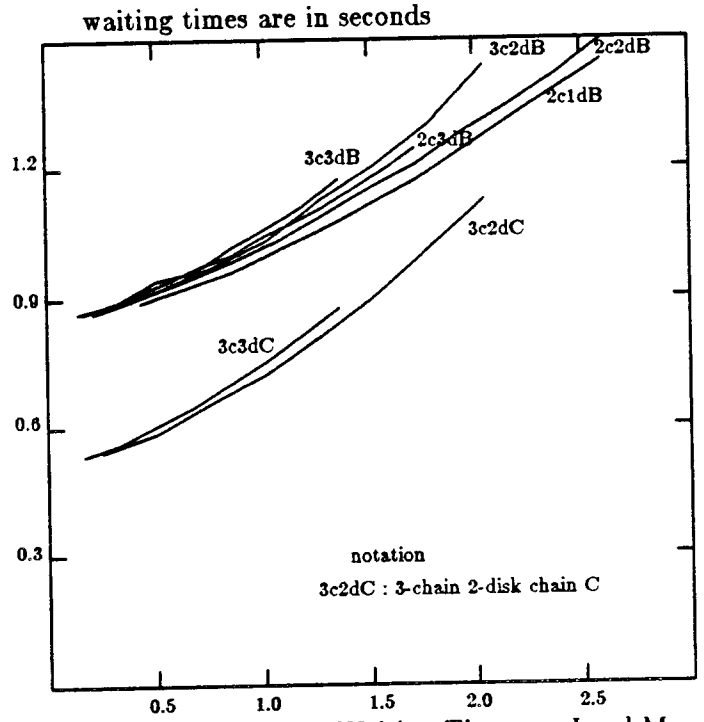

Figure 6.16 File Server Waiting Times vs. Load Measure

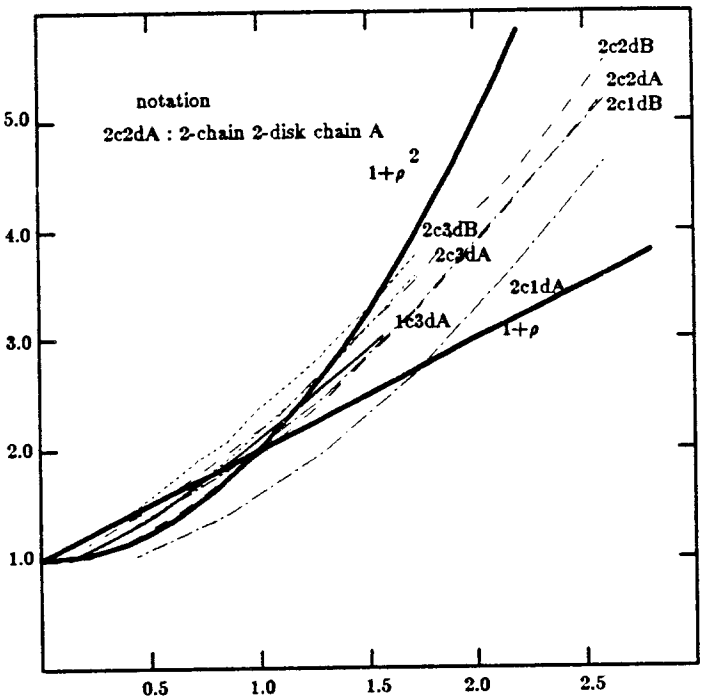

Figure 6.17 Normalized Waiting Time Vs. Load Measure

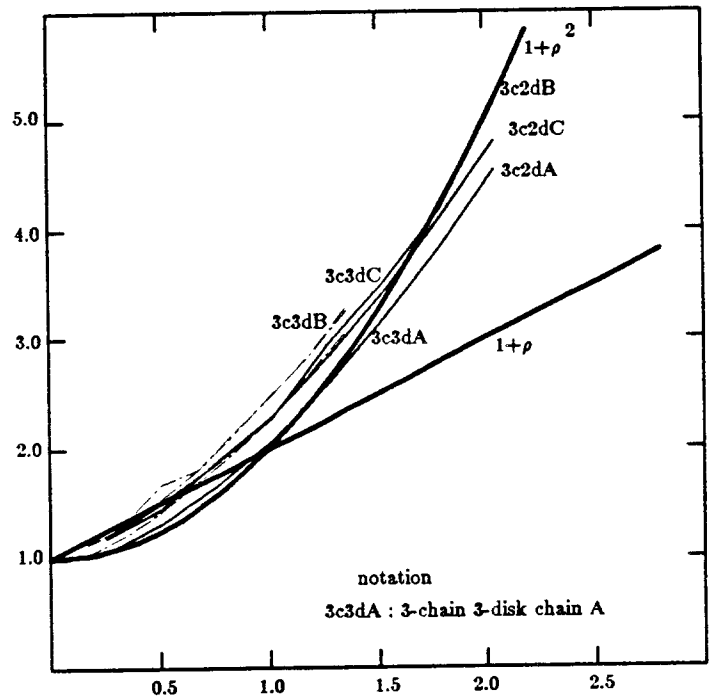

Figure 6.18 Normalized Waiting Time vs. Load Measure

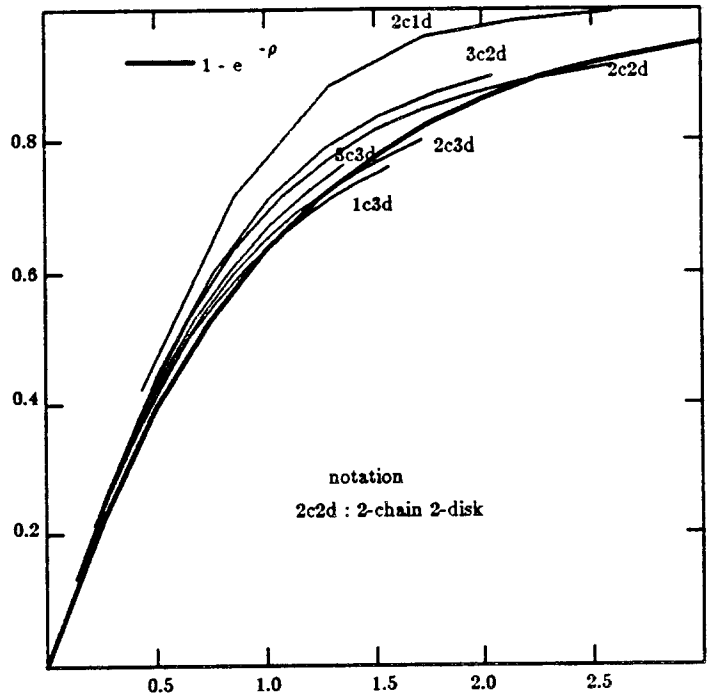

Figure 6.19 Disk Utilizations vs. Load Measure

CHAPTER 7

Conclusions

7.1. Summary

We have presented an iterative methodology for configuring local area network-based distributed systems. The characterization of the system's workload was heavily influenced by our assumption that the system is an interactive transaction-oriented business computer system. Since shared files are of great concern in a distributed environment, two primitive metrics based on a simple user-object model were proposed to capture the spatial component of file sharing. The workload of business computer installations is naturally clustered into several major types of transactions, corresponding to different business activities. The same type of characterization becomes more involved when applied to a non-business environment, mainly because of the need to cluster and identify workload types. A set of measurements taken on a medium-sized interactive transaction-oriented computer system was used to derive parameters for the various models employed in our configuration design methodology.

The methodology consists of two parts : the assignment of transaction types and computing resources, and the analysis of a queueing network model. The first part of the methodology is aimed at balancing the CPU loads among the various host systems and minimizing the total number of remote file accesses in the distributed system. This task is inherently complicated since it is difficult, to say the least, to express in some simple functional form the impact of resource contention on the performance measures of various transactions in the distributed system. The initial assignment of computing resources and transaction types (workload) is based on the assumption of a uniprogramming environment. Queueing network models are employed in the second part of the methodology to determine stochastically the impact of contention on performance measures. The results of the analysis, particularly the queueing delays and transaction throughputs, are fed back into the first part of the configuration methodology; if necessary, the assignments of transactions and computing resources in the proposed distributed configuration are modified. This methodology promises to become an indispensable tool for the system designer if we integrate these models into monitoring and measurement facilities in a running system. Periodic reconfiguration of the distributed system due to shifts in workload characteristics can thus be carried out efficiently.

Attempts to break the global resource demands of each transaction into several phases resulted in the introduction of a special class of queueing networks, the phase-free queueing networks. Product-form results from Kelly's work were introduced in Chapter 5 to show the equivalence in terms of some important performance measures between network models based on a simple phase-free workload characterization and network

models based on a multi-phase workload characterization. This treatment not only explains the results obtained by characterizing the workload as consisting of several phases, but also simplifies some of the computational efforts involved in getting the equilibrium solution of a queueing network.

Lastly, we focused on the issues that arise in the design of a special type of local area network-based distributed system, i.e., a distributed system consisting of workstations and file servers. Guidelines for designing a balanced file server were derived. A very simple workload characterization for a workstation was proposed, i.e., that of single-workstation's file server arrival rate. A system-wide load measure based on these simple individual workload characterizations was proposed and evaluated experimentally for possible use in capacity planning. We also showed how queueing network models can be used in configuring workstation-based distributed systems.

7.2. Directions for Future Research

The areas for research in the field of local area network-based distributed systems are many. We shall briefly outline and discuss some of them from a performance and modeling perspective.

7.2.1. Workload Data Measurements and Monitoring

There have been only a few reports on, and analyses of, workload data measurements obtained from operational distributed systems [McD75]. This is in part due to the small number of such systems, and in part to the inherent complexity of collecting and coordinating data obtained from various computer systems connected by a network. Further work in this area is needed to increase the level of our understanding of distributed systems, and to validate various models that have been proposed for the distributed systems. Tools for understanding and characterizing distributed programs [Mil84] will provide insights into the applicability of distributed computing to various problems.

The incorporation of network monitoring mechanisms into an operational distributed system will facilitate the task of file migration (i.e., file re-assignment) due to shifts of workload characteristics, and the task of process migration [Pow83] to achieve load balancing [Alo84] among computer systems connected by the network. Process migration can be most easily implemented in the context of interactive transaction-oriented business systems as transaction re-assignment.

7.2.2. Dynamic Reconfiguration

The workload will never remain constant, and the various components in a distributed system will not always remain operational. Many applications require a highly available system [Bar78]. The need for dynamic reconfiguration is evident. Data replication and multipathing for resource access have been widely studied at the functional level [Bar80, Wal83]. The tradeoff between performance degradation and availability improvement due to such approaches needs further quantitative study.

From a performance point of view, the issue of load balancing is of great interest. The load metric, the load forecasting technique, and the control policies are all important

components of an effective load balancing mechanism. The issues of file migration and file caching are interesting since programs in a distributed environment are expected to have both spatial and temporal locality in their data accessing behavior. The degree of file replication, the granularity of data caching, and the file migration policies [Smi81, Por82] should be investigated. The effectiveness of various algorithms in the context of local area network-based distributed systems needs to be evaluated.

In essence, there are three basic areas that can be "reconfigured" to either improve system availability and/or increase performance. Computing resources are subject to reconfiguration; data files need to be replicated and migrated; programs should be migrated to balance the utilizations of various computing resources. In the context of local area network-based distributed systems, all these issues are of great interest and have important practical application.

7.2.3. Performance Impact of Serialization Delays

We have not represented in our configuration models the overhead due to the synchronization of accesses to shared files. The performance impact of serialization delays is of great interest to the methodology presented in this dissertation as well as to the general context of distributed processing [Hać83]. It is intuitive to expect that the degradation of performance due to synchronization will depend heavily on the granularity of locks and on the amount of sharing [Rie77]. When the locking granule is a file record or a physical disk sector which may contain a few records, the probability of actual conflict between concurrent transactions sharing the same file may be quite small. Measurements of record/sector accesses by concurrent transactions will be of interest in performance modeling studies. That of the performance impact of other concurrency mechanisms such as timestamping in comparison with locking is an interesting study that has attracted little attention.

The problem of consistency among replicated files needs to be examined with care. Models that reflect the replication of shared files for fast access are good candidates for future research in this area.

BIBLIOGRAPHY

[Ack82]
R. D. Acker and P. H. Seaman, "Modeling Distributed Processing across Multiple CICS/VS Sites," IBM System Journal, 21, 4, 1982, pp. 471-489.

[Alm79]
G. T. Almes and E. D. Lazowska, "The Behavior of Ethernet-like Computer Communications Networks," Proceedings of the 7th Symposium on Operating System Principles, 1979, pp. 66-81.

[Alo84]
R. Alonso, "Query Optimization in Distributed Databases Through Load Balancing," Ph.D. dissertation, Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in preparation.

[Bar78]
J. F. Bartlett, "A NonStop Operating System," Proc. Eleventh Hawaii International Conference on System Sciences, Vol. III, 1978, pp. 103-117.

[Bar80]
Y. Bard, "A Model of Shared DASD and Multipathing," Comm. of ACM, 23, 10, October 1980, pp. 564-572.

[Bas75]
F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios, "Open, Closed, and Mixed Networks of Queues with Different Classes of Customers," Journal of ACM, 22, 2, April 1975, pp. 248-260.

[Bec82]
A. Bechtolsheim, F. Baskett, and V. Pratt, "The SUN Workstation Architecture," Stanford Computer Systems Laboratory Technical Report No. 229, March 1982, 15 pp.

[Bel62]
R. E. Bellman and S. E. Dreyfus, "Applied Dynamic Programming," Princeton University Press, Princeton, New Jersey, 1962, 363 pp.

[Bir82]
A. D. Birrell, R. Levin, R. M. Needham, and M. D. Schroeder, "Grapevine : An Exercise in Distributed Computing," Comm. of ACM, 25, 4, April 1982, pp. 260-274.

[Boy75]
J. H. Boyse and D. R. Warn, "A Straightforward Model for Computer Performance Prediction," ACM Computing Surveys, 7, 2, June 1975, pp. 73-93.

[Bra84]
J. Brann, "MIT Goes on the 5-year Plan," PC, 3, 5, March 1984, pp. 269-279.

[Buz71]
J. P. Buzen, "Analysis of System Bottlenecks Using a Queueing Network Model," Proc. ACM-SIGOPS Workshop on System Performance Evaluation, Harvard University, April 1971, pp. 82-103.

[Buz73]
J. P. Buzen, "Computational Algorithms for Closed Queueing Networks with Exponential Servers," Comm. of ACM, 16, 9, September 1973, pp. 527-531.

[Buz78]
J. P. Buzen, "A Queueing Network Model of MVS," ACM Computing Surveys, 10, 3, September 1978, pp. 319-331.

[Cas72]
R. G. Casey, "Allocation of Copies of a File in an Information Network," Proc. AFIPS 1972 SJCC, 40, pp. 617-625.

[Cha75]
A. K. Chandra and C. K. Wong, "Worst-Case Analysis of a Placement Algorithm Related to Storage Allocation," SIAM J. Computing, 4, 3, September 1975, pp. 249-263.

[Chu69]
W. W. Chu, "Optimal File Allocation in a Multiple Computer System," IEEE Trans. on Computers, C-18, 10, October 1969, pp. 885-889.

[Chu80]
W. W. Chu, L. J. Holloway, M. T. Lan, and K. Efe, "Task Allocation in Distributed Data Processing," Computer, 13, 11, November 1980, pp. 57-69.

[Cla78]
D. D. Clark, K. T. Pogran, and D. P. Reed, "An Introduction to Local Area Network," Proc. IEEE, 66, 11, November 1978, pp.1497-1517.

[Dei82]
M. Deitch, "Analytic Queuing Model for CICS Capacity Planning," IBM System Journal, 21, 4, 1982, pp. 454-470.

[Den81]
L. A. Denoia, "An Approach to the Cost/Performance Comparison of Distributed Systems," Proc. of the 5th Berkeley Workshop on Distributed Data Management and Computer Networks, February 1981, pp. 38-64.

[Dow82]
L. W. Dowdy, and D. V. Foster, "Comparative Models of the File Assignment Problem," ACM Computing Surveys, 14, 2, June 1982, pp. 287-313.

[Ens78]
P. H. Enslow, Jr., "What is a 'Distributed' Data Processing System?," Computer, 11, 1, January 1978, pp. 13-21.

[Eth80]
"The Ethernet, A Local Area Network : Data Link Layer and Physical Layer Specifications," Version 1.0, September 1980, Digital Equipment Corporation, Intel Corporation, and Xerox Corporation, 82 pp.

[Fer78]
D. Ferrari, "Computer Systems Performance Evaluation," Prentice-Hall, Englewood Cliffs, New Jersey, 554 pp, 1978.

[Fer82]
D. Ferrari, "On the Foundations of Artificial Workload Design," UCB/CSD 82/110, Computer Science Division (EECS), University of California at Berkeley, November 1982, 12 pp.; to appear in Proc. of the 1984 SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Cambridge, Massachusetts, August 1984.

[Fos81]
D. V. Foster, L. W. Dowdy, and J. E. Ames, IV, "File Assignment in a Computer Network," Proc. of the 5th Berkeley Workshop on Distributed Data Management and Computer Networks, February 1981, pp. 259-283. A concise version also appears in Computer Networks, 5, 5, September 1981, pp. 341-349.

[Fri81]
M. Fridrich and W. Older, "The FELIX File Server," Proc. of the 8th Symposium on Operating Systems Principles, December 1981, pp. 37-44.

[Gar79]
M. R. Garey and D. S. Johnson, "Computers and Intractability : A Guide to the Theory of NP-Completeness," W. H. Freeman and Co., San Francisco, 1979, 340 pp.

[Gif81]
D. K. Gifford, "Violet, an Experimental Decentralized System," Computer Networks, 5, 6, December 1981, pp. 423-433.

[Gol83]
A. Goldberg, G. Popek, and S. Lavenberg, "A Validated Distributed System Performance Model," Performance '83, edited by A. K. Agrawala and S. K. Tripathi, North-Holland, 1983, pp. 251-268.

[Goo83]
J. R. Good, "Experience with a Large Distributed Banking System," Database Engineering, 6, 2, June 1983, pp. 50-56.

[Hać83]
A. Hać, "On the Modeling of Shared Resources by Queueing Networks with Serialization Delays," Computer Science Division Report No. UCB/CSD 83/153, University of California at Berkeley, November 1983.

[Hor78]
E. Horowitz and S. Sahni, "Fundamentals of Computer Algorithms," Computer Science Press, Rockville, Maryland, 1978, 626 pp.

[Isr78]
I. E. Israel, J. G. Mitchell, and H. E. Sturgis, "Separating Data from Function in a Distributed File System," Palo Alto Research Center Report CSL-78-5, September 1978, 13 pp.

[Kar72]
R. M. Karp, "Reducibility among Combinatorial Problems," Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85-103.

[Kay77]
A. C. Kay, "Microelectronics and the Personal Computer," Scientific American, September 1977, pp. 231-244.

[Kel79]
F. P. Kelly, "Reversibility and Stochastic Networks," John Wiley & Sons, New York, 230 pp, 1979.

[Kle75]
L. Kleinrock, "Queueing Systems, Vol. 1 : Theory," John Wiley & Sons, New York, 417 pp, 1975.

[Koh81]
W. H. Kohler, " A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems," Computing Surveys, 13, 2, June 1981, pp. 149-184.

[Lam83]
S. S. Lam and Y. L. Lien, "A Tree Convolution Algorithm for the Solution of Queueing Networks," Comm. of ACM, 26, 3, March 1983, pp. 203-215.

[Law76]
E. L. Lawler, "Combinatorial Optimization : Networks and Matroids," Holt, Rinehart and Winston, New York, 1976, 374 pp.

[Laz81]
E. D. Lazowska, H. M. Levy, G. T. Almes, M. J. Fischer, R. J. Fowler, and S. C. Vestal, "The Architecture of the Eden System," Proc. of the Eighth Symposium on Operating Systems Principles. December 1981, pp. 148-159.

[Lud81]
G. W. R. Luderer, H. Che, J. P. Haggerty, P. A. Kirslis, and W. T. Marshall, "A Distributed UNIX System based on a Virtual Circuit Switch," Proc. of the Eighth Symposium on Operating Systems Principles. December 1981, pp. 160-168.

[Mah76]
S. Mahmoud and J. S. Riordon, "Optimal Allocation of Resources in Distributed Information Network," ACM Trans. on Database Systems, 1, 1, March 1976, pp. 66-78.

[McD75]
G. McDaniel, "METRIC : A Kernel Instrumentation System for Distributed Environments," Proceedings of the 6th Symposium on Operating System Principles,

November 1975, pp. 93-99.

[McK82]
J. McKenna and D. Mitra, "Integral Representations and Asymptotic Expansions for Closed Markovian Queueing Networks : Normal Usage," Bell System Technical Journal, 61, 1982, pp. 661-683.

[Met76]
R. M. Metcalfe and D. R. Boggs, "Ethernet : Distributed Packet Switching for Local Computer Networks," Comm. of ACM, 19, 7, July 1976, pp. 395-404.

[Mil84]
B. P. Miller, "Performance Characterization of Distributed Programs," Ph.D. dissertation, Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, May 1984.

[Mit82]
J. Mitchell and J. Dion, "A Comparison of Two Network-Based File Servers," Comm. of ACM, 25, 4, April 1982, pp. 233-245.

[Moo71]
C. G. Moore, III, "Network Models for Large-scale Time-sharing Systems," Tech. Rep. No. 71-1, Department of Industrial Engineering, University of Michigan, April 1971.

[Mor77]
H. L. Morgan and K. D. Levin, "Optimal Program and Data Locations in Computer Networks," Comm. of ACM, 20, 5, May 1977, pp. 315-322.

[Opp83]
D. C. Oppen and Y. K. Dalal, "The Clearinghouse : A Decentralized Agent for Locating Named Objects in a Distributed Environment," ACM Transactions on Office Information Systems, 1, 3, July 1983, pp. 230-253.

[Par82]
D. S. Parker, and R. A. Ramos, "A Distributed File System Architecture Supporting High Availability," Proc. of the 6th Berkeley Workshop on Distributed Data Management and Computer Networks, February 1982, pp. 161-183.

[Pec81]
M. A. Pechura, "Microcomputers as Remote Nodes of a Distributed System," Comm. of ACM, 24, 11, November 1981, pp. 734-738.

[Pop81]
G. Popek, B. Walker, J. Chow, D. Edwards, C. Kline, G. Rudisin, and G. Thiel, "LOCUS : A Network Transparent, High Reliability Distributed System," Proc. of the Eighth Symposium on Operating Systems Principles. December 1981, pp. 169-177.

[Por82]
J. M. Porcar, "File Migration in Distributed Computer Systems," Ph.D. dissertation, Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, July 1982, 151 pp.

[Pow83]
M. L. Powell and B. P. Miller, "Process Migration in DEMOS/MP," Proc. of the Ninth Symposium on Operating Systems Principles, October 1983, pp. 110-119.

[Rei75]
M. Reiser and H. Kobayashi, "Queueing Networks with Multiple Closed Chains : Theory and Computational Algorithms," IBM Journal of Research and Development, 19, 3, May 1975, pp. 283-294.

[Rei79]
M. Reiser, "A Queueing Network Analysis of Computer Communication Networks with Window Flow Control," IEEE Trans. on Communications, Com-27, 8, August 1979, pp. 1199-1209.

[Rei80]
M. Reiser and S. S. Lavenberg, "Mean-Value Analysis of Closed Multichain Queuing Networks," Journal of ACM, 27, 2, April 1980, pp. 313-322.

[Rei81]
M. Reiser, "Mean-Value Analysis and Convolution Method for Queue-Dependent Servers in Closed Queueing Networks," Performance Evaluation, 1, 1, January 1981, pp. 7-18.

[Rie77]
D. R. Ries and M. R. Stonebraker, "Effects of Locking Granularity in a Database Management System," ACM Trans. on Database Systems, 2, 3, September 1977, pp. 233-246.

[Sal75]
H. M. Salkin, "Integer Programming," Addison-Wesley, Reading, Mass., 1975.

[Sau82]
C. H. Sauer, E. A. MacNair, and J. F. Kurose, "The Research Queueing Package, Version 2 : CMS Users Guide," IBM Research Report RA 139 (#41127), April 1982.

[Sch78]
A. L. Scherr, "Distributed Data Processing," IBM System Journal, 17, 4, 1978, pp. 324-343.

[Sho80]
J. F. Shoch and J. A. Hupp, "Measured Performance of an Ethernet Local Network," Comm. of ACM, 23, 12, December 1980, pp. 711-721.

[Sin80]
W. David Sincoskie and D. J. Farber, "SODS/OS : A Distributed Operating System for the IBM Series/1," Operating Systems Review, 14, 3, July 1980, pp. 46-54.

[Smi81]
A. J. Smith, "Long Term File Migration : Development and Evaluation of Algorithms," Comm. of ACM, 24, 8, August 1981, pp. 512-532.

[Sto77]
H. S. Stone, "Multiprocessor Scheduling with the Aid of Network Flow Algorithms," IEEE Trans. on Software Engineering, SE-3, 1, January 1977, pp. 85-93.

[Sto76]
M. Stonebraker, E. Wong, P. Kreps, and G. Held, "The Design and Implementation of INGRES," ACM Transactions on Database Systems, 1, 3, September 1976, pp. 189-222.

[Stu80]
H. Sturgis, J. Mitchell, and J. Israel, "Issues in the Design and Use of a Distributed File System," Operating Systems Review, 14, 3, July 1980, pp. 55-69.

[Tan81]
A. S. Tanenbaum, "Computer Networks," Prentice-Hall, Englewood Cliffs, New Jersey, 517 pp, 1981.

[Ter84]
D. B. Terry, "Distributed Name Servers : A Uniform Approach to Managing Objects in Computer Internetworks," Ph.D. dissertation, Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in preparation.

[Tha82]
C. P. Thacker, E. M. McCreight, B. W. Lampson, R. F. Sproull, and D. R. Boggs, "Alto : A Personal Computer," appears in "Computer Structures : Principles and Examples," by D. P. Siewiorek, C. G. Bell, and A. Newell, McGraw-Hill, 926 pp, 1982, Chapter 33, pp. 549-572.

[Thu80]
K. J. Thurber, "A Survey of Local Network Hardware," Proc. of Distributed Computing, COMPCON Fall 1980, pp. 273-275.

[Tho84]
K. Thompson, "Macintosh : Apple's Saucy New Micro," Microcomputing, March 1984, pp. 66-73.

[Tri80]
K. S. Trivedi, R. A. Wagner, and T. M. Sigmon, "Optimal Selection of CPU Speed, Device Capacities, and File Assignments," Journal of ACM, 27, 3, July 1980, pp. 457-473.

[Wah84]
B. W. Wah, "File Placement on Distributed Computer Systems," Computer, 17, 1, January 1984, pp. 23-32.

[Wal82]
J. Walrand, "Poisson Flows in Single Class Open Networks of Quasireversible Queues," Stochastic Processes and Their Applications, 13, 1982, pp. 293-303.

[Wal83]
B. Walker, G. Popek, R. English, C. Kline, and G. Thiel, "The LOCUS Distributed Operating System," Proceedings of the Ninth ACM Symposium on Operating Systems Principles, October 1983, pp. 49-70.

[Whi83]
S. Whitford, "Measurement Data of a Transaction-oriented Business System,"

unpublished notes, NCR Corporation, 1983.

[Wil81]
R. Williams, D. Daniels, L. Haas, G. Lopis, B. Lindsay, P. Ng, R. Obermack, P. Selinger, A. Walker, P. Wilms, and R. Yost, "R* : An Overview of the Architecture," IBM Research Report RJ3325, San Jose, California, December 1981.

[Wil83]
G. Williams, "The Lisa Computer System," Byte, 8, 2, February 1983, pp. 33-50.

[Wit83]
I. H. Witten, G. M. Birtwistle, J. Cleary, D. R. Hill, D. Levinson, G. Lomow, R. Neal, M. Peterson, B. W. Unger, and B. Wyvil, "Jade : A Distributed Software Prototyping Environment," Operating Systems Review, 17, 3, July 1983, pp. 10-23.

[Wu80]
S. B. Wu and M. T. Liu, "Assignment of Tasks and Resources for Distributed Processing," Proc. of Distributed Computing, COMPCON Fall 1980, pp. 655-662.

[Zie79]
K. Ziegler, Jr., "A Distributed Information System Study," IBM System Journal, 18, 3, 1979, pp. 374-401.