# DYNAMIC LOAD BALANCING IN DISTRIBUTED HETEROGENEOUS COMPUTER SYSTEMS

by

**Tze-Hore Howard Liu**

**Technical Report CRI-88-43**

---

A Dissertation Presented to the

FACULTY OF THE GRADUATE SCHOOL

UNIVERSITY OF SOUTHERN CALIFORNIA

In Partial Fulfillment of the
Requirements for the Degree

DOCTOR OF PHILOSOPHY
(Electrical Engineering)

May 1988

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90089

*This dissertation, written by*

Tze-Hore Howard Liu

*under the direction of h..is..... Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of*

*DOCTOR OF PHILOSOPHY*

Barbara Solomon
*Dean of Graduate Studies*

*Date* March 10, 1988

DISSERTATION COMMITTEE

*Chairperson*

C. S. Raghavendra

*The fear of the Lord is the beginning of wisdom;*

*all who follow his precepts have good understanding.*

*To him belongs eternal praise.*

— Psalm 111;10

# Acknowledgements

I would like to express my gratitude to my advisor, Dr. J. A. Silvester, who initially suggested that I investigate the load-balancing problem and also provided encouragement and valuable advice throughout the course of this work.

My dissertation committee (Dr. C.S. Raghavendra and Dr. M.D. Huang) must be commended for the efficiency with which they read my thesis and gave their valuable suggestions.

My fellow graduate students in the Computer Engineering Department, Jeff Dill, Jonathon Wang, Syu-Je Wang, Thomas Pappavassilliou, Anastasios Economides, and John Yang were always a source of inspiration and support. In particular, I would like to thank Chin Yuan for many stimulating discussions.

I would like to acknowledge the financial support of the Jet Propulsion Laboratory and the encouragement of JPL management. Some in particular deserve my deepest gratitude for their extraordinary patience and support: Shirley Stoneberger, Tom Thorton, Art Zygielbaum, Richard Moulder, John Gainsborough, Joseph Kahr and Bob Collinge.

My parents have my deepest appreciation for their moral support. The knowledge that they were always behind me was sometimes the only thing that kept me going.

Last, but certainly not least, I would also like to thank my wife Alice for her patience and understanding.

# Table of Contents

# List of Figures

# List of Tables

# Abstract

This thesis presents a study of load-balancing in a distributed system consisting of a number of heterogeneous hosts connected by a local area network. We motivate resource sharing by investigating imbalance and find that the probability of uneven loading is high for large systems under moderate load.

In order to be able to evaluate alternate load sharing schemes, an approximate queueing technique, the decoupled iterative technique, is proposed. This methodology determines the mean response time performance of large networks with heterogeneous interactive queues. To illustrate the effectiveness and efficiency of the technique, we include simple solutions for several complicated examples. A GPSS/PC simulation is used to validate our approximate technique. Another validation method used is comparison with known solutions to special cases.

We propose a new dynamic load-balancing scheme, called the enhanced receiver-initiated dynamic algorithm (ERIDA). The impact of the communication overhead is discussed. We also propose another effective scheme for load balancing, a two-mode dynamic algorithm (TMDA). The performance profile of this algorithm shows that this algorithm has better delay performance in the medium-to-high load range than either the sender-initiated scheme or the receiver initiated scheme. The iterative modelling technique and extensive simulation are applied to the two new load-balancing schemes to evaluate the mean response time performance.

# Chapter 1

# Introduction

## 1.1 Motivation

To centralize or to decentralize has been an issue of debate for many years. The arguments for centralizing are generally based on efficiency, since Grosh's law [13] states that the processing power of a computer is proportional to the square of the its cost. The traditional mainframe/supermini interactive environment concentrates on the capacity for sophisticated database management, powerful peripherals, and high-speed data computation. In contrast, the argument in favor of decentralization tends to focus on effectiveness; distributed systems improve computing service for users[20]. The greatest strength of distributed microcomputers is the immediacy of their interaction with the end user, which has led to rapid development of new software packages, new applications, attachment of new devices, and ease of use for the computer user.

The introduction of local area networks (LANs) and the rapid spread of

microcomputers throughout information systems have dramatically changed the computer development environment to one that is distributed. A major advantage of a distributed multi-microcomputer system is that most of the processing can be done locally. Each microcomputer, however, can still gain access to more powerful devices by using the LAN. These devices are too expensive or too infrequently used to justify being installed at each microcomputer station. For example, a robot-vision system might experience a sudden increase in requests to perform trajectory planning when an object on a collision course is detected. The overload condition could prevent the processor from responding in time to avoid the collision. Instead of maintaining a huge reserve capacity at each subsystem to ensure adequate performance under the highest anticipated load at that subsystem, dynamic load-sharing techniques can be employed. A temporary load at one region can be reduced by means of load balancing.

A distributed computer system consists of several computers with the same or different processing capabilities, connected together by a network. Figure 1.1 shows a typical example of a distributed heterogeneous computer system. This computer system consists of several CPUs with different capacities, along with various sets of peripheral devices: disks, printers, tape drives, and file servers. The CPU and peripherals are connected via a LAN so that they can share their resources. CPUs can even share the resources of another network via a bridge.

In such a distributed multicomputer system, there is a probability that

one of the computer stations will be idle while another computer station has more than one job waiting in the queue for service. We call this probability the imbalance probability.

## 1.2 Imbalance Probability for Heterogeneous Computer Systems

The imbalance probability for a homogeneous system was derived by Livny and Melman[28]. To extend this measurement to include the heterogeneous case, the imbalance probability, $IP$, can be evaluated by the following formula:

$$IP = \sum_{X \in \xi(N)} \prod_{i \in X} P_{i0} [ \prod_{i \in (N-X)} (1 - P_{i0}) - \prod_{i \in (N-X)} P_{i1} ] \qquad (1.1)$$

where $N = \{\text{all stations}\}$, $\xi(N) =$ set of all subsets of $N$, $P_{i0}$ is the probability that computer station $i$ is idle, and $P_{i1}$ is the probability that computer station $i$ has exactly one job.

The term $\prod_{i \in X} P_{i0}$ represents the probability that the subset $X$ of the total of $N$ computer stations in the system are idle. The first term in the bracket represents the probability that all stations other than those included in $X$ are busy. The second term in the bracket represents the probability that there is exactly one job in each of the remaining stations. Thus the term in the bracket shows the probability that at least one of the remaining stations has more than one job waiting for service. By summing over all subsets, the imbalance probability for the whole system is obtained.

Figure 1.1: A distributed heterogeneous computer system.

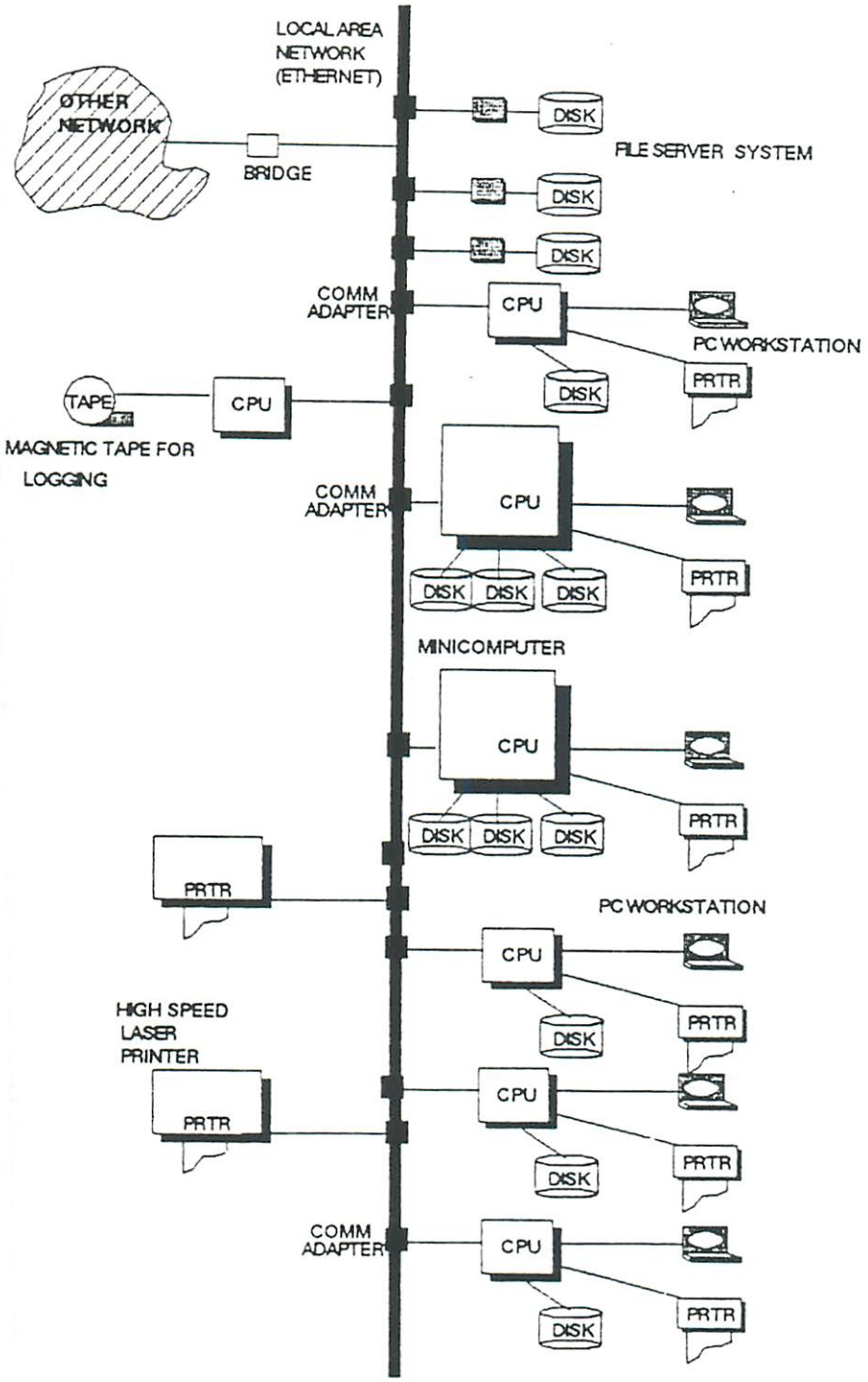
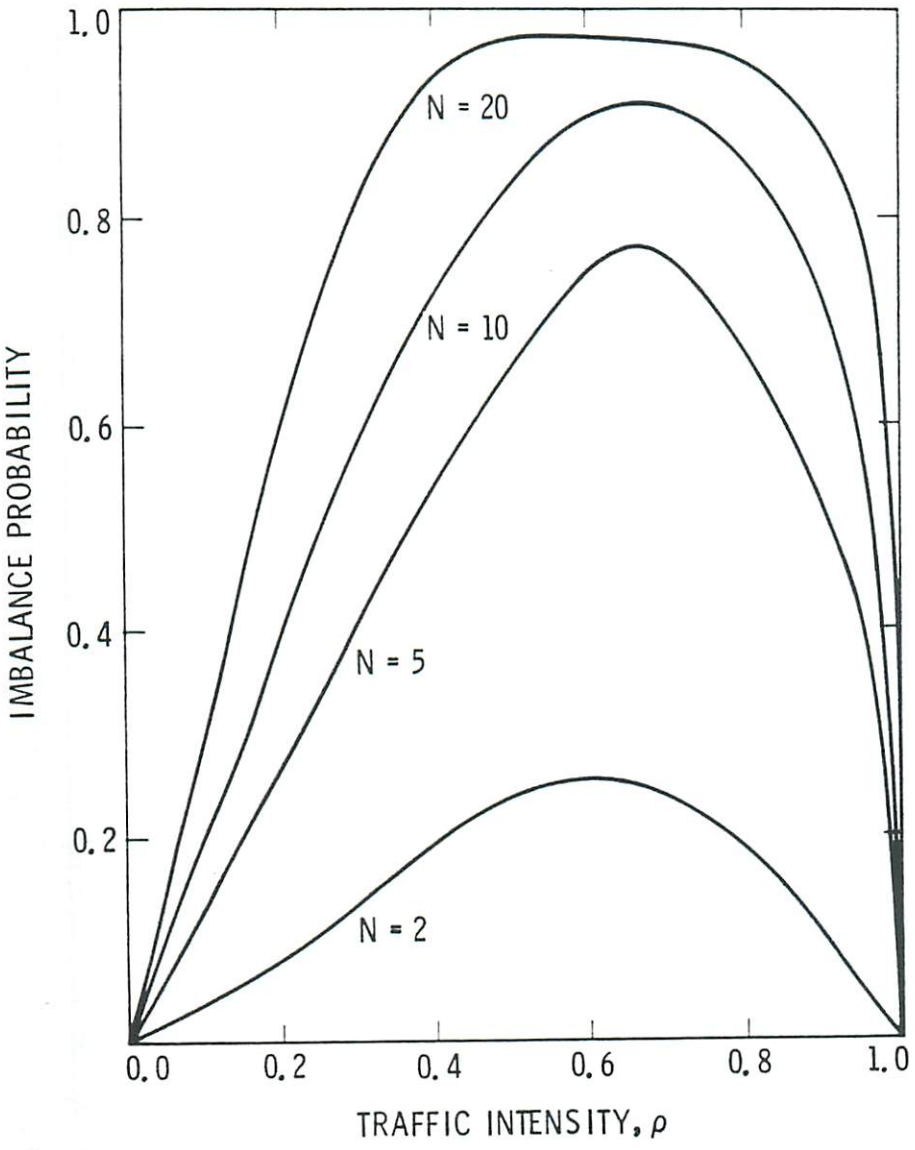


Figure 1.2: Imbalance as a function of utilization in a homogeneous system. N is number of servers in the system. Traffic intensity($\rho$)=arrival rate($\lambda$)/service rate($\mu$).



In Figure 1.2, a homogeneous system is considered. All the nodes have the same service rate and the same arrival rate. As the number of nodes increase, the peak of the imbalance probability goes higher. As the number of stations increases to 20, the imbalance probability approaches 1 when the traffic intensity (arrival rate divided by the service rate at each station) ranges between 40% and 80%. The curves also indicate that the probability of imbalance is high during moderate traffic intensity. This occurs because all nodes are lightly loaded or idle at low traffic intensity and all are busy at high traffic intensity.

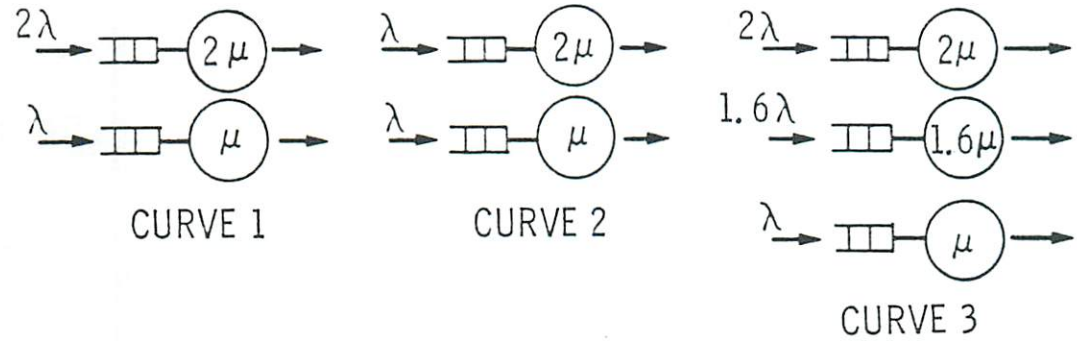
If the arrival rate is not evenly distributed, the imbalance probability becomes even higher. Curve 1 in Figure 1.3 shows the imbalance probability for a two-node heterogeneous system. In this configuration, the faster node is twice as fast as the slower one and the work is evenly distributed, i.e., $\lambda_{fast} = 2\lambda_{slow}$. If the utilizations are not balanced, e.g., $\lambda_{fast} = \lambda_{slow}$, curve 2 shows that the imbalance probability is even larger during high traffic loads. At this point, the slower node is heavily loaded even though the faster node is only 50% utilized. Curve 3 shows a three-node heterogeneous system, where the service rate ratio is 2:1.6:1 and the utilizations are balanced. We note that as was the case for a homogeneous system, the higher the number of nodes, the greater is the imbalance probability.

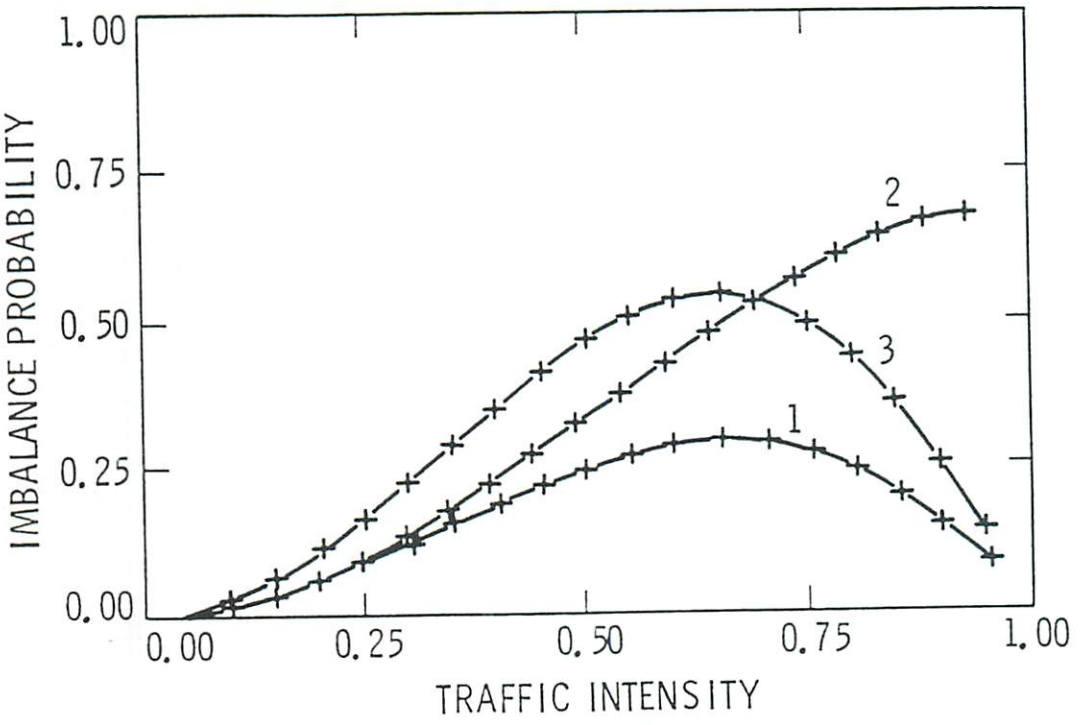
In summary, a high imbalance probability results when:

a. The number of stations is large.

b. The arrival rates are not evenly distributed.

Figure 1.3: Imbalance as a function of utilization for uniformly and nonuniformly distributed job arrival cases.



c. The traffic intensity is moderate.

A high imbalance probability implies a reduction in system performance. By reallocating queued jobs to the idle or lightly loaded stations, a reduction in system response time can be expected. This technique is called resource sharing [1] and is one of the main foci of this thesis.

The resource-sharing algorithm not only improves system performance, but also produces other advantages, including the following:

a. Increased reliability and availability:

   In the presence of hardware failures, the failed computer must be taken off line so that it can be repaired. Without resource sharing, all jobs associated with that computer would be delayed until the computer was placed back on line. With a resource-sharing scheme, such jobs can be transferred to other computers.

b. Improved maintainability:

   Maintenance commonly requires that a computer station be taken off line. With resource sharing, maintenance can be performed without interrupting normal service by transferring users' requests to other computers.

[1] The technique of balancing the workload over the available service stations is generally known as load sharing. Load sharing is usually looked at from the workload's point of view. The process of sharing the usable station service capability is generally known as resource sharing. Resource sharing is usually looked at from the service station's point of view. Loads can be shared only by a set of stations with similiar capabilities. Resources can be shared among a set of stations with different capabilities. Therefore, load sharing is a subset of resource sharing.

c. Provision of extra resources:

   Resource sharing can also provide alternate computing power. When a local host is overloaded, it may be possible to shift the job to another computer, thus eliminating the need for larger and more expensive computers. Resource sharing can also make unobtainable resources available to users who would not be able to afford the extra resources.

Given the importance of resource sharing, the main objectives of this study are to develop algorithms for efficiently distributing the workload to each resource in a given distributed heterogeneous computer system.

## 1.3 Thesis Outline

In the following chapter we review previous research on resource sharing as related to our present work. In Chapter 3, an approximate queueing technique, the decoupled iterative technique (DIT), is proposed to evaluate the mean response time performance of a large network with heterogeneous interactive queues. 'Normal' service centers in queueing network models are load independent: the server delivers 'service units' and the jobs arrive at a constant rate. However, for dynamically changing job arrival rates and various service rates depending on the populations of the remote stations, the situation becomes much more complex. For these complicated problems, this proposed queueing technique is found to be extremely useful. To illustrate the effectiveness

and efficiency of DIT, we include simple solutions for several complex examples in Chapter 3. A general-purpose simulation system/personal computer (GPSS/PC) simulation is used to validate our approximation technique. Another validation method used is comparison with known solutions to similar problems. In Chapter 4 we propose a new dynamic load-balancing scheme, called an enhanced receiver-initiated dynamic algorithm (ERIDA). Communication overhead is also discussed in this chapter. In Chapter 5 another effective scheme for load balancing, a two-mode dynamic algorithm (TMDA) is proposed. The performance profile of the algorithm shows that the TMDA has better delay performance in the medium-to-high load range than either the sender-initiated scheme or the receiver initiated scheme. We also develop a TMDA threshold selection strategy. DIT and extensive simulation are applied to both ERIDA and TMDA to evaluate the mean response time performance. Numerical results are compared with other known schemes in both Chapters 4 and 5. In all the load-balancing schemes, the potential wrong-transfer probability introduced by nonnegligible overhead was investigated. Some suggestions are made for reducing that potential problem. We conclude with suggestions for further research in Chapter 6.

# Chapter 2

# A taxonomy of resource sharing

Numerous studies have addressed the problem of resource sharing in distributed systems. It is convenient to classify these strategies as being either static or dynamic in nature and as having either a centralized or decentralized decision-making capability. We can further distinguish the algorithms by the type of node that takes the initiative in the resource sharing. Algorithms can be either sender initiated or receiver initiated. Some algorithms can be adapted to a generalized heterogeneous system, while others can be used only in a homogeneous system. These categories are further explained below.

Static/dynamic: Static schemes use only the information about the long-term average behavior of the system, i.e., they ignore the current state. Dynamic schemes differ from static schemes in that they determine how and when to transfer jobs based on the time-dependent current system state instead of the average behavior. The major drawback of static algorithms is that they do not respond to fluctuations in the workload. Dynamic schemes attempt to

correct this drawback but are more difficult to implement and may introduce additional overhead. In addition, dynamic schemes are difficult to analyze.

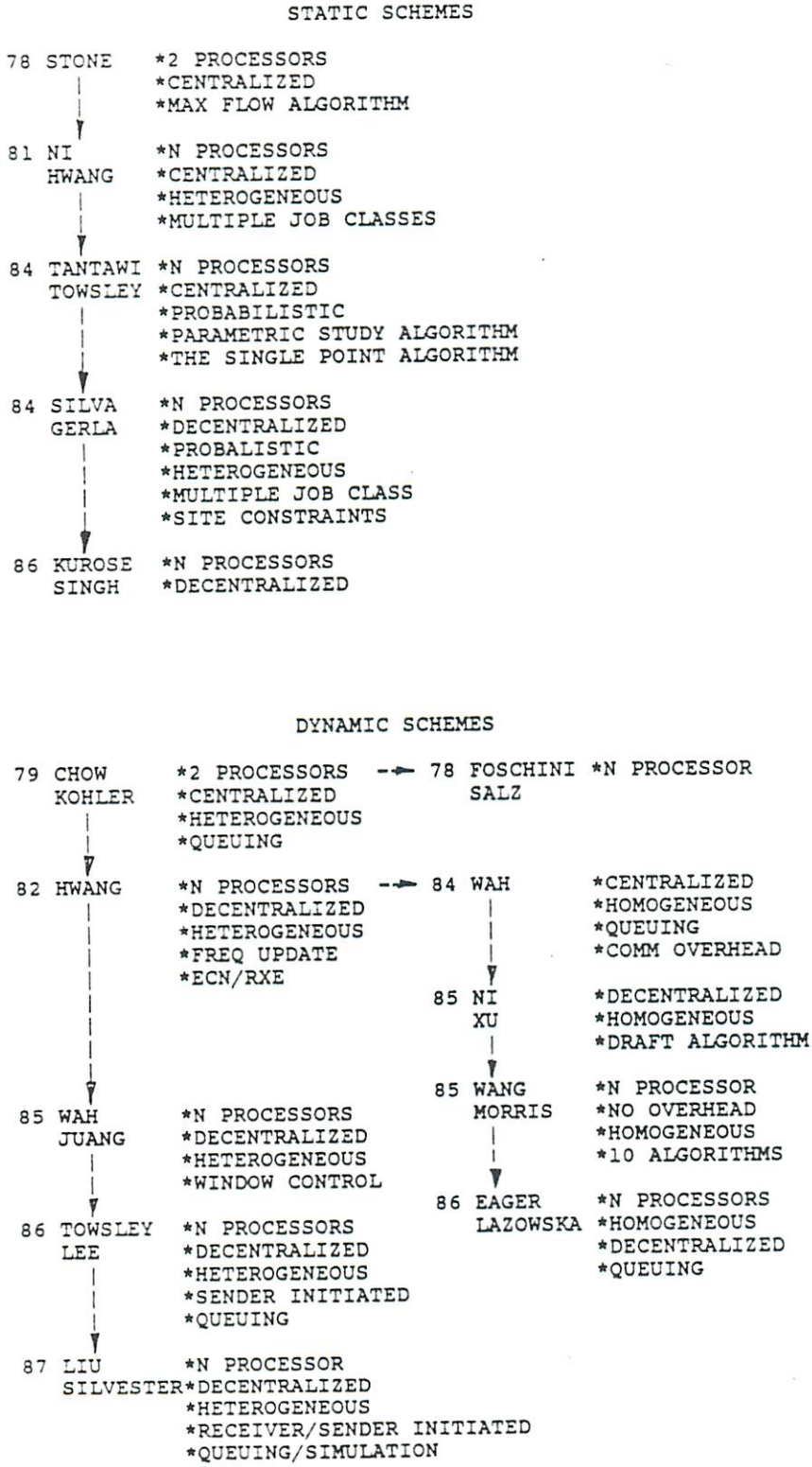
Centralized/decentralized: In a system with centralized control, jobs arrive at the central controller, which is responsible for distributing the jobs among the network stations. In a decentralized system, jobs are submitted to the individual stations and the decision to transfer a job to another station is made locally. The central dispatcher approach is quite restrictive for a distributed system model.

Homogeneous/heterogeneous: In a homogeneous system, all the computer stations are identical and have the same service rate. In a heterogeneous system, the computer stations do not have the same processing power.

Sender/receiver initiated: If the source node makes a determination as to where to route a job, this is defined as a sender-initiated strategy. In receiver-initiated strategies, the situation is reversed, i.e., lightly loaded stations search for congested stations from which work may be transferred.

The major studies are summarized in Table 2.1. First, we survey static strategies. Stone [39] developed a centralized maximum-flow algorithm for two processors by holding the load of one processor fixed and varying the load on the other processor. Ni and Hwang [14] studied the problem of load balancing in a multiple heterogeneous processor system with many job classes. In this system the number of processors was extended to more than two. Tantawi and Towsley [44] formulated the static resource-sharing problem as a nonlinear programming problem and presented two efficient algorithms, the

Table 2.1: A Taxonomy of Resource Sharing



parametric study algorithm and the single-point algorithm, which solved the centralized static load-balancing problem. Silva and Gerla [38] used a downhill queueing procedure to search for the static optimal job assignment in a heterogeneous system that supports multiple job classes and site constrains. Recently, Kurose and Singh [24] used an iterative algorithm to deal with the static decentralized load-sharing problem. Their algorithm was examined by theoretical and simulation techniques.

Next we examine the dynamic strategies. Chow and Kohler [4] used a queuing theory approach to examine a resource-sharing algorithm for a heterogeneous two-processor system with a central dispatcher. Their objective was to minimize the mean response time. Foschini and Salz [8] generalized one of the methods developed by Chow and Kohler to include multiple job dispatchers. Load balancing on the Purdue Engineering Computer Network (ECN) was implemented with a dynamic decentralized remote execution environment (RXE) program [15]. With the decentralized RXE, the load information of all the processors is maintained in each network machine's kernel. One of the problems with this approach is the potentially high cost of obtaining the required state information. It is also possible for an idle processor to acquire jobs from several processors and thus become overloaded. Wah [46] studied the communication overhead of a centralized resource-sharing scheme designed for a homogeneous system. Hwang and Liu [16] studied the performance of a centralized load-balancing algorithm for a homogeneous system. Ni and Xu [36] proposed the "draft" algorithm for a homogeneous system.

Wang and Morris [48] studied ten different algorithms for homogeneous systems to evaluate their performance differences. Eager et al.[6] evaluated the performance of sender-initiated and receiver-initiated load-sharing policies on homogeneous computer systems. They observed that the receiver-initiated approach is inferior at low loads and superior at high loads. They also addressed the problem of decentralized load sharing in a multiple-station system using dynamic state information [7]. Eager discussed the appropriate level of complexity for load-sharing policies and showed that schemes using relatively simple state information do very well and perform quite close to the optimal expected performance. The system configuration studied by Eager et al. is once again a homogeneous system. Wah and Juang [2] proposed a window control algorithm to schedule the resources in local computer systems with a multiaccess network. Towsley and Lee [41] used the threshold of the local job queue length at each host to make decisions for remote processing. This computer system was generalized to be a heterogeneous system.

In summary, most of the work has been limited to either static schemes, centralized control, homogeneous systems, or two-processor systems where overhead considerations were ignored. All of these approaches make assumptions that are too restrictive to apply to most real computer system installations. The main contribution of our work is the development of two dynamic decentralized resource-sharing algorithms for a heterogeneous multiple-processor system, along with an approximate queueing model technique for analysis, which can also be used to study other balancing algorithms. Both of

our algorithms are receiver initiated in heavy load, thus our approach differs significantly from the sender-initiated approach described in [41]. The advantage of our approach is that it does not impose extra overhead in the heavily loaded situation. Therefore, it will not bring the system to an unstable state.

# Chapter 3

# An approximate model for load-dependent queues

## 3.1 Introduction

A distributed heterogeneous computer system consists of several computers with different processing capabilities. Designing and constructing a distributed computer system is complex, costly, and time-consuming. Analytical modeling serves as an important technique to avoid costly mistakes early, as an aid in developing new algorithms, and as a predictor of overall performance of the various alternatives. In order to improve performance, a load-balancing scheme is needed to transfer load to other stations *dynamically*, depending on the current loading status of each station. In other words, the arrival rate (due to external and transfer arrivals) and the departure rate (due to service

completions and transfers) of each station are functions of the state of all stations in the system. For a heterogeneous computer system, the service rates are not all the same. The dynamic interactions among the queues in each station exhibit a complexity of behavior that can make an exact analytical solution unachievable.

## 3.2 Review of the State of the Art

Numerous studies have addressed this interactive-queue problem in distributed systems. Flatto and McKean[10] considered two homogeneous queues with dynamic routing and obtained an exact solution for the generating function. Baran and Dorsey[1] investigated adaptive control of two queues competing for the services of a single server. Lin and Kumar[26] considered optimal control of a queueing system which consists of a common queue served by two heterogeneous servers. Rubinovitch[37] studied a queue with two different exponential servers. It was assumed that the customers were allowed to stall, i.e., to wait for a busy fast server at times when the slow server was free. Knessel et al. [21] developed an asymptotic approach to obtain approximations to the steady-state joint distribution for the number of customers in a system of two heterogeneous queues where a newly arriving customer joins the shorter of the two queues. However, their results are all limited to two servers or two queues and cannot be extended to a larger number of stations.

Foschini [8], [9] used diffusion approximations to analyze networks with dynamic routing. Diffusion theory leads to a simplification of the problem of determining the equilibrium probabilities in that the partial difference equations are replaced by second-order partial differential equations. Unfortunately, in the absence of a heavy-traffic assumption, diffusion approximations generally yield poor results and do not agree with known exact results for even the simplest of queueing systems [17].

Goldberg and Popek [11] used mean value analysis approximation techniques for a multistation distributed computer system. However, their model cannot solve the *dynamic* queueing dispatching scheme where the behavior of the station is dependent on the *current* status of the system.

Zahorjan and Lazowska [50] used an approximate mean value analysis to solve the load-dependent server system. In their study, the service rate of each load-dependent server is a function only of the customer population at that local station and does not consider the loading of the remote stations.

In summary, most of the work, however, has been limited to a small number of stations or can be used only in a static environment. Exact analysis of the heterogeneous interactive queue problem is very complex, since we need an $N$-dimension Markov chain for a system with $N$ stations. The size of the state space grows rapidly with the number of queues. The major contribution of our study is an approximate model for this problem that has general application.

An approximate queueing technique, the decoupled iterative technique

(DIT), is proposed in this chapter to evaluate the mean response time performance of a large network with heterogeneous interactive queues. The job arrival and service rates depend on the state of the remote stations and can be dynamically changed. For these complex problems, this proposed queueing technique is found to be extremely useful. The technique has been used for packet radio network and has also been used in earlier studies [25] but only for very limited sets of parameters. We have generalized the approach to allow heterogeneous system to be studied. We even able to model overhead in some cases. To illustrate the effectiveness and efficiency of DIT, we include simple solutions for several complicated examples. A general-purpose simulation system/personal computer (GPSS/PC) simulation is used to validate our approximation technique. Another validation method used is comparison with known solutions to similar problems.

## 3.3 The DIT Approximate Queueing Model

The DIT approximate queueing model is described in this section. First, we define the following notation:

$N$=total number of homogeneous-station groups in the system.

$R$=queue length threshold for job transfer.

$P_m^i(l)$=$i^{th}$ iteration estimate for the steady-state probability that the queue length at station group $m$ is equal to $l$ ($m = 1, \ldots, N$ and $l = 1, \ldots, R$).

$q_m^i(j,k)$=$i^{th}$ iteration estimate for the transition rate from state $j$ to state $k$ at station $m$.

$P_m(l)$= $lim_{i\to\infty} P_m^i(l)$.

$\epsilon$=predetermined tolerance factor.

$\lambda_m$=external arrival rate of service group $m$.

$\vec{\lambda}$=$\{\lambda_1, \lambda_2, \ldots, \lambda_N\}$.

$\mu_m$=service rate of service group $m$.

$\vec{\mu}$=$\{\mu_1, \mu_2, \ldots, \mu_N\}$.

$\overline{N}_m$=average number of customers of homogeneous-station group $m$.

$T$=average system response time.

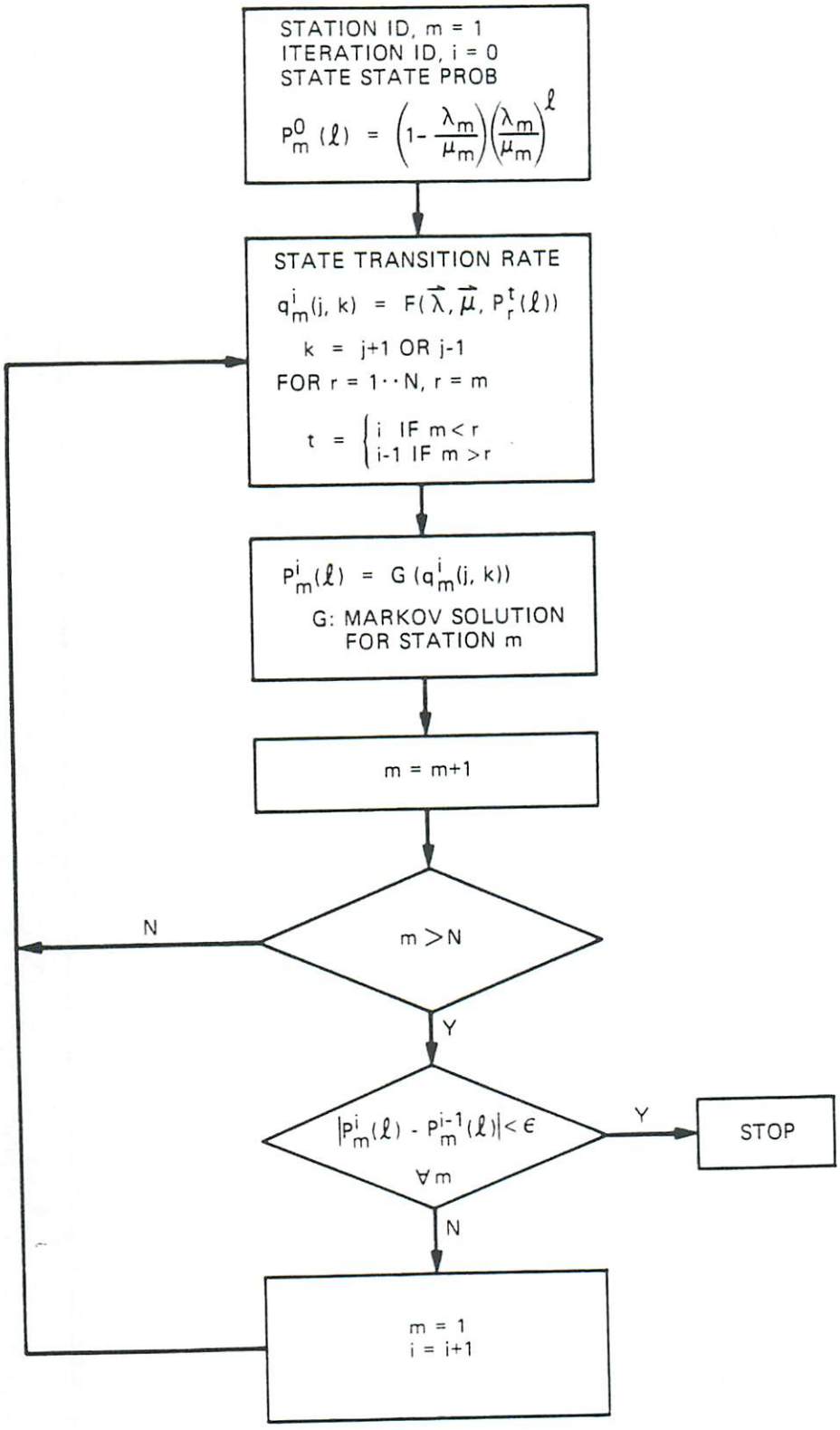
The premise for our proposed approximate method is the assumption that, for any station, the influence of other stations is adequately represented by their average state probability.

The structure of the DIT approximate queueing model is shown in Figure 3.1 and can be outlined as follows:

a. Decompose the multiple-server heterogeneous system into a collection of single-station systems. Categorize all the stations with identical arrival rates and identical service rates into homogeneous groups. Construct a Markov chain for each homogeneous group of stations $m$, $m = 1, \ldots, N$.

b. Pair up the Markov chains into all possible subsets of two groups. The state transitions for node $m$, $q_m^i(j,k)$, are a function of the local arrival

Figure 3.1: Flow chart of the DIT approximate queueing model.



and service rates $(\lambda_m, \mu_m)$ and depend on the arrival and service rates of all other stations in the same set. $q_m^i(j,k)$ is a function of $P_r^{i-1}(l)$, where $r = 1, \ldots, N$ and $r \neq m$.

c. Set $i = 0$; estimate initial steady state $P_m^0(l)$ by M/M/1 queue formula, i.e., $P_m^0(l) = (1 - \frac{\lambda_m}{\mu_m})(\frac{\lambda_m}{\mu_m})^l$.

d. Set $i = i + 1$; for $m = 1, \ldots, N$ do

   1. $q_m^i(j,k) = F(\vec{\lambda}, \vec{\mu}, P_r^t(l))$ where $j = 1, \ldots, R$ and $k = j + 1$ or $k = j - 1$; $t = i$ for $m < r$ and $t = i - 1$ for $m > r$; $r = 1, \ldots, N$ $r \neq m$; and $l = 1, \ldots, R$. The function $F$ depends on the specific load-sharing algorithm being modeled.

   2. For $l = 1, \ldots, R$, compute $P_m^i(l) = G(q_m^i(j,k))$ where $G$ is the Markov chain solution for the group $m$.

e. If $\mid P_m^i(l) - P_m^{i-1}(l) \mid < \epsilon$ for $m = 1, \ldots, N$ and $l = 1, \ldots, R$, stop; else go to step d.

There is an potential source of inaccuracy in this solution; it is the very premise of the method. This scheme replaces the average of a function of a random variable with the function of the average of that variable. In practice, however, the accuracy of this scheme is reasonable, as illustrated in the following section.

## 3.4 Examples: Some Interactive Queue Problems

In this section we describe the application of DIT to several typical state-dependent interactive queue problems. These problems are solved by our approximate queueing technique, DIT, presented in the previous section. The results are validated either by the known exact solution or by a GPSS/PC simulation exercise.

### 3.4.1 The Basic Sender-Initiated Load-Sharing Problem

**Problem Description** In a multiple-station computer system, reallocating queued jobs to idle or lightly loaded stations by a load-sharing scheme will reduce system response time. The basic sender-initiated load-sharing policy proposed by Eager et al. [6] is a typical example of interactive queue problem.

The sender-initiated load-sharing policy is described as follows:

A task is transferred from its originating node to another node if and only if the number of tasks already in service or waiting for service is greater than or equal to some threshold value.

**Model Construction** In this section we show how the DIT approximate analytical model can be used to find the average response time of a distributed system with the sender-initiated load-balancing scheme in a heterogeneous environment.

Let us review the main assumptions:

a. The threshold is 1. i.e., a station tries to give away a job whenever there is more than one job in the queue at that station.

b. There is only one class of tasks. New tasks arrive at node $i$ at a rate $\lambda_i$, and the service rate at node $i$ is $\mu_i$. Both the arrival and service times are exponentially distributed.

c. The network transmission delay in propagating requests, probing the status, and returning results is negligible. This assumption is valid if the transmission bandwidth is large compared to the traffic on the network.

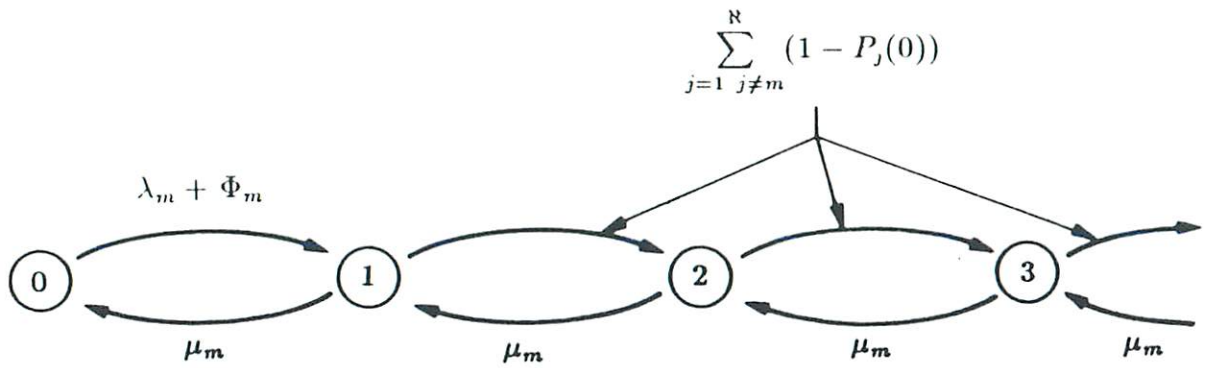
d. All processing overhead for probing, packing and unpacking data, request, and result transfer is ignored. This assumption is valid if the time required to pack and unpack the job is significantly less than the time required to process the job.

An exact analytical solution of this load-sharing algorithm with multiple-interaction queues is quite complex, since an $N$-dimensional Markov chain is needed to show the interactions among the queues. An exact solution for such a system with more than two queues becomes intractable. Instead of describing the number of jobs at all the stations in one state representation, our approximate model concentrates on the behavior of one queue at a time. It attempts to express the interaction relations in the state transition rates of the Markov chain of the queue in each station. This approach characterizes the iteration between the various queues in terms of their steady-state occupancy probabilities. Iteration is then used to update estimates of the interaction.

The Markov chain for the sender-initiated load-sharing scheme is shown in Figure 3.2. Transitions from state $(k+1)$ to state $k$ correspond only to the internal service rate, $\mu_m$.

Figure 3.2: Markov chain for the basic sender-initiated load-sharing scheme.

$\theta_m = 1$



For state $k(\geq 1)$, upward state transitions occur only if none of the remote stations is idle. This upward transition rate, $\delta_m(k)\lambda_m$, where $\prod_{j=1\ j\neq m}^{N}(1 - P_j(0)) \leq \delta_m(k) \leq 1$. The leftmost term of this inequality is the probability that all stations are not idle and assumes independent. In reality, when the local station $m$ is busy, the remote stations $j$ will be more likely to be busy, i.e., P[remote station $j$ busy | local station $m$ busy] $\geq$ P[remote station $j$ busy]. In the DIT model, $\delta_m(k)$ is approximated as $\lambda_m \prod_{j=1\ j\neq m}^{\aleph}(1 - P_j(0))$, where $1 < \aleph < N - 1$.

Upward transitions from state 0 to state 1 in the Markov chain occur due to:

a. Arrivals at a rate $\lambda_m$.

b. The transfer rate from the remote heavily loaded stations. This transfer rate, $\Phi_m$, can be estimated from the system flow balance equation:

$$\sum_{m=1}^{N} \lambda_m \Phi_m P_m(0) = \sum_{m=1}^{N} \lambda_m (1 - P_m(0))(1 - \prod_{j=1 j\neq m}^{\aleph} (1 - P_j(0))) \quad (3.1)$$

Combining the above two factors, the upward transition rate is

$$\lambda_m + \Phi_m \quad (3.2)$$

Given this model, we can immediately proceed to the iteration described in the previous section.

Let

$$\alpha_m = \frac{\lambda_m + \lambda_m \Phi_m}{\mu_m} \quad (3.3)$$

$$\beta_m = \frac{\delta_m \lambda_m}{\mu_m} \quad (3.4)$$

$$P_m(1) = \alpha_m P_m(0) \tag{3.5}$$

$$\text{and } P_m(n) = \alpha_m \beta_m^{n-1} P_m(0) \text{ for } n \geq 2 \text{ and } m \in N \tag{3.6}$$

Since $\beta_m < 1$,

$$\sum_{n=0}^{\infty} P_m(n) = 1 \tag{3.7}$$

$$= P_m(0) + \alpha_m \frac{1}{1-\beta_m} P_m(0) \tag{3.8}$$

$$P_m(0) = \frac{1}{1 + \frac{\alpha_m}{1-\beta_m}} \tag{3.9}$$

Finally, the average number of customers and the mean system response time are

$$\overline{N_m} = \alpha_m \sum_{n=1}^{\infty} n \beta_m^{n-1} P_m(0)$$

$$= \frac{\alpha_m}{(1-\beta_m)^2} P_m(0) \tag{3.10}$$

$$\overline{T_m} = \frac{\overline{N_m}}{\lambda_m} \tag{3.11}$$

and hence the average time in system for the whole system can be obtained by Little's formula [27]:

$$T = \frac{\sum_{m=1}^{N} \overline{N}_m}{\sum_{m=1}^{N} \lambda_m} \tag{3.12}$$

We see that the transition rates, and hence the steady-state solution, for node $m$ depend on the steady-state probabilities of the other queues in the system. The iteration procedure shown in Figure 3.1 is used to generate $P_m(0)$

and $P_m(1)$ for each station. The average number of customers at each station and the average system response time can be obtained by means of Equations 3.10 and 3.12. It has been the authors' experience that this scheme converges within a small number of iterations. In the examples considered, the maximum number of iterations was five for $\epsilon = 10^{-3}$.

**Validation Studies.**

Ten-Homogeneous Stations

The approximate queueing model is validated by simulation. A system with ten homogeneous stations is analyzed. The results from the average response time of the sender-initiated resource-sharing scheme generated by the approximate queueing model and the results from the GPSS/PC simulation exercise are compared in Figure 3.3. The simulation result is close to the DIT approximation with $\aleph = 4$.

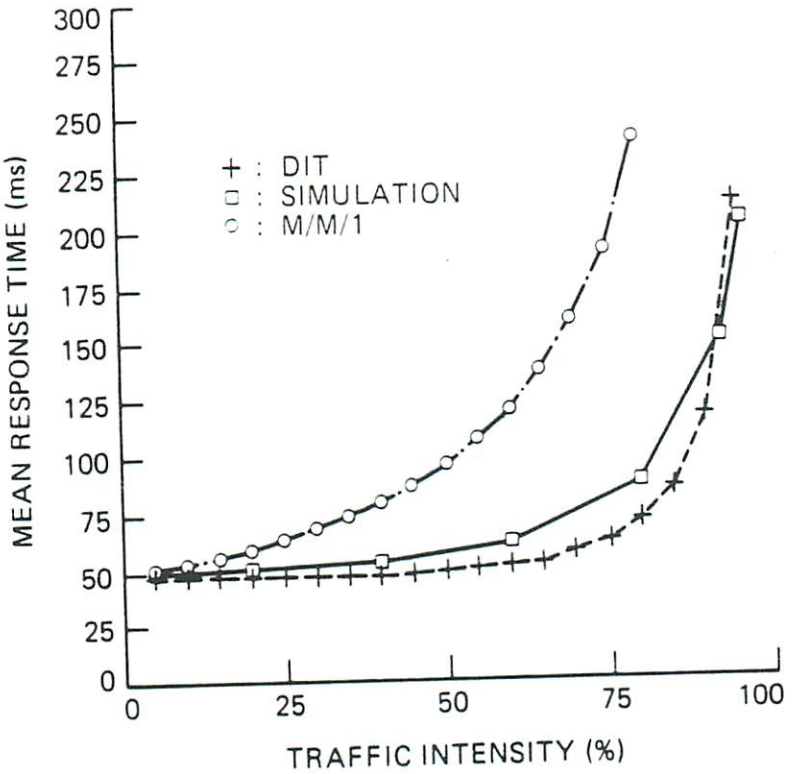
Thirty-Heterogeneous Stations

We also apply DIT to a highly heterogeneous system with various speed groups. Each group consists of several stations. The speed ratios among the different groups are different. All thirty stations have the same arrival rate.

For clarity, the DIT model will be re-illustrated here. Categorize all the stations with identical service rate and arrival rate in one group. Label the group ID to be $1, 2, \cdots, N$ according to the desending order of service rate.

Construct a Markov chain pair for each two of service groups. The steady-state probability for group $m$ with queue length $l$, $P_m^i(l)$, can be obtained by solving those Markov chain pairs iteratively. For the sender-initiated case, the downward transition rate is always $\mu_m$. The upward transition rates of Markov chain are summarized in the following table:

Figure 3.3: Average response time comparison between DIT approximation and GPSS/PC simulation for the basic sender-initiated load-sharing scheme.



| Group Pair | Group | Upward Transition Rate $P_m(0) \rightarrow P_m(1)$ | Upward Transition Rate $P_m(k) \rightarrow P_m(k+1)\ k \geq 1$ |
|---|---|---|---|
| 1:m | 1 | $\lambda_1 + \lambda_1 \Phi_{1:m}$ | $\lambda_1 \delta_1$ |
| | m | $\lambda_m$ | $\lambda_m \delta_1$ |
| 2:m | 2 | $\lambda_2 + \lambda_2 \Phi_{2:m}$ | $\lambda_2 \delta_2$ |
| | m | $\lambda_m$ | $\lambda_m \Pi_{l=1}^{2} \delta_l$ |
| 3:m | 3 | $\lambda_3 + \lambda_3 \Phi_{3:m}$ | $\lambda_3 \delta_3$ |
| | m | $\lambda_m$ | $\lambda_m \Pi_{l=1}^{3} \delta_l$ |
| ... | ... | ... | ... |
| m-1:m | m-1 | $\lambda_{m-1} + \lambda_{m-1} \Phi_{m-1:m}$ | $\lambda_{m-1} \delta_{m-1}$ |
| | m | $\lambda_m$ | $\lambda_m \Pi_{l=1}^{m-1} \delta_l$ |
| 1:m-1 | 1 | $\lambda_1 + \sum_{l=1:m,1:m-1} \Phi_l$ | $\lambda_1 \delta_1^2$ |
| | m-1 | $\lambda_{m-1}$ | $\lambda_{m-1} \delta_1 \delta_{m-1}$ |
| 2:m-1 | 1 | $\lambda_2 + \sum_{l=2:m,2:m-1} \Phi_l$ | $\lambda_2 \delta_2^2$ |
| | m-1 | $\lambda_{m-1}$ | $\lambda_{m-1} \delta_2 \delta_{m-1}$ |
| ... | ... | ... | ... |
| 1:1 | 1 | $\lambda_1 + \lambda_1 \sum_{l=1:m,1:1} \Phi_l$ | $\lambda_1 \Pi_{l=1}^{m-1} \delta_l$ |
| ... | ... | ... | ... |
| m:m | m | $\lambda_m + \lambda_m \Phi_{m:m}$ | $\lambda_m \Pi_{l=1}^{m} \delta_l$ |

For service groups $i$ and $j$, with $i < j$,

$$\begin{aligned} \sum_{l=i_1 \ldots i_{n_i}} \lambda_l \Phi_{i:j} P_l(0) &= \sum_{l=i_1 \cdots i_{n_i}} \lambda_l \delta_i^{N-j} (1 - P_l(0))(1 - \delta_i) \\ &+ \sum_{l=j_1 \cdots j_{n_j}} \lambda_l \delta_j^{N-j} \Pi_{l=1}^{i-1} \delta_l (1 - P_l(0))(1 - \delta_i) \end{aligned} \tag{3.13}$$

$$\delta_i = \Pi_{l=i_1}^{\aleph}(1 - P_l(0)) \quad \text{where } \aleph = 4 \tag{3.14}$$

We solve these Markov chains as long as the updated service utilization of group $i$ is less than group $j$ for $i < j$, i.e, $\rho_{i,updated} < \rho_{j,updated}$, where $\rho_{i,updated} = \frac{\lambda_{upward\ transition\ rate, P_m(0) \to P_m(1)}}{\mu_i}$.

We consider a heterogeneous system with three speed groups. Each group consists of ten stations. The speed ratio among the three groups is either 5:3:1 or 3:2:1. All 30 stations have the same arrival rate. The DIT performance predictions of the two speed ratios are compared with simulation results in Figures 3.4 and 3.5. The comparisons show a fairly good match except for very high load range, i.e., $\rho > 90\%$.

### 3.4.2 The Basic Receiver Initiated Load Sharing Problem

#### Problem Description

The receiver-initiated load-sharing policy proposed by Eager et al. [6] is another typical example of an interactive queueing problem.

The basic receiver-initiated load-sharing policy is described as follows:

A station attempts to replace a job that has completed processing if there are less than a certain threshold number of jobs remaining at the station. A remote station is selected at random and probed to determine whether the transfer of a job from that station would place its queue length below the threshold. If not, a job is transferred to the station initiating the probe. Otherwise, another station is selected at random and probed in the same manner.

Figure 3.4: Average response time comparison between DIT approximation and GPSS/PC simulation for the basic sender-initiated load-sharing scheme in a highly heterogeneous system with three speed groups. Each group contains ten stations. The speed ratio among the three groups is 5:3:1. All 30 stations have the same arrival rate.

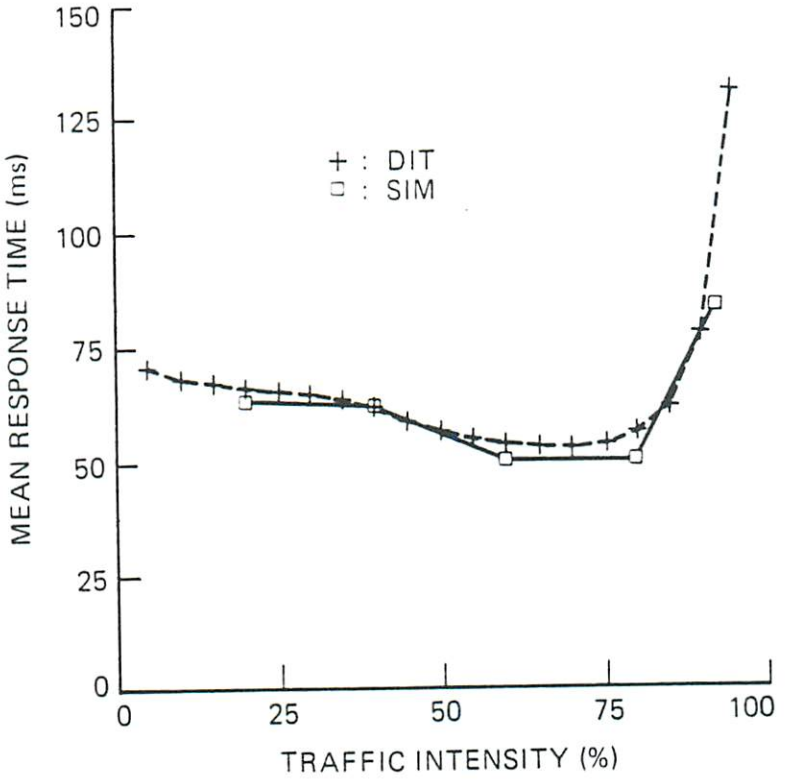
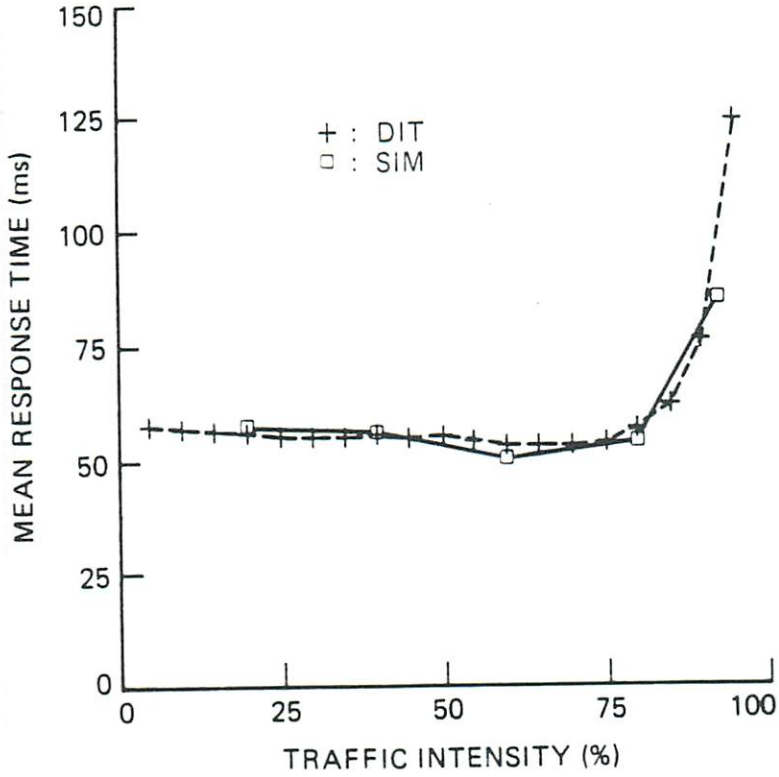


Figure 3.5: Average response time comparison between DIT approximation and GPSS/PC simulation for the basic sender-initiated load-sharing scheme in a highly heterogeneous system with three speed groups. Each group contains ten stations. The speed ratio among the three groups is 3:2:1. All 30 stations have the same arrival rate.



### Model Construction

Again, we show how the DIT approximate analytical model can be used to find the average response time with the receiver-initiated load-balancing scheme in a heterogeneous environment.

The following are the main assumptions:

a. The threshold is 0. In other words, a station tries to pull a job from other stations whenever it is idle.

b. There is only one class of tasks. New tasks arrive at node $i$ at a rate $\lambda_i$, and the service rate at node $i$ is $\mu_i$. Both the arrival and service times are exponentially distributed.

c. The network transmission delay in propagating requests, probing the status, and returning results is negligible. This assumption is valid if the transmission bandwidth is large compared to the traffic on the network.

d. All processing overhead for probing, packing and unpacking data, request, and result transfer is ignored. This assumption is valid if the processing required to pack and unpack the job is significantly less than the processing required to process the job.

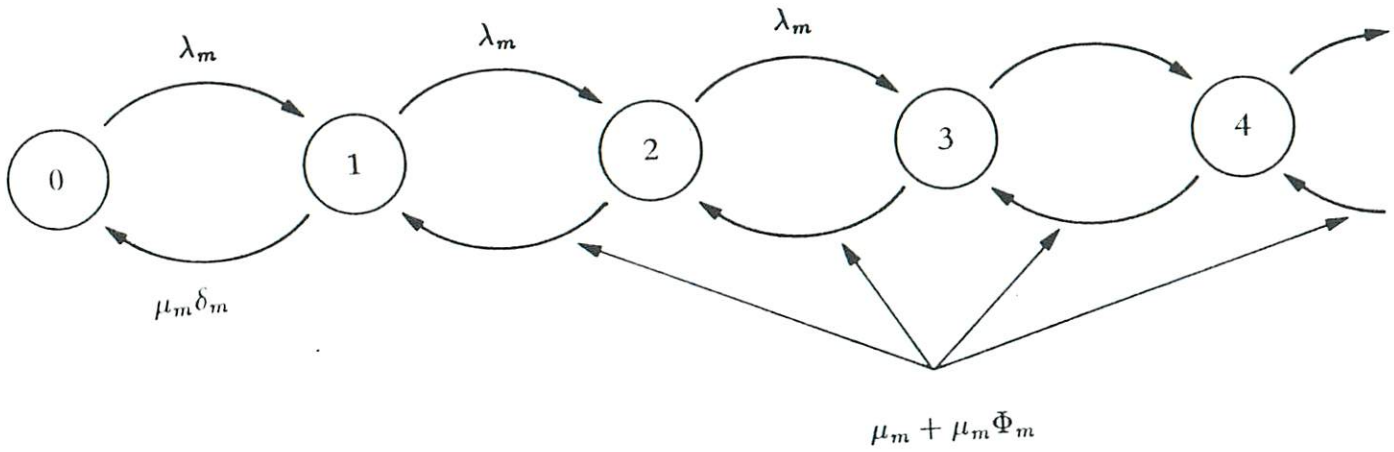
The Markov chain for the receiver-initiated load-sharing scheme is shown in Figure 3.6. Transitions from state $k$ to state $(k+1)$ correspond only to external arrivals, $\lambda_m$, for $k \geq 0$.

State 1 transfers back to idle state only if none of the remote stations is heavily loaded. The transition rate, $\delta_m \mu_m$, is

$$\mu_m \prod_{j=1 \; j \neq m}^{\aleph} (P_j(0) + P_j(1)) \quad \text{where} \quad 1 \leq \aleph \leq N \tag{3.15}$$

When $\aleph$ out of $N$ remote stations are in states $P(0)$ or $P(1)$ and the local station $m$ is in state $P(1)$, the other $N - 1 - \aleph$ remote stations will be most

Figure 3.6: Markov chain for the basic receiver-initiated load-sharing scheme.



likely to be in state $P(0)$ and $P(1)$, since the load of each station is balanced through the load-balancing scheme. In other words, P[all the $N-1$ remote stations are in state $P(0)$ or $P(1)$ | $\aleph$ remote stations are in state $P(0)$ or $P(1)$ and local station $m$ is in state $P(1)$] $\approx 1$. In our study, $\aleph = 4$ is a good approximation for a 10-station receiver-initiated system. By using $\aleph = 9$, DIT will generate a low response time prediction at high load range. This is because the downward transition rate $\mu_m \prod_{j=1\ j\neq m}^{\aleph}(P_j(0)+P_j(1))$ is much less than $\mu_m \prod_{j=1\ j\neq m}^{4}(P_j(0)+P_j(1))$ and does not reflect the real situation.

Downward transitions in the Markov chain for $k \geq 1$ occur due to:

a. Service completions at a rate $\mu_m$.

b. Load sharing. When the queue length of station $m$ is 2 or more, it can be reduced through load sharing. The reduced rate, $\Phi_m\mu_m$, can be obtained from the system flow balance equation:

$$\sum_{m=1}^{N} \Phi_m\mu_m(1-P_m(0)-P_m(1)) = \sum_{m=1}^{N} P_m(1)\mu_m\delta_m \tag{3.16}$$

The left side of the equation represents the total extra service obtained for all stations with a queue length of 2 or more. The right side of the equation represents the total service provided by all the lightly loaded stations.

The load sharing rate, $\Phi_m\mu_m$, can be approximated as

$$\mu_m\Phi_m = \mu_m \frac{\sum_{m=1}^{N} P_m(1)\mu_m(1-\delta_m)}{\sum_{m=1}^{N}(1-P_m(0)-P_m(1))} \tag{3.17}$$

Combining the above two factors, the downward transition rate is

$$\mu_m + \mu_m\Phi_m \tag{3.18}$$

Given this model, we can immediately proceed to the iteration described in Section 3.3. For clarity, we give here the solution to the birth/death equations for station $m$:

$$P_m(n) = \alpha_m \beta_m^{n-1} P_m(0) \quad \text{for } n \geq 1 \tag{3.19}$$

$$\text{where} \quad \alpha_m = \frac{\lambda_m}{\mu_m \prod_{j=1 \; j \neq m}^{K} (P_j(0) + P_j(1))} \tag{3.20}$$

$$\beta_m = \frac{\lambda_m}{\mu_m + \mu_m \delta_m} \tag{3.21}$$

By knowing that

$$\sum_{n=0}^{\infty} P_m(n) = 1 \tag{3.22}$$

we have

$$P_m(0) = \frac{1}{1 + \frac{\alpha_m}{1 - \beta_m}} \tag{3.23}$$

and the average number of customers at station $m$ is

$$\overline{N}_m = \alpha_m \sum_{n=1}^{\infty} n \beta_m^{n-1} P_m(0) \quad \text{for } n \geq 1 \tag{3.24}$$

$$= \frac{\alpha_m}{(1 - \beta_m)^2} P_m(0) \tag{3.25}$$

$$= \frac{\alpha_m}{(1 - \beta_m)(1 + \alpha_m - \beta_m)} \tag{3.26}$$

### Validation Studies

In this section we report on validation studies of the approximate queueing model using simulation. The simulation models were produced using a discrete-event GPSS/PC simulation program on the IBM AT [19]. Each simulation point corresponds to a total of 100,000 job completions. A system

with ten homogeneous stations is analyzed. The results from the average response time of a receiver-initiated resource-sharing scheme generated by the approximate queueing model and the results from the GPSS/PC simulation exercise are compared in Figure 3.7. This figure shows that our approximate queueing model is very close to the simulation result.

### 3.4.3 The M/M/N Problem

#### Problem Description

The heterogeneous M/M/N is an another typical example of an interactive queue problem. In this section our DIT approximate queueing model is applied to solve that problem.

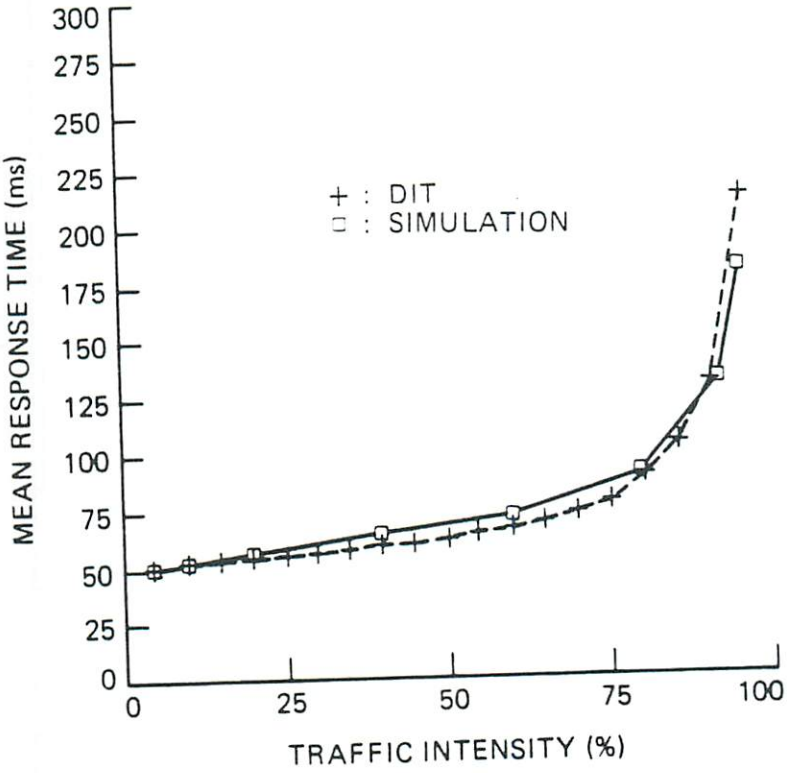
#### Model Construction

The state transition diagram of a centralized heterogeneous M/M/N system is shown in Figure 3.8. The Markov chain for each station consists of two states: idle (state 0) and busy (state 1). Jobs wait in a central queue in the order of their arrival. The central queue is represented by station 0. Jobs will be dispatched to idle servers as soon as an idle station is available. No station will take more than one job at a time. Hence, the transition rate from idle to busy for station $m$ is the total system arrival rate, $\lambda$, shared by station $m$ and the number of the other idle stations, i.e., $\frac{\lambda}{1+\sum_{j=1 j\neq m}^{N} P_j(0)}$. Station $m$ transfers from busy state to idle state when he completes a service (rate $\mu_m$) and there is no job in the central queue. Hence, the departure rate is $\mu_m P_0(0)$.

The birth/death equation for station $m$ is

$$\frac{\lambda}{1+\sum_{j=1 j\neq m}^{N} P_j(0)} P_m(0) = \mu_m P_0(0) P_m(1) \qquad (3.27)$$

Figure 3.7: Average response time comparison between DIT approximation and GPSS/PC simulation for the basic receiver-initiated load-sharing scheme.



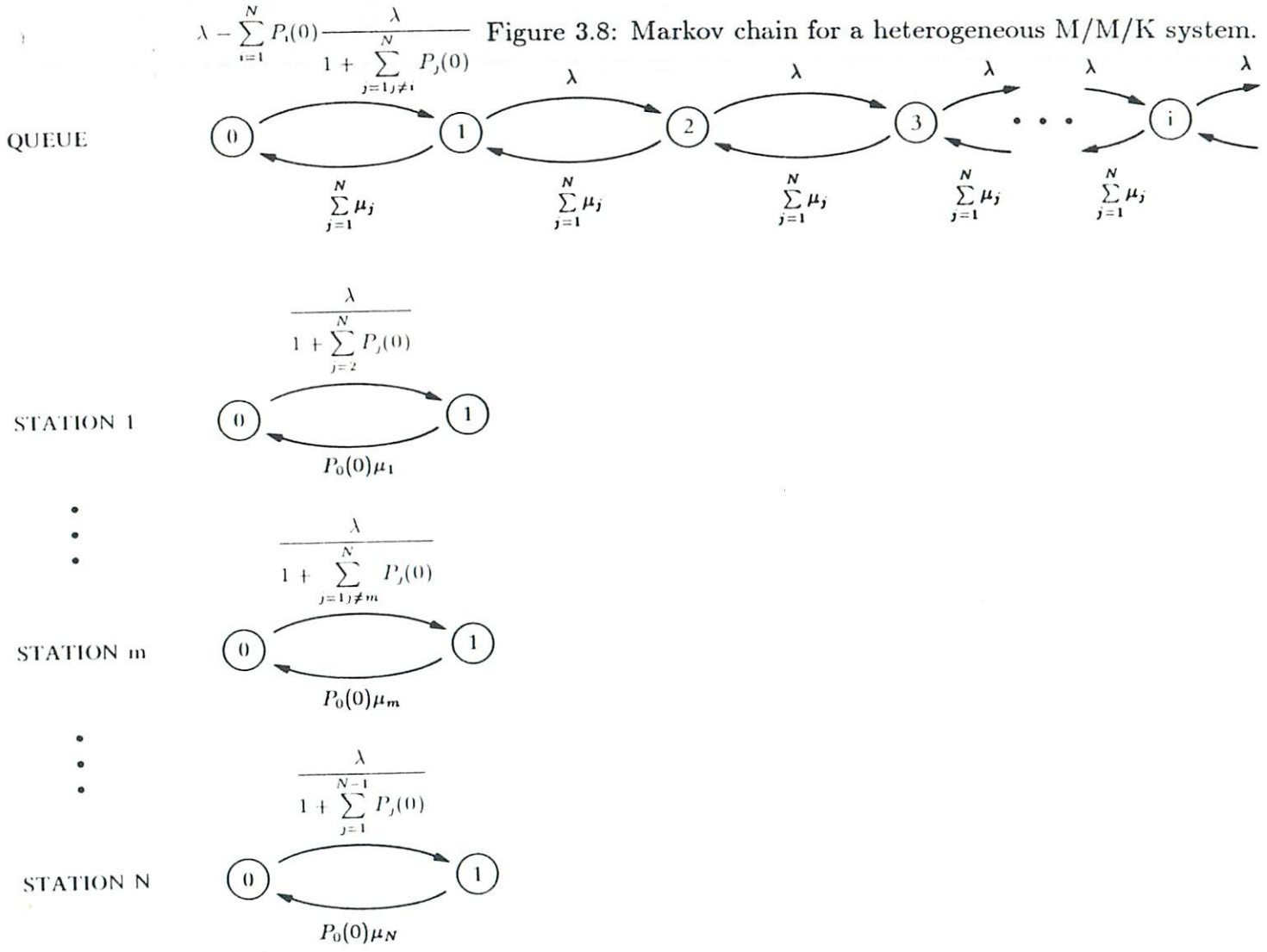



Figure 3.8: Markov chain for a heterogeneous M/M/K system.

$$\text{Knowing} \qquad P_m(0) + P_m(1) = 1 \tag{3.28}$$

$$P_m(1) = \frac{\dfrac{\lambda}{1 + \sum_{j=1 j\neq m}^{N} P_j(0)}}{P_0(0)\mu_m + \dfrac{\lambda}{1 + \sum_{j=1\ j\neq m}^{N} P_j(0)}} \tag{3.29}$$

The birth/death equation for the central queue is

$$P_0(n) \;=\; \beta^{n-1}\alpha P_0(0) \qquad \text{for} \quad n \geq 1 \tag{3.30}$$

$$\text{where} \quad \alpha \;=\; \frac{\lambda - \sum_{i=1}^{N} P_m(0) \dfrac{\lambda}{1 + \sum_{j=1 j\neq m}^{N} P_j(0)}}{\sum_{j=1}^{N} \mu_j} \tag{3.31}$$

$$\beta \;=\; \frac{\lambda}{\sum_{j=1}^{N} \mu_j} \tag{3.32}$$

The summation of all probabilities equals one:

$$P_0(0) + \alpha \sum_{n=1}^{\infty} \beta^{n-1} P_0(0) \;=\; 1 \tag{3.33}$$

$$\text{Therefore,} \quad P_0(0) = \frac{1}{1 + \dfrac{\alpha}{1-\beta}} \tag{3.34}$$

By applying the DIT, the steady-state probabilities for $P_i(j)$ can be obtained.

The average number of customers in the whole system is

$$\overline{N_m} \;=\; \sum_{i=1}^{\infty} i\alpha\beta^{i-1} P_0(0) + \sum_{j=1}^{N} P_j(1) \tag{3.35}$$

$$= \frac{\alpha P_0(0)}{(1-\beta)^2} + \sum_{j=1}^{N} P_j(1) \tag{3.36}$$

**Validation Studies**

To compare our approximation result with the known homogeneous M/M/N system solution, we assume the service rates of each station to be the identical. The exact solution of mean response time for homogeneous M/M/N can be represented by Erlang's delay-call formula [12]:

$$T = \frac{1}{\mu} + [\frac{(\frac{\lambda}{\mu})^N \mu}{(N-1)!(N\mu-\lambda)^2}][\sum_{j=0}^{N-1} \frac{1}{j!}(\frac{\lambda}{\mu})^j + \frac{1}{N!}(\frac{\lambda}{\mu})^N \frac{N\mu}{N\mu-\lambda}]^{-1} \tag{3.37}$$

The response time comparison of an M/M/2, M/M/10, and M/M/20 systems between the DIT and the exact solution are shown in Figure 3.9, Figure 3.10, and Figure 3.11. Again, the result shows that our approximate DIT gives a fairly accurate result. The result also indicates that the more number of the stations, the more accurate DIT prediction is.

### 3.4.4 Queues with Ordered Heterogeneous Servers

With minor modification, the M/M/N DIT model discribed in the previous section can also be applied in the queueing system with ordered servers that work at different rates. The exact solution for the system with ordered heterogeneous servers was obtained by Cooper [5]. The queueing system in his analysis can be described as follows: There are $n$ servers numbered $1, 2, ..., n$ which are accessed by the incoming jobs in a fixed order. Specifically, an arriving job is processed by the lowest-numbered idle server, if such a server exists. When all servers are busy, a single queue is formed and jobs enter for service on a first-come first-served basis.

Figure 3.9: Average response time comparison between DIT approximation and exact result for the M/M/2 system.

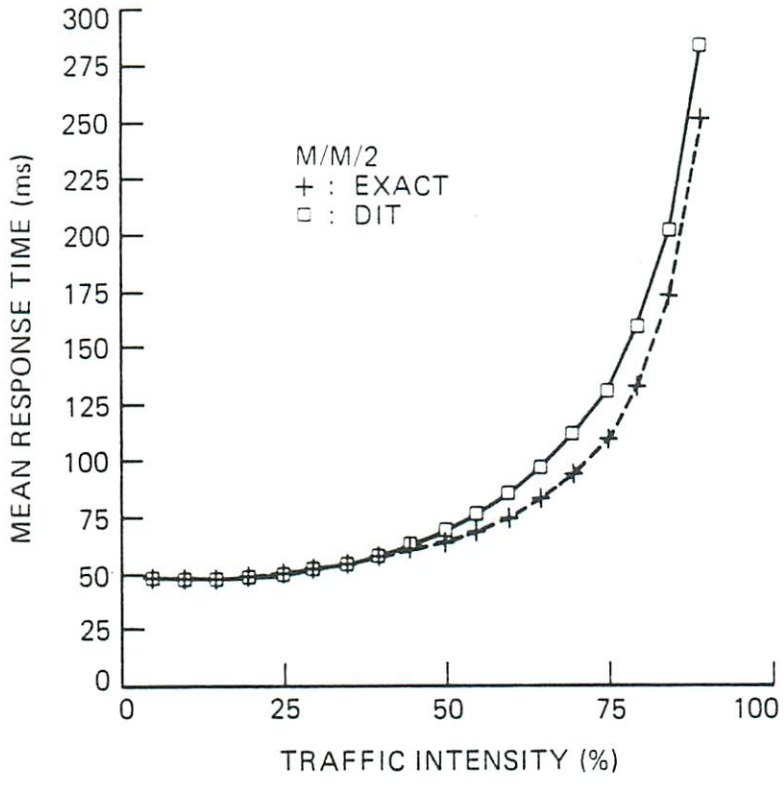


Figure 3.10: Average response time comparison between DIT approximation and exact result for the M/M/10 system.

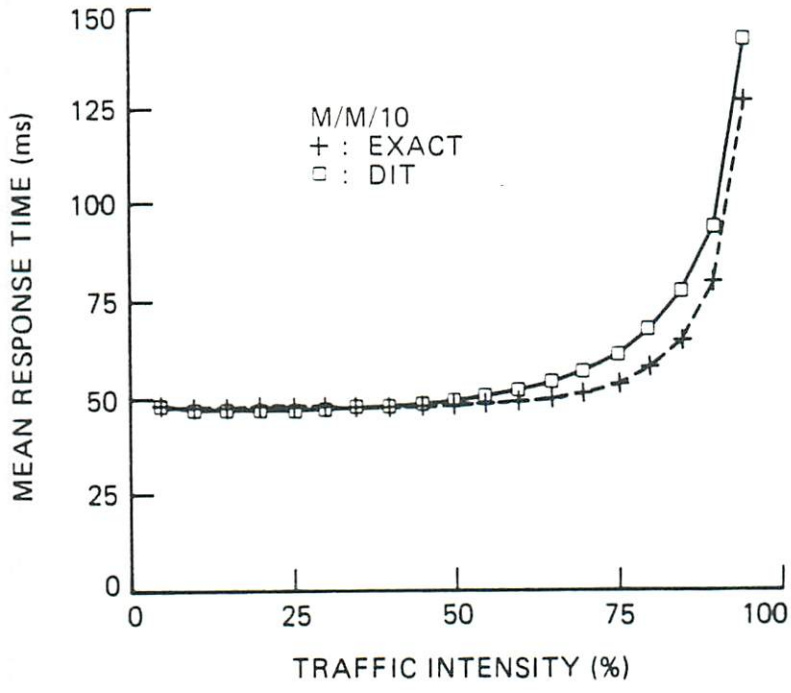


Figure 3.11: Average response time comparison between DIT approximation and exact result for the M/M/20 system.

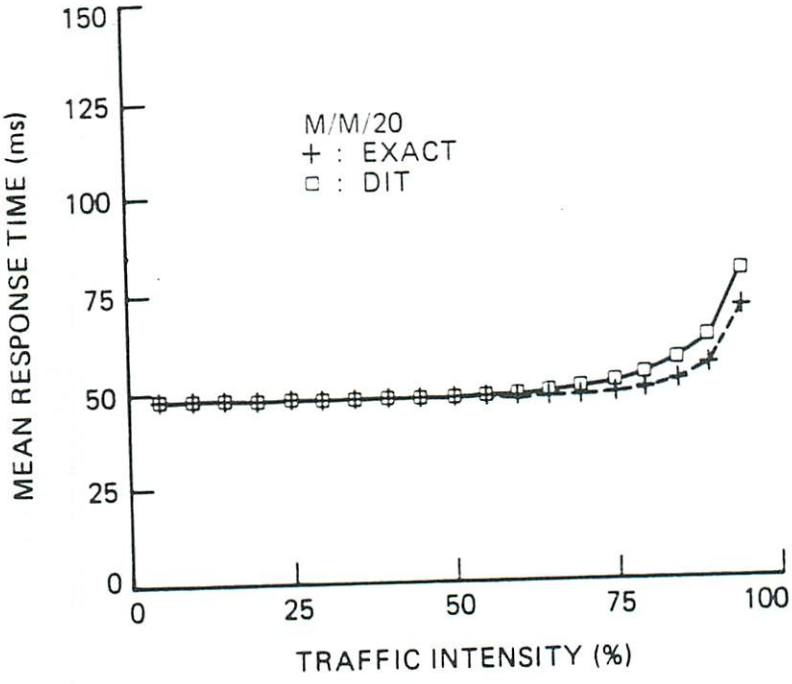
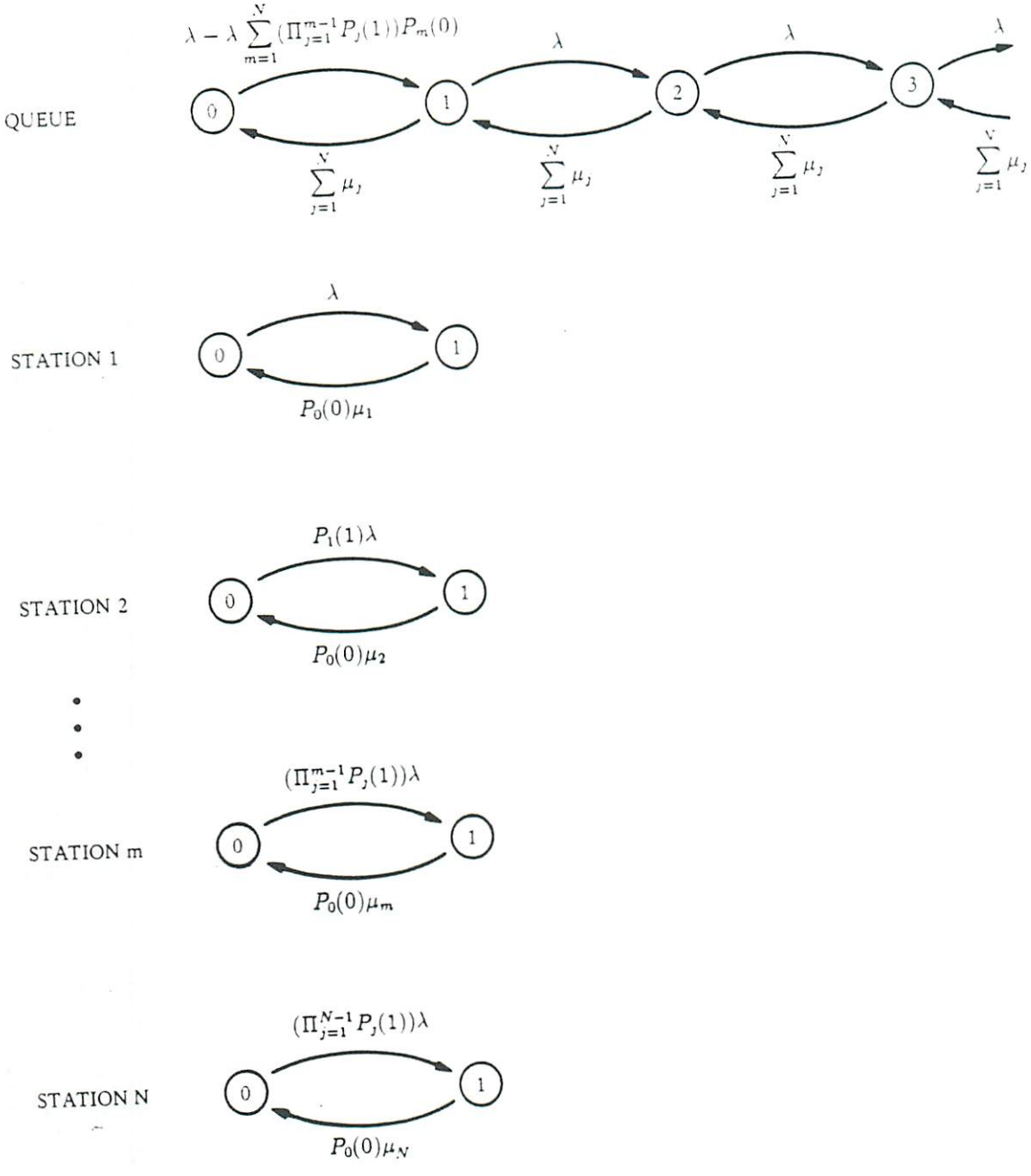


Figure 3.12: Markov chain for a queueing system with ordered heterogeneous servers.



The DIT Markov chain for a queueing system with ordered heterogeneous servers is shown in Figure 3.12. The following two modifications need to be made in the M/M/N DIT model in order to solve Cooper's problem:

$$P_m(1) = \frac{\sum_{j=1}^{m-1} P_j(1)\lambda}{P_0(0)\mu_m + \sum_{j=1}^{m-1} P_j(1)\lambda} \tag{3.38}$$

$$\alpha = \frac{\lambda - \lambda \sum_{m=1}^{N} (\Pi_{j=1}^{m-1} P_j(1)) P_m(0)}{\sum_{j=1}^{N} \mu_j} \tag{3.39}$$

The exact expected service time, $T_{s,exact}$, and the exact expected waiting time, $T_{w,exact}$, for a $n$ servers system are given by Cooper [5]:

$$T_{s,exact} = \frac{1}{\lambda}(p_1 + p_2 + \cdots + p_n) \tag{3.40}$$

$$T_{w,exact} = \frac{1}{\lambda} \sum_{i=1}^{\infty} i P_{n+i} \tag{3.41}$$

$$= \frac{1}{\lambda(1-\rho)} \frac{\rho B_n}{1 - \rho + \rho B_n} \tag{3.42}$$

$P_k$ is the equabilium probability that there are $K$ customers in the system (in service or waiting for service) at an arbitrary instant.

$$P_{n+i} = \begin{cases} [B_n/(1 + AB_n]A_i & \text{when} \quad A < \infty \\ 0 & \text{when} \quad A = \infty \end{cases} \tag{3.43}$$

$$A_i = \begin{cases} 1 & \text{if } i = 0 \\ \dfrac{\lambda_n \lambda_{n+1} \cdots \lambda_{n+i-1}}{\mu_{n+1} \mu_{n+2} \cdots \mu_{n+i}} & \text{if } i = 1, 2, \cdots, \end{cases} \tag{3.44}$$

$$A = \sum_{i=1}^{\infty} A_i \tag{3.45}$$

$$B_j = \gamma_1[\mu(1)]\gamma_2[\mu(2)]\cdots\gamma_j[\mu(j)], \tag{3.46}$$

where $j = 1, 2, \cdots, n$.

$$\gamma_{j+1}(z) = \frac{\gamma_j[z+\mu(j)]}{1-\gamma_j(z)+\gamma_j[z+\mu(j)]} \tag{3.47}$$

where $j = 1, 2, \cdots, n-1$,

and $\gamma_1(z) = \dfrac{\lambda}{\lambda+z}$

$p_j$ is the utilization of the $j_{th}$ ordered server.

$$p_j = \frac{\lambda}{\mu(j)}(B_{j-1}-B_j)\left(1-\sum_{i=1}^{\infty}P_{n+i}\right)+\sum_{i=1}^{\infty}P_{n+i} \tag{3.48}$$

where $j = 1, 2, \cdots, n$ (3.49)

and $B_0 = 1$ (3.50)

Figure 3.13 shows the response time comparison between the exact solution and the DIT approximation model when there are three servers with geometrically decreasing service-rate ratio of 4, 2, and 1. In Figure 3.14 the response time comparision is made for three servers with geometrically decreasing service-rate ratio of 100, 10, and 1. In both cases, the results show that the DIT model gives a very close approximation to the exact solution.

It is interesting to note the 'bending' behavior of the mean response time between 2.5 jobs/s to 5 jobs/s arrival rate. It can be explained as follows: While arrival rate is below 2.5 jobs/s, most of the jobs are served by the faster stations and the mean response time is short. As the arrival rate increases, the faster stations are busy and jobs are routed to the slower station. Thersfore, the overall mean response time increase dramatically. As the arrival rate increases futher, the percentage of the incoming jobs served by the faster

Figure 3.13: Response time comparision between the exact solution and the DIT approximation model with three servers having geometrically decreasing service ratio of 4, 2, and 1.

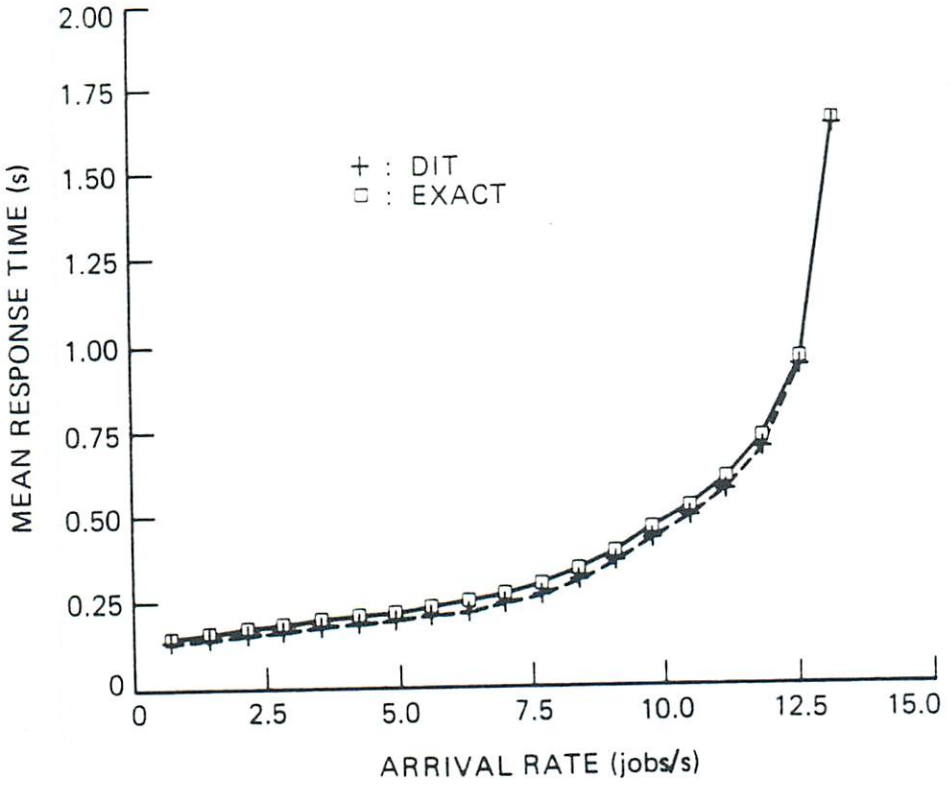
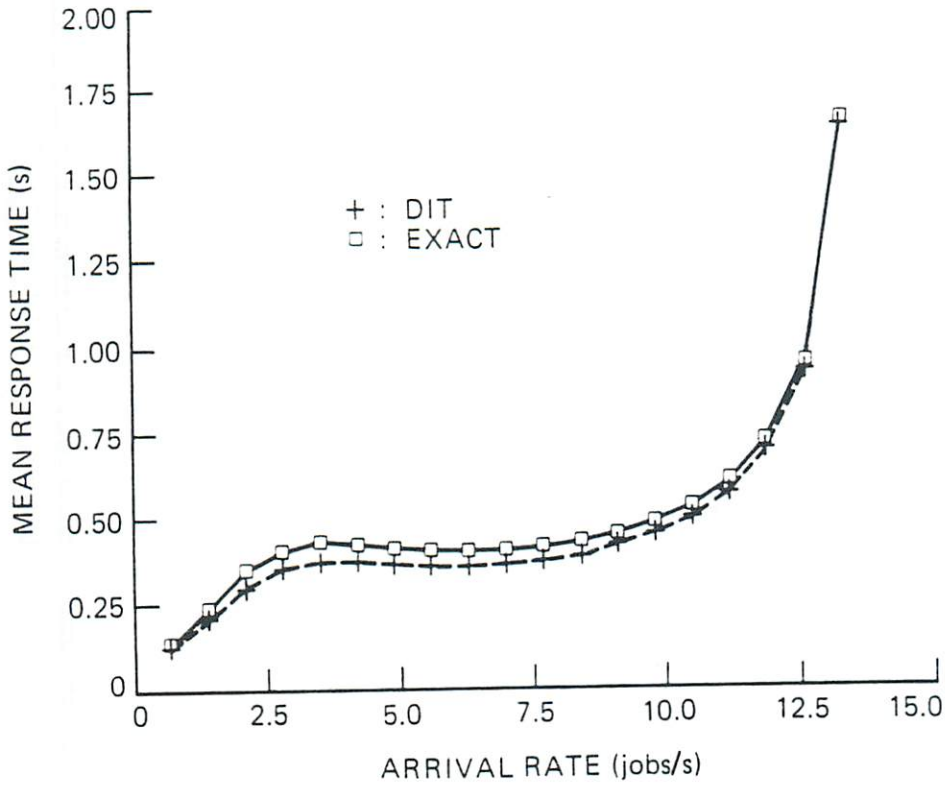


Figure 3.14: Response time comparision between the exact solution and the DIT approximation model with three servers having geometrically decreasing service ratio of 100, 10, and 1.



stations increases and the mean resonse time decreases. This phenomenon was also observed by Tantawi and Towsley [44].

## 3.5 Conclusion

This chapter presents an approximate performance evaluation technique for distributed computer systems with heterogeneous state-dependent interactive queues. This technique provides an elegant and efficient iterative procedure for an intractable $N$-dimension Markov chain. The average response times of several typical interactive-queue problems, including a receiver-initiated load-sharing scheme, a sender-initiated load-sharing scheme and a homogeneous/heterogeneous M/M/K, were studied. The exact solution or simulation validation indicate that our DIT contributes a reasonable estimate of the mean response time for load-dependent interactive queues system.

# Chapter 4

# ERIDA:A new resource-sharing scheme

In this chapter, an enhanced receiver-initiated dynamic algorithm (ERIDA) is proposed for efficient dynamic load sharing in a multiple-station distributed heterogeneous computer system. This algorithm represents an improvement over the performance of the previously developed receiver-initiated load sharing policy [6]. ERIDA dynamically adjusts to the traffic load and does not generate extra overhead during high-traffic conditions. An approximate queueing model validated by a GPSS/PC simulation is presented. These queueing and simulation models are used to evaluate the improvement in the average response time for a multiple-node heterogeneous scenario.

## 4.1 Algorithm Description

An ideal resource-sharing algorithm should have the following characteristics:

a. Dynamic: The load distribution can adapt to rapid system load changes.

b. Decentralized: Each processor determines, on its own, whether to process a job locally or to send the job to some other station for processing. There is no need for a central dispatcher. Since no central dispatcher is required, the problem of a single point of failure is eliminated.

c. Heterogeneous: The computer stations do not have the same processing power.

The enhanced receiver-initiated dynamic algorithm (ERIDA) discussed in this thesis for resource sharing fulfills all the above criteria.

The following is an outline of the ERIDA:

Given N : total number of stations in the heterogeneous system;
$\mu_i$ : service rate of station $i$;
$Q_i$ : queue length at station $i$;
$\varpi_i$ : $\frac{Q_i}{\mu_i}$, the workload indicator of station $i$;
$\vartheta_{low}$ : low workload threshold;
$\vartheta_{high}$ : high workload threshold.

While ( ((a job completes at station i) or wakeup-timeout) and ($\varpi_i \leq \vartheta_{low}$ ) )
do
probe $\varpi_k$ for k=1 to N, k $\neq$ i;
identify the station, j, with the MAX($\varpi_j$);
if $\varpi_j \geq \vartheta_{high}$, then
do
transfer 1 job from station $j$ to station $i$;
(* job is processed at station $i$ and the
results are returned to station $j$ *)
end;
end.

When a job finishes service and leaves a station, the station checks the workload indicator by dividing the queue length by the service rate. If the

workload indicator is below a certain low threshold level, the lightly loaded station initiates a search for the busiest station. If the workload indicator of the busiest station is a above certain high threshold, a job from that busy station is transferred to the lightly loaded station. Thus, the ERIDA provides a method which balances the workload among hosts, resulting in an improvement in the response time and throughput performance of the total system.

The ERIDA adjusts dynamically to the traffic load of each station. When the workload indicator of every station is greater than a certain threshold level, the algorithm generates no overhead - an important advantage.

In Wah's research [46], the priority of the load-balancing process is assumed to be lower than that of regular jobs, thereby preventing the resource-sharing procedure from inhibiting normal operation. However, for our algorithm, it is necessary to assign the highest priority to the resource-sharing process. Otherwise, the lightly loaded stations would not receive the workload indicators and jobs would never be transferred from the heavily loaded stations in a timely manner, i.e., before the heavily loaded stations become lightly loaded.

More detail can be found in Appendix A, which presents a Pascal like high level language implementation of the ERIDA.

The most important advantages of the ERIDA over the basic receiver-initiated load-sharing policy developed by Eager et al.[6] are:

a. The ERIDA uses the local queue length and the local service rate ratio at each host as the workload indicator. In the heterogeneous computer system, it is more efficient to use this workload indicator rather than just the local queue length as described in [6].

b. To prevent an idle station from becoming isolated from the resource-sharing process, as can be happen with Eager's policy, a wakeup timer

is included in the ERIDA. This wakeup timer is used at each idle station to periodically cause the idle station to search for a job that can be transferred from a heavily loaded station.

## 4.2 ERIDA Performance Model

In this section, we show how the approximate analytical model, DIT, presented in the previous chapter can be used to find the average response time of a distributed system with the ERIDA load-balancing scheme. By sharing the processing power, the average response time using the ERIDA should be better than when no load sharing is used. On the other hand, a centralized resource-sharing scheme should have better system response time than the decentralized ERIDA scheme because all the decisions are made by a central controller. These two alternative of 'resource-sharing' schemes will be used as the lower and upper bounds limit checking of our model. Before presenting the ERIDA model, we analyze the upper and lower bound systems.

### 4.2.1 Upper Bound: Independent Systems

We begin by considering a collection of $N$ independent servers. Each station $m$ has a different service rate, $\mu_m$, and is individually accessed by a job stream at an exponential rate $\lambda_m$. This organization corresponds to $N$ parallel independent M/M/1 queues. The mean number of customers present in each independent M/M/1 queue $m$ is then

$$\overline{N}_m = \frac{\rho_m}{1 - \rho_m} \tag{4.1}$$

where

$$\rho_m = \frac{\lambda_m}{\mu_m} \tag{4.2}$$

and, using Little's formula, we have the mean system response time

$$T = \frac{\sum_{j=1}^{N} \overline{N}_j}{\sum_{j=1}^{N} \lambda_j} \tag{4.3}$$

### 4.2.2 Lower Bound: Centralized Resource Sharing

We consider a single infinite queue accessed by the collection of $N$ stations at a total arrival rate equal to $\lambda = \sum_{m=1}^{N} \lambda_m$. First of all, let us consider the homogeneous case where each station has the same service rate, $\mu$. This is just a homogeneous M/M/N system, and the mean response time can be represented by Erlang's delay-call formula [12]:

$$T = \frac{1}{\mu} + [\frac{(\frac{\lambda}{\mu})^N \mu}{(N-1)!(N\mu-\lambda)^2}][\sum_{j=0}^{N-1} \frac{1}{j!}(\frac{\lambda}{\mu})^j + \frac{1}{N!}(\frac{\lambda}{\mu})^N \frac{N\mu}{N\mu-\lambda}]^{-1} \tag{4.4}$$

This formula, however, is not applicable to a heterogeneous system where processors do not all have the same service rate. A heterogeneous M/M/2 queue is studied in [45], and a closed-form solution of the average response time is given by

$$T = \frac{1}{\frac{\mu_1\mu_2(1+2\rho)(1-\rho)^2}{\lambda+\mu_2} + \lambda(1-\rho)} \tag{4.5}$$

where

$$\rho = \frac{\lambda}{\mu_1 + \mu_2} \tag{4.6}$$

The three-server central dispatcher system is a variant of the M/M/3 queue where the service rates of the three processors are not identical. Assume

without loss of generality that $\mu_1 \geq \mu_2 \geq \mu_3$. The state of the system is defined by the triple $(n_1, n_2, n_3)$, where $n_1$ denotes the number of jobs in the queue, including the one at the fastest server (if there is one), and $n_2, n_3 \in \{1, 0\}$ denote the number of jobs at the two slower servers. Jobs wait in line in the order of their arrival. When all three servers are idle, the fastest server is scheduled for service before the other two.[1]. The system diagram and the state diagram are given in Figures 4.1 and 4.2.

Balance equations for the steady state can be written by equating the rate of flow into a state to the rate of flow out of that state. The steady-state probabilities for states $(n, 1, 1)$ can be expressed in terms of state $(1, 1, 1)$ explicitly:

$$P_{n,1,1} = \left(\frac{\mu_1 + \mu_2 + \mu_3}{\lambda}\right)^{n-1} P_{1,1,1} \quad \text{for } n > 1 \tag{4.7}$$

The global balance equations for the states in the cubic arrangement in Figure 4.2 can be represented by their neighbor states and summarized into the following seven equations:

$$\lambda P_{0,0,0} = \mu_1 P_{1,0,0} + \mu_2 P_{0,1,0} + \mu_3 P_{0,0,1} \tag{4.8}$$

$$(\lambda + \mu_1) P_{1,0,0} = \lambda P_{0,0,0} + \mu_2 P_{1,1,0} + \mu_3 P_{1,0,1} \tag{4.9}$$

$$(\lambda + \mu_2) P_{0,1,0} = \mu_1 P_{1,1,0} + \mu_3 P_{0,1,1} \tag{4.10}$$

$$(\lambda + \mu_3) P_{0,0,1} = \mu_1 P_{1,0,1} + \mu_2 P_{0,1,1} \tag{4.11}$$

$$(\lambda + \mu_1 + \mu_2) P_{1,1,0} = \lambda P_{1,0,0} + \lambda P_{0,1,0} + \mu_3 P_{1,1,1} \tag{4.12}$$

$$(\lambda + \mu_1 + \mu_3) P_{1,0,1} = \lambda P_{0,0,1} + \mu_2 P_{1,1,1} \tag{4.13}$$

$$(\lambda + \mu_2 + \mu_3) P_{0,1,1} = \mu_1 P_{1,1,1} \tag{4.14}$$

[1]Note that this queueing discipline does not necessarily give the lowest mean response time. In a two-server system with $\mu_1 \gg \mu_2$, it may be beneficial not to use server 2 at all [32]. Thus this result does not necessarily give a lower bound for heterogeneous load-sharing system. It is a real lower bound only for the homogeneous case.

Figure 4.1: System diagram for a central-dispatcher three-server heterogeneous system.

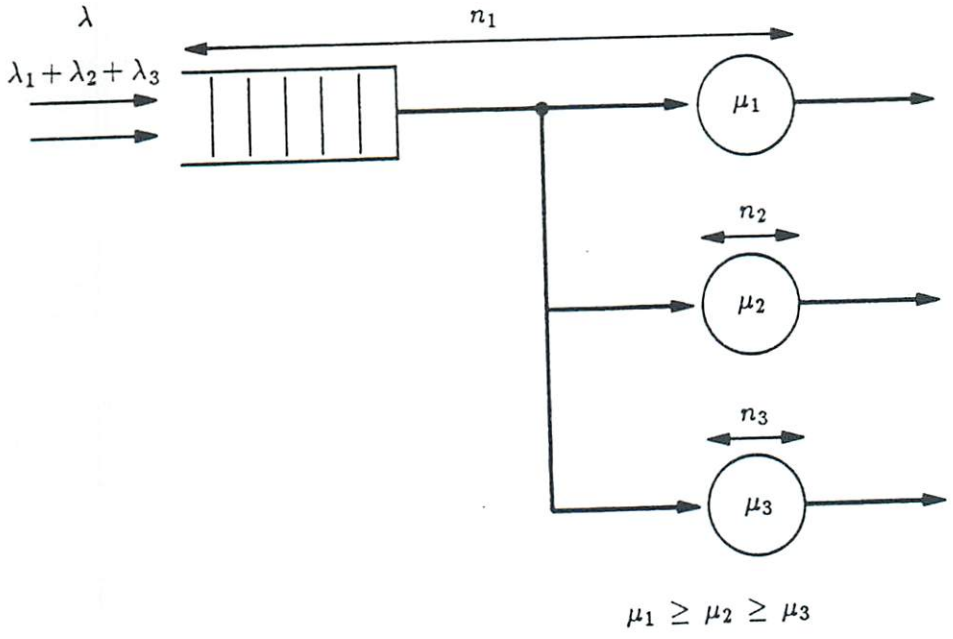
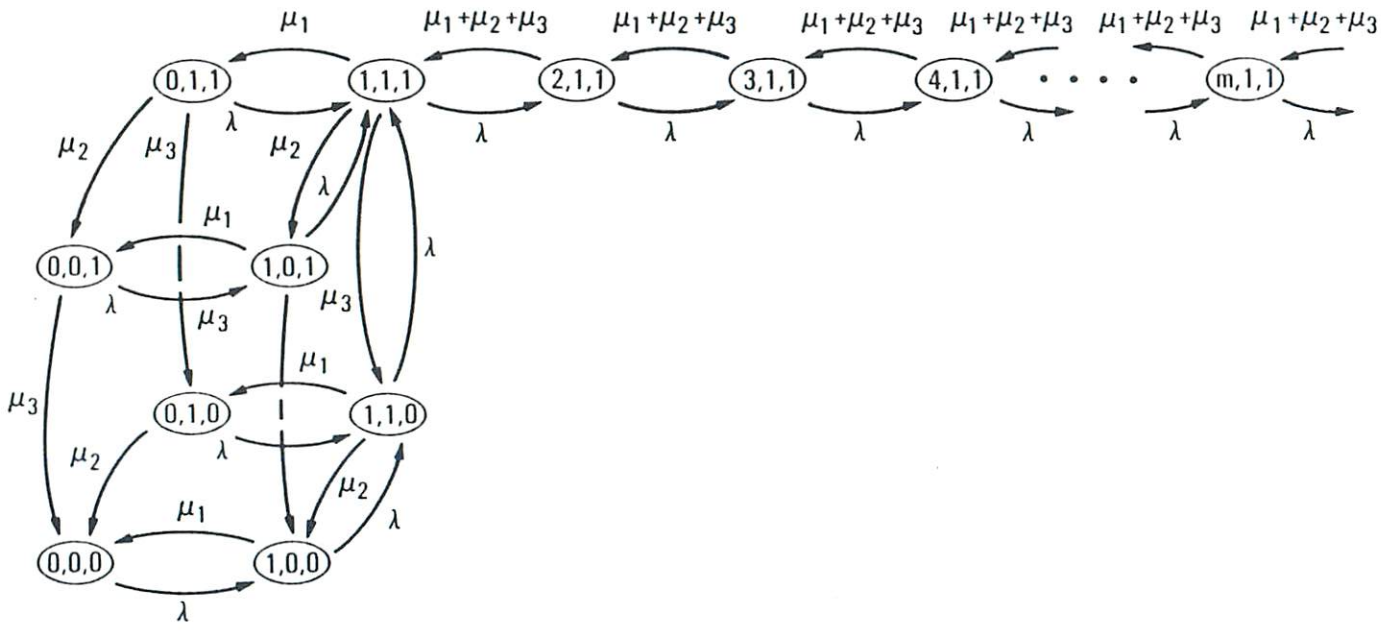


Figure 4.1: System diagram for a central-dispatcher three-server heterogeneous system.



These equations and the boundary condition can be used to find the steady-state probabilities, $\Pi$.

### 4.2.3 ERIDA Analytical Model

A multiple-processor heterogeneous system is considered in which the service rates of the stations are not necessarily identical. Each station is modeled as a queueing center. For a particular station $m$, new jobs arrive at rate $\lambda_m$. The average service rate is $\mu_m$, and the interwakeup time, which is the same for all nodes, is $1/\omega$. The state of the system is defined as the number of jobs in the station, either in the queue or being served. The objective of the analysis is to determine the effects of resource sharing on the average system response time. These effects are a function of the traffic intensity, which is defined as the ratio of the job arrival rate to the job service rate.

The basic assumptions made in this performance study are as follows:

a. There is only one class of task. The task arrival rate and service rate of each processor may be different. The arrival time and service time are exponentially distributed.

b. The average wakeup rate $\lambda_w$ for each idle station is the same, and the wakeup time is exponentially distributed.

c. Each job needs only one resource.

d. The network transmission delay in propagating requests, probing the status, and returning results is negligible. This assumption is valid if the transmission bandwidth is large compared to the traffic on the network.

e. All processing overhead for probing, packing and unpacking data, request, and result transfer is ignored. This assumption is valid if the

processing required to pack and unpack the job is significantly less than the processing required to process the job.

f. For simplicity, the queue length is used as the workload indicator.

g. Lightly loaded stations pull jobs from *any* heavily loaded station rather than selecting the most heavily loaded one.

h. The low threshold is 0. In other words, a station tries to pull a job from other stations while it is idle.

i. The high threshold is 2.

Exact analysis of the algorithm is quite complex, since we have an N-dimensional Markov chain. Figure 4.3 shows the exact ERIDA Markov model for a two-server heterogeneous system.

Using the DIT approach, we find the Markov chain for the ERIDA is as shown in Figure 4.4. Transitions from state $k$ to state $k+1$ correspond to external arrivals plus job transfers, which occur only in state 0. Job transfers occur at a rate $\omega$ (the wakeup rate), provided some other station has more than one job, i.e., with probability $1-\delta_m$, where $\delta_m$ is the probability that all stations except $m$ have 0 or 1 job, i.e.,

$$\delta_m = \prod_{i=1 i\neq m}^{N} (P_i(0) + P_i(1)) \tag{4.15}$$

Downward transitions in the Markov chain occur due to:

a. Service completions at a rate $\mu_m$.

b. When the queue length is 2 or more, the queue length of station $m$ can be reduced through load sharing. The reduced rate, $\Phi_m \mu_m$, can

Figure 4.3: Exact ERIDA Markov model for a two-server heterogeneous system.

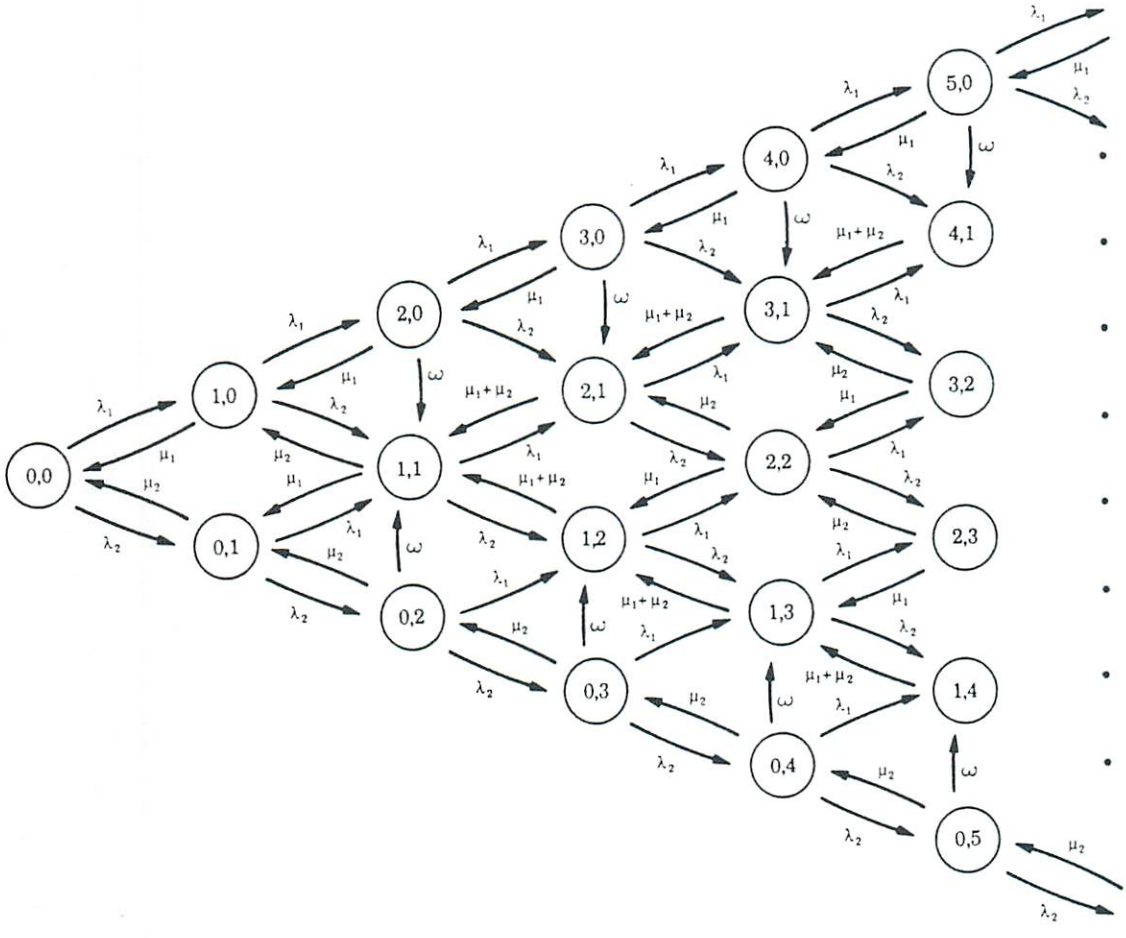
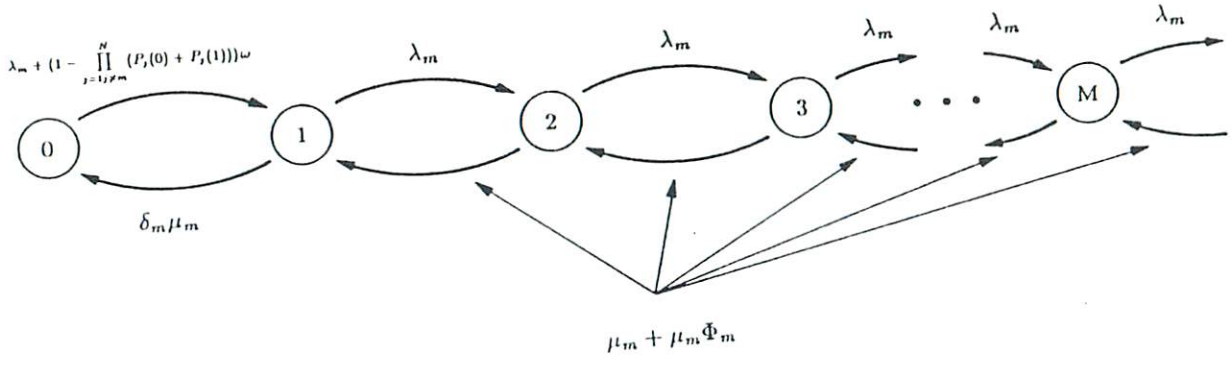


Figure 4.4: Markov model for ERIDA.



be obtained from the system flow balance equation. The system flow balance equation can be written as:

$$\sum_{j=1}^{N} \Phi_j \mu_j (1 - P_j(0) - P_j(1)) = \sum_{j=1}^{N} P_j(1)\mu_j(1-\delta_j) + \sum_{j=1}^{N} P_j(0)(1-\delta_j)\omega \quad (4.16)$$

The left side of the equation represents the total extra service obtained for all the stations with queue length of two or more. The right side of the equation represents the total service provided by the all the lightly loaded stations. $P_j(0)(1-\delta_j)\omega$ is the job acquisition rate when remote station $j$ is woken up from idle; and $P_j(1)\mu_j$ is the job acquisition rate when remote station $j$ has just finished the last job and interrogates other stations for another job. Thus, $\sum_{j=1\ j\neq m}^{N}(P_j(0)(1-\delta_j)\omega + P_j(1)\mu_j)$ is the total load-balancing service power.

The load sharing rate, $\Phi_m \mu_m$, can be approximated as

$$\mu_m \Phi_m = \mu_m \frac{\sum_{j=1}^{N} P_j(1)\mu_j(1-\delta_j) + \sum_{j=1}^{N} P_j(0)(1-\delta_j)\omega}{\sum_{j=1}^{N}(1 - P_j(0) - P_j(1))} \quad (4.17)$$

Combining the above two factors, the downward transition rate is

$$\mu_m + \mu_m \Phi_m \quad (4.18)$$

Given this model, we can immediately proceed to the iteration described in Chapter 3. For clarity, we give here the solution to the birth/death equations for station $m$:

$$P_m(k) = \left(\frac{\lambda_m}{\mu_m + \mu_m \Phi_m}\right)^{k-1} \left(\frac{\lambda_m + (1-\delta_m)\omega}{\mu_m \delta_m}\right) P_m(0) \quad (4.19)$$

For convenience we define

$$\rho_m = \frac{\lambda_m}{\mu_m + \mu_m \Phi_m} \tag{4.20}$$

and

$$\sigma_m = \frac{\lambda_m + (1 - \delta_m)\omega}{\mu_m \delta_m} \tag{4.21}$$

We have

$$P_m(0) = \frac{1 - \rho_m}{1 - \rho_m + \sigma_m} \tag{4.22}$$

and

$$\overline{N}_m = \frac{\sigma_m}{(1 - \rho_m)(1 - \rho_m + \sigma_m)} \tag{4.23}$$

and hence the average time in system for the whole system can be obtained by Little's formula [22]:

$$T = \frac{\sum_{m=1}^{N} \overline{N}_m}{\sum_{m=1}^{N} \lambda_m} \tag{4.24}$$

We see that the transition rates, and hence the steady-state solution, for node $m$ depend on the steady-state probabilities of the other queues in the system. Iteration is used to produce an approximate result which is found to be in close agreement with simulation, see Chapter 3 for more details.

## 4.3 Numerical Results

In this section we present results from the DIT approximate model described in Section 4.2.3 and a comparison with simulation. To develop the simulation analysis for the resource sharing, one assumption was made that is different from the queuing model discussed in Section 4.2.3.: The lightly loaded stations always pull jobs from the busiest stations. We also compare the ERIDA to no load sharing and to a centralized queuing system. We first consider a

homogeneous two-node case, then a two-node and a three-node heterogeneous case, and finally a homogeneous 10-node and a homogeneous 20-node case.

### 4.3.1 Homogeneous Two-Node System

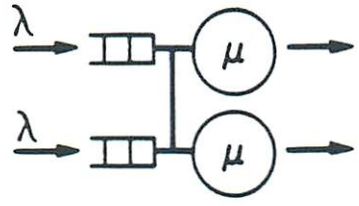
In a homogeneous two-node system, two service stations have the same service rate and the arrival rate at the two stations is also assumed to be equal.

The approximate ERIDA queueing model is validated by a GPSS/PC simulation. The comparison of results from the average response time generated by the approximate queuing model and the results from the simulation exercise are shown in Figure 4.5. This figure shows that our approximate queueing model is very close to the simulation result.

In Figure 4.6 the results of the ERIDA are compared against the case without load sharing, i.e., the case of multiple independent M/M/1 queues. We find that the average response time is always improved when the ERIDA resource-sharing scheme is applied. Obviously, the independent M/M/1 queue system is inefficient, since there may be jobs queued up in front of one of the stations when the other one is idle. The ERIDA is less effective when the traffic intensity is low, since almost no queues are developed at either processor.

Comparing the ERIDA to a centralized resource-sharing system (M/M/2), Figure 4.7 shows that the ERIDA performance is worse than the centralized M/M/2 system. The inefficiency of the ERIDA can be illustrated by the following scenario: If one of the stations finishes the jobs in its queue and sees that all the other stations are not heavily loaded, this station will go idle and remain so until either another local job arrives or until wakeup timeout. This idle station will not be able to share the load of any other stations that become overloaded during its idle period. For this homogeneous system configuration, we can conclude that a centralized resource-sharing scheme provides the best

Figure 4.5: Average response time comparison between DIT analysis and simulation for a two-server homogeneous resource-sharing system. The arrival rate is evenly distributed.



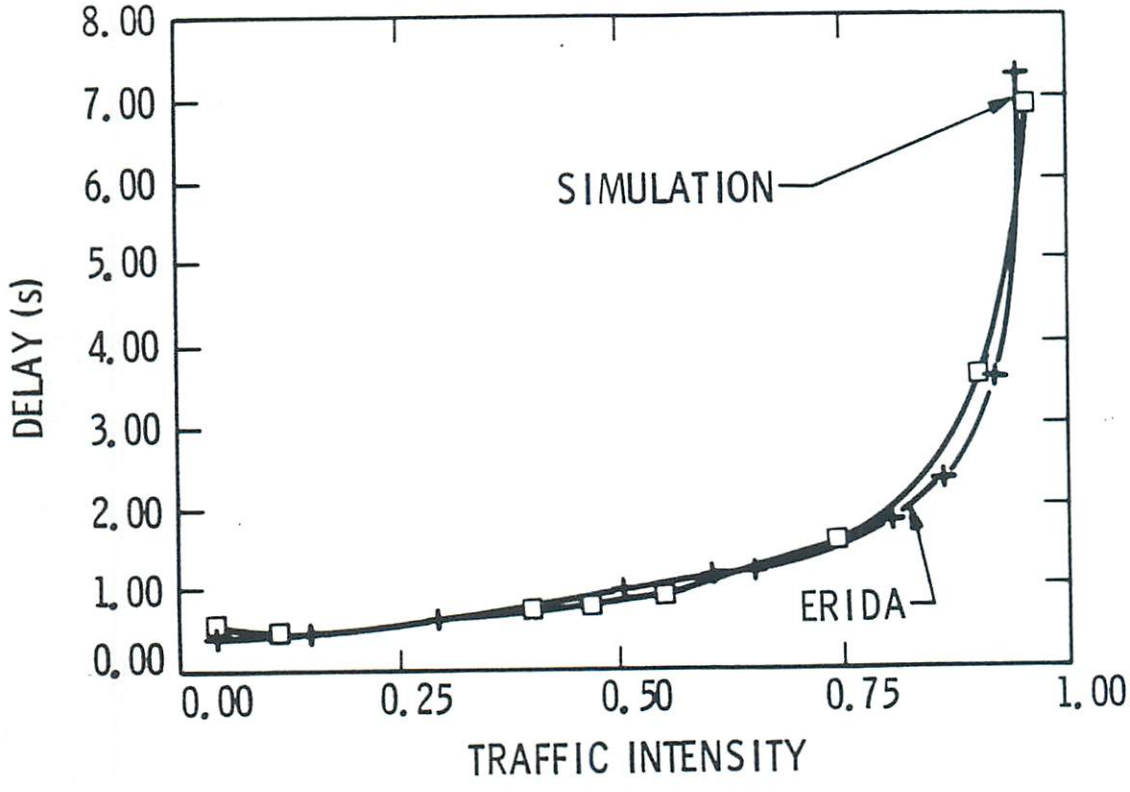
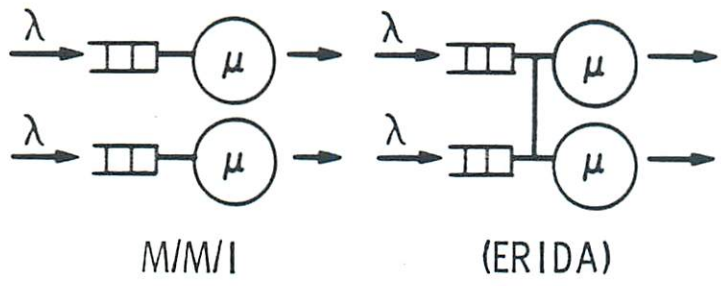
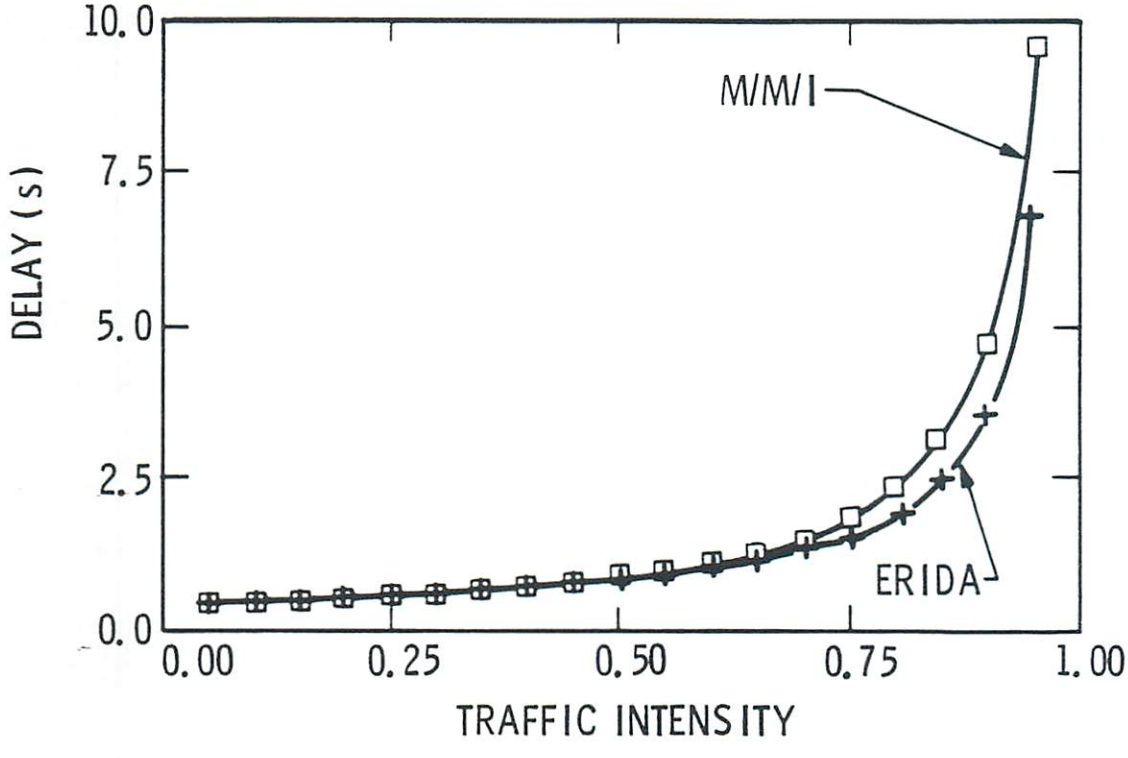



Figure 4.6: Comparison of ERIDA and non-resource-sharing algorithm average response time in a two-server homogeneous system. The arrival rate is evenly distributed.



performance, since no job will wait if any station is idle (although this scheme is typically not practical for implementation).

### 4.3.2 Heterogeneous Two-Node System

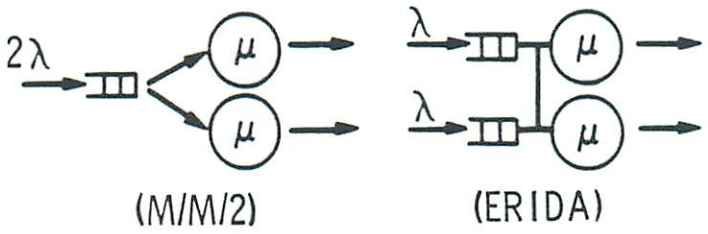
We now allow the two nodes to have different service rates. In the example, the fast server is twice as fast as the slow one. The workload is evenly distributed, i.e., the arrival rate to the fast server is twice that to the slow one. A comparison of ERIDA average response time results generated by the approximate queuing model and the results from a simulation exercise are shown in Figure 4.8. The comparison again shows that the approximate queueing model is very close to the simulation result.

In Figure 4.9 the results of the ERIDA are compared against the case without load sharing, i.e., multiple independent M/M/1 queues. We find that the average response time is improved when the ERIDA resource-sharing scheme is applied. The ERIDA is less effective when the traffic intensity is low, since almost no queues are developed at either processor. In this figure, we also show the response times for a stand-alone fast server and a stand-alone slow server.

When the service rate ratio is high, a more significant improvement can be achieved by use of the ERIDA, as shown in Figure 4.10. In this system, the fast server is four times as fast as the slow server. If the arrival rate is not evenly distributed, the ERIDA can be expected to provide even greater improvement in system efficiency.

In Figure 4.11, we compare the ERIDA to the centralized resource-sharing model. Clearly, the centralized scheme has a better response time. However, it is too restrictive for the distributed system design and has much lower reliability.

Figure 4.7: Comparison of ERIDA and centralized M/M/2 average response time in a two-server homogeneous system. The arrival rate is evenly distributed.



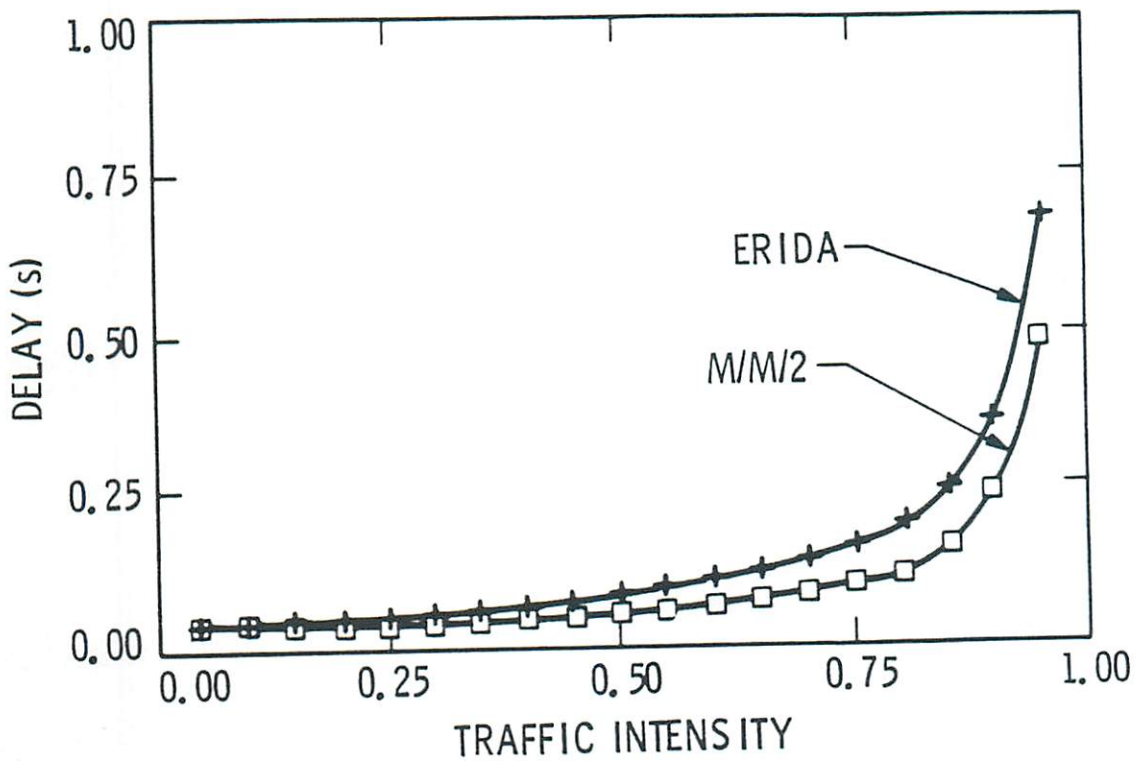
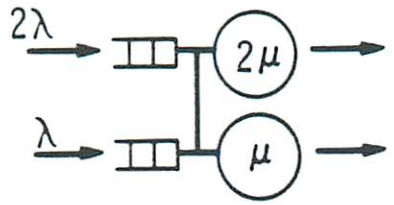



Figure 4.8: Average response time comparison between DIT analysis and simulation for a two-server heterogeneous resource-sharing system. The fast server is twice as fast as the slow one, and the arrival rate is evenly distributed.



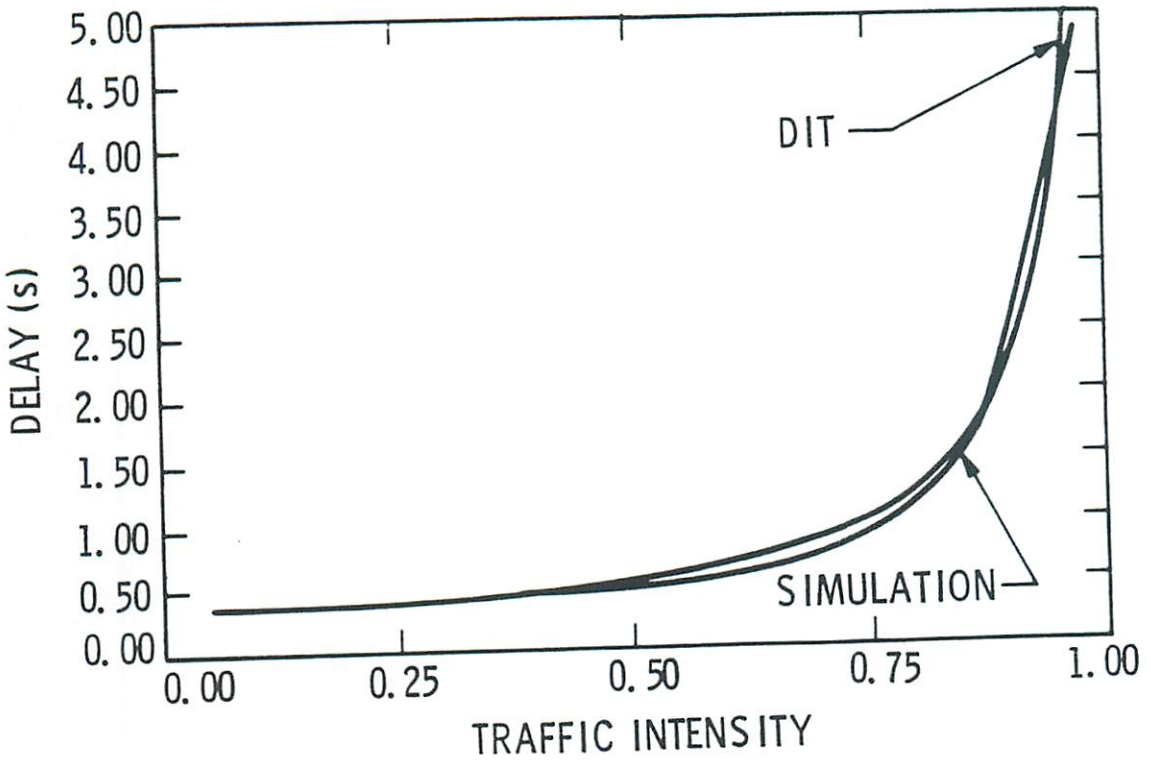
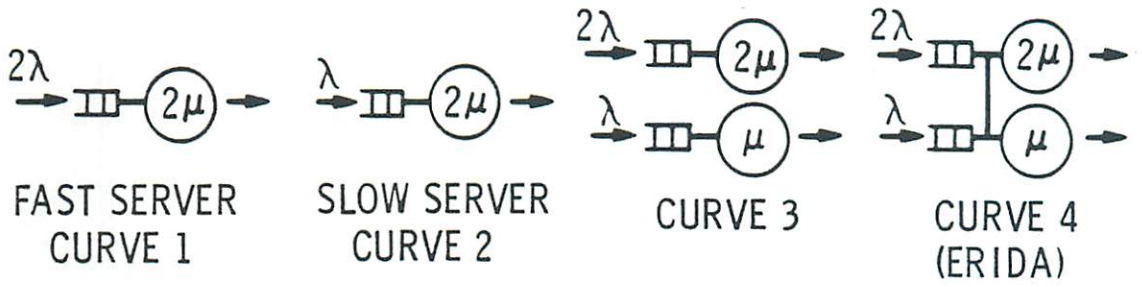



Figure 4.9: Comparison of ERIDA and non-resource-sharing algorithm average response time in a two-server system. The fast server is twice as fast as the slow one, and the arrival rate is evenly distributed.



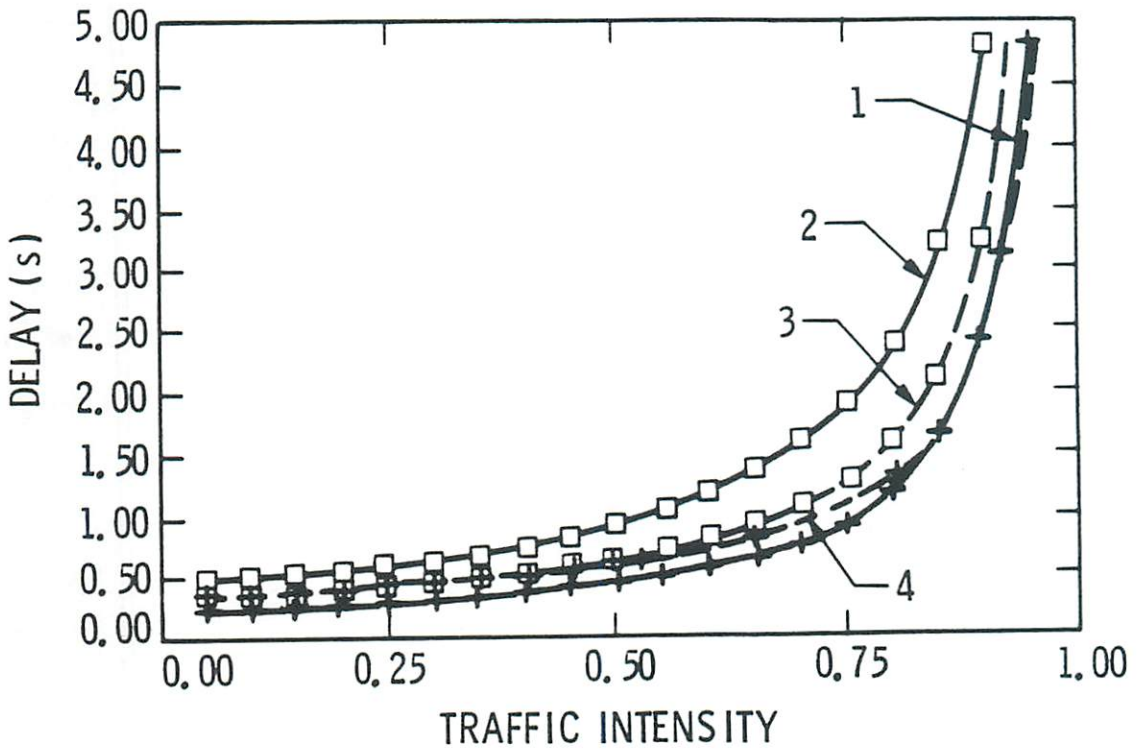
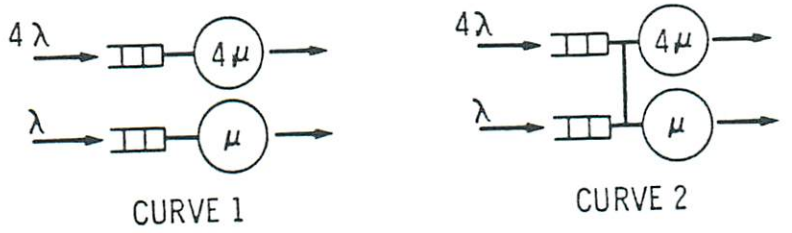



Figure 4.10: Comparison of ERIDA and non-resource-sharing average response time in a two-server system. The fast server is four times as fast as the slow one, and the arrival rate is evenly distributed.



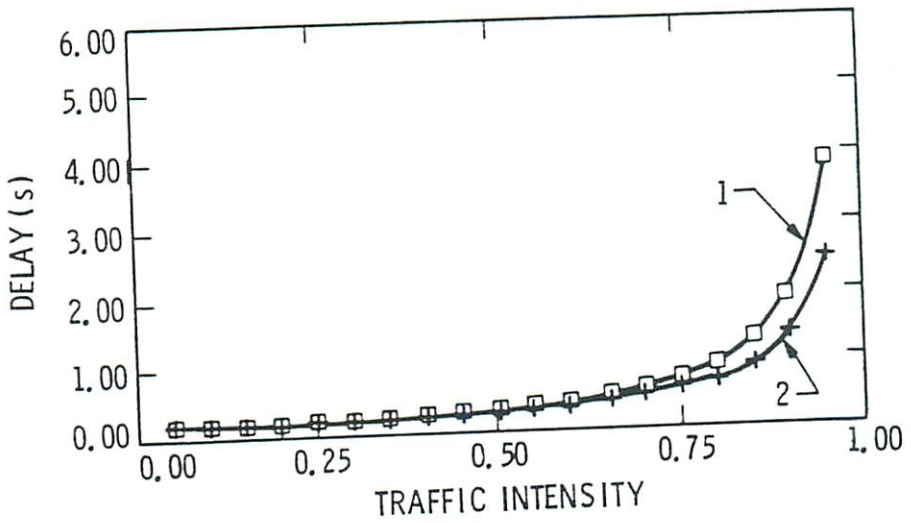
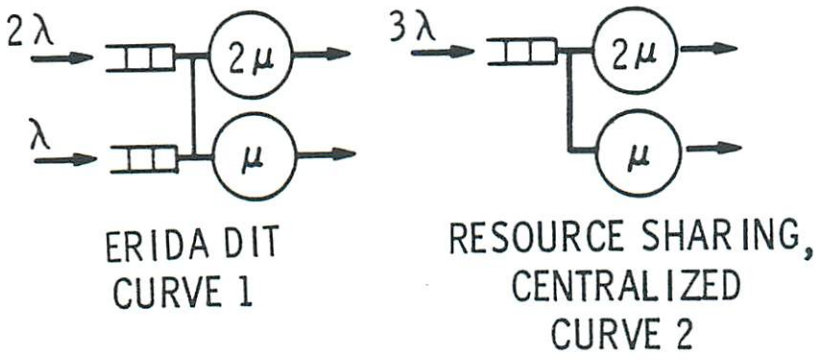
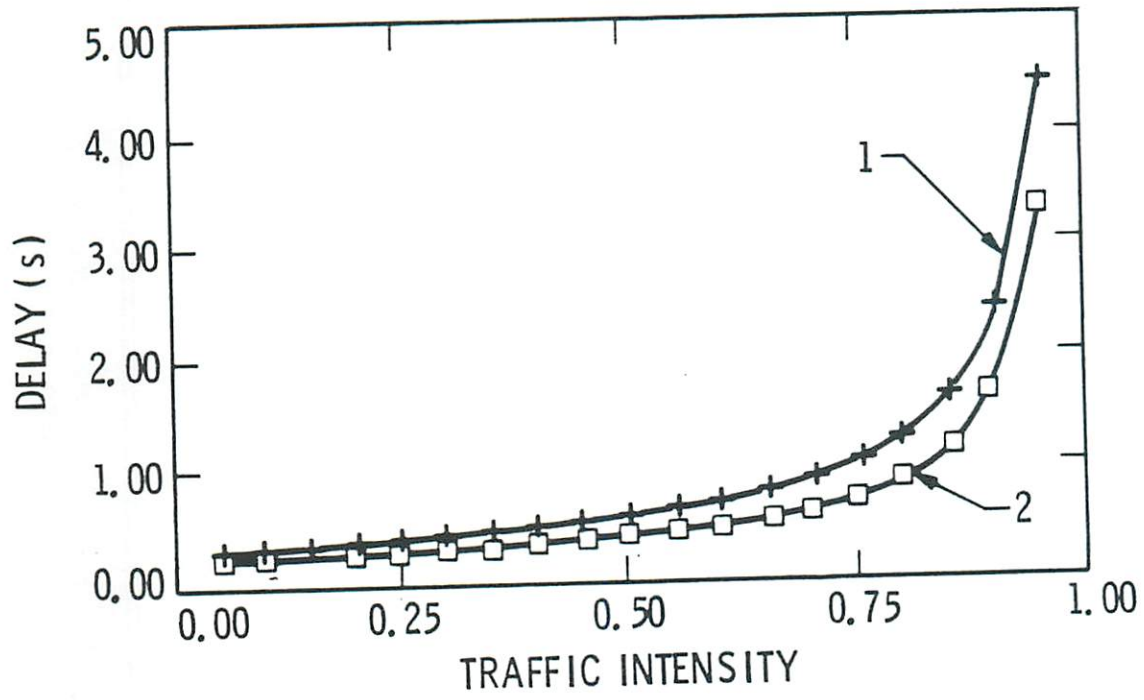



Figure 4.11: Comparison of average response time between two-server centralized and decentralized resource-sharing schemes. The fast server is twice as fast as the slow one.



### 4.3.3 Heterogeneous Three-Node System

A distributed computer system with three heterogeneous stations is analyzed in this section. The results from the average response time of the ERIDA generated by the approximate queuing model and the results from the GPSS/PC simulation exercise are compared in Figure 4.12. Again, the comparison shows that our approximate queueing model agrees very closely with the simulation results.

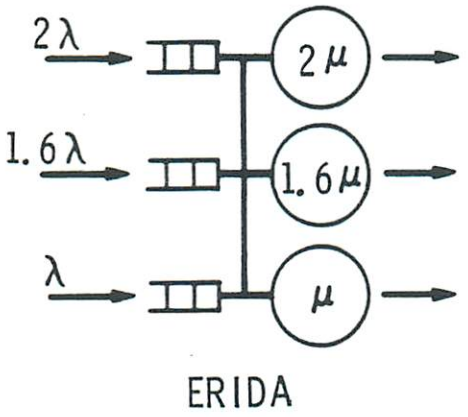
As discussed in Chapter 1, if there are more stations in the resource-sharing system, the probability that they can share their load is greater. Figure 4.13 shows the comparison of ERIDA and non-resource-sharing (M/M/1) average response time versus traffic intensity in a three-server system.

Response time comparisons of a central dispatcher scheme and a decentralized ERIDA resource-sharing scheme for a three-node system are shown in Figure 4.14, based on a heterogeneous M/M/3 analysis. Again, the centralized scheme clearly has a better response time performance. With more stations in the resource-sharing system, the probability that they can share their load is greater, since the probability of imbalance (one node heavily loaded when others are idle) is greater [32]. Figure 4.15 clearly shows that the system response time is reduced with more stations in the ERIDA.

### 4.3.4 Homogeneous 10-Node System

To investigate the behavior of this algorithm in large systems, we applied it to a system with a large number of stations. First, a system with ten homogeneous stations is analyzed in this section. The results from the average response time of ERIDA generated by the approximate queuing model and the results from the GPSS/PC simulation exercise are compared in Figure 4.16. The behavior

Figure 4.12: Average response time comparison between DIT analysis and simulation for a three-server resource-sharing system. The service ration is 2:1.6:1. The arrival rate is evenly distributed.



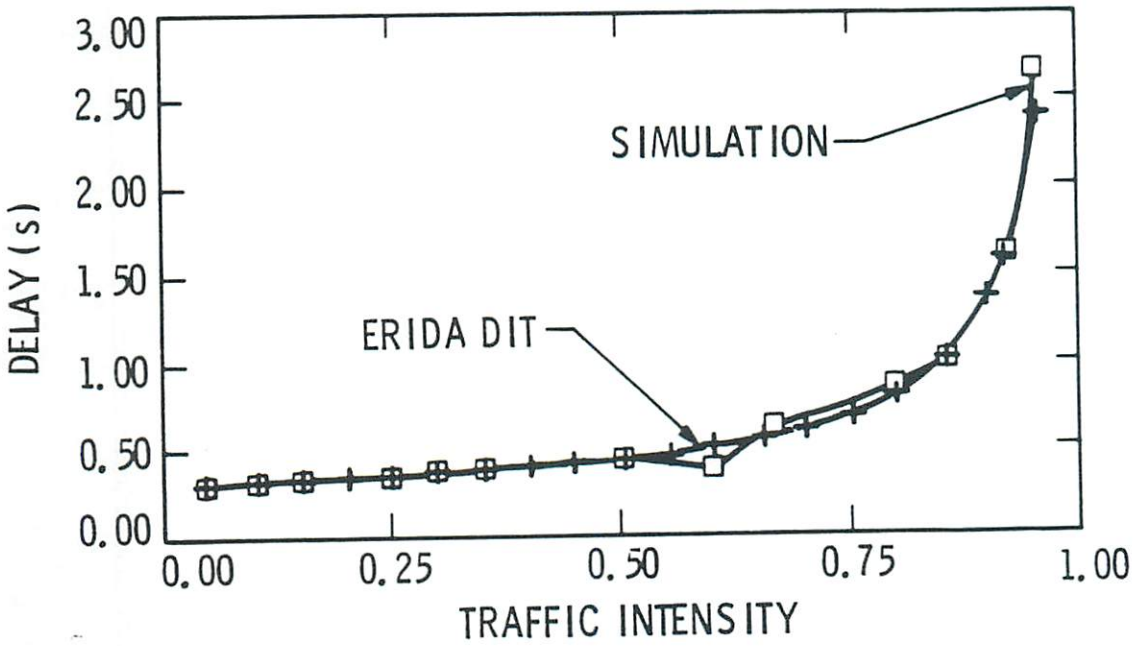
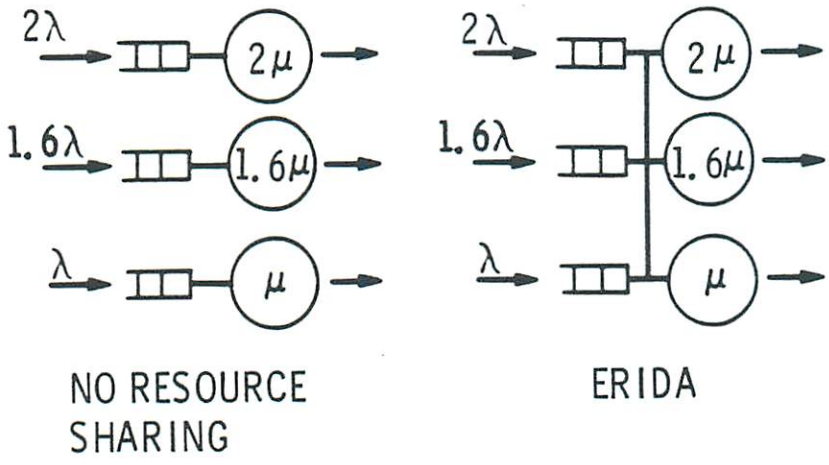



Figure 4.13: Comparison of ERIDA and non-ERIDA algorithm average response time versus traffic intensity in a three-server system. The service ratio is 2:1.6:1. The arrival rate is evenly distributed. Curve PK is the exact M/M/1 result.



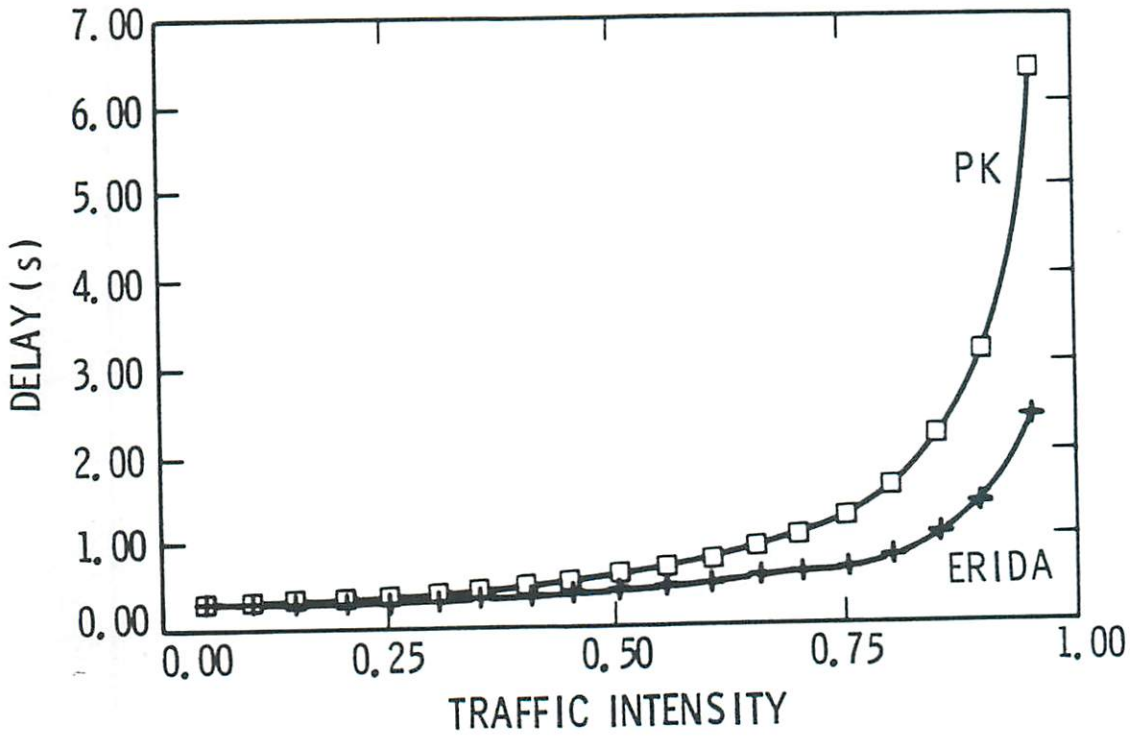
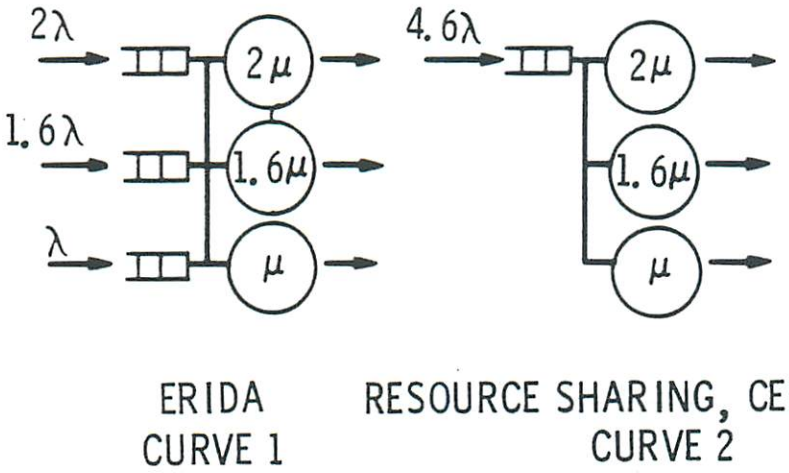



Figure 4.14: Comparison of average response time between centralized and decentralized three-server resource-sharing schemes.



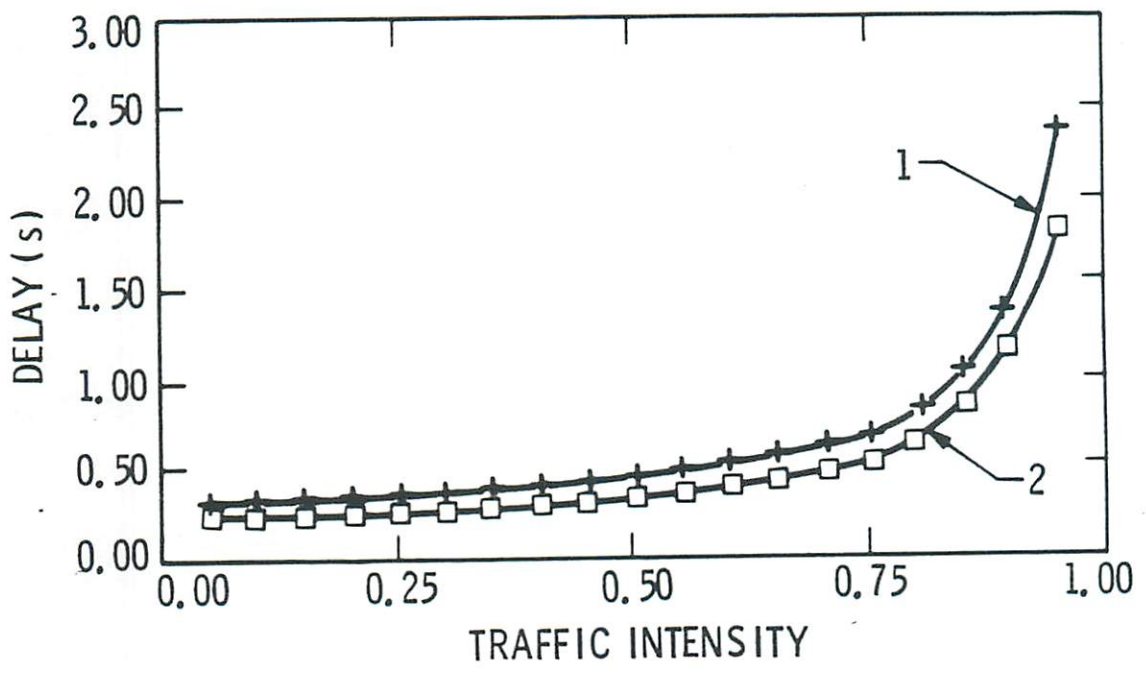
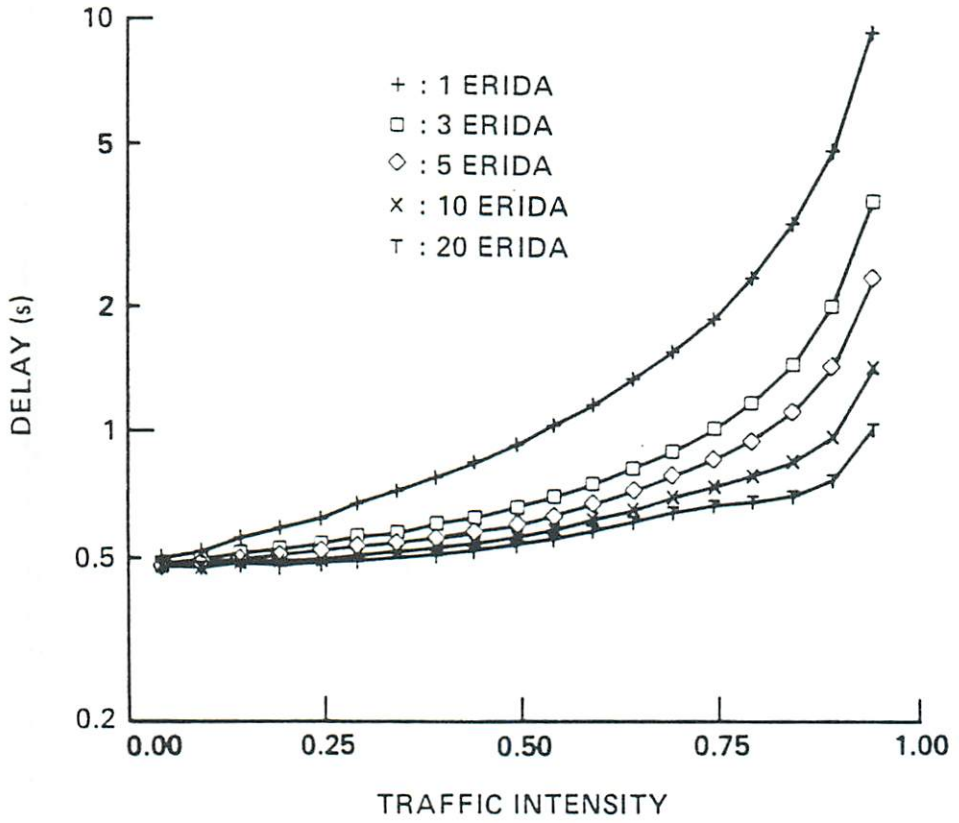



Figure 4.15: Average response time comparison in ERIDA approximate queueing analysis with various numbers of stations in the system.



is consistent with the results of the previous three configurations.

### 4.3.5 Homogeneous 20-Node System

In this section the number of homogeneous stations is increased to 20. The results from the average response time of ERIDA generated by the approximate queueing model and the results from the GPSS/PC simulation exercise are compared in Figure 4.17. The behavior is consistent with the results of the previous three configurations. The result shows that the approximate model accurately predicts the system performance even if the number of stations is large. The difference noted at traffic intensity $\rho = 0.8$ is due to the fact that our analytical model pulls jobs from *any* heavily loaded station rather than just the *most* heavily loaded station.

## 4.4 Wakeup Frequency

To prevent an idle station from becoming isolated from the resource-sharing process, a wakeup timer is included in the ERIDA scheme. This wakeup timer at each station causes an idle station to search periodically for remote heavily loaded stations. In this section, the effect of wakeup rate on response time is investigated.

A comparison of the ERIDA response times for low wakeup rate, $\omega = \frac{1}{10}\mu_m$, high wakeup rate, $\omega = 10\mu_m$, and perfect load balancing, M/M/10 is shown in Figure 4.18. With a higher wakeup rate, the no-overhead ERIDA has a shorter response time.

However, in the real operational environment, frequent wakeup of the idle station to conduct status polling incurs higher communication and computation overhead due to excessive use of polling messages.

Figure 4.16: Average response time comparison among ERIDA DIT (ERIDA-APX: □), ERIDA simulation (ERIDA-SIM: ◇), centralized (M/M/10: X), and an independent system (M/M/1: +), each operating in a ten-server homogeneous system.

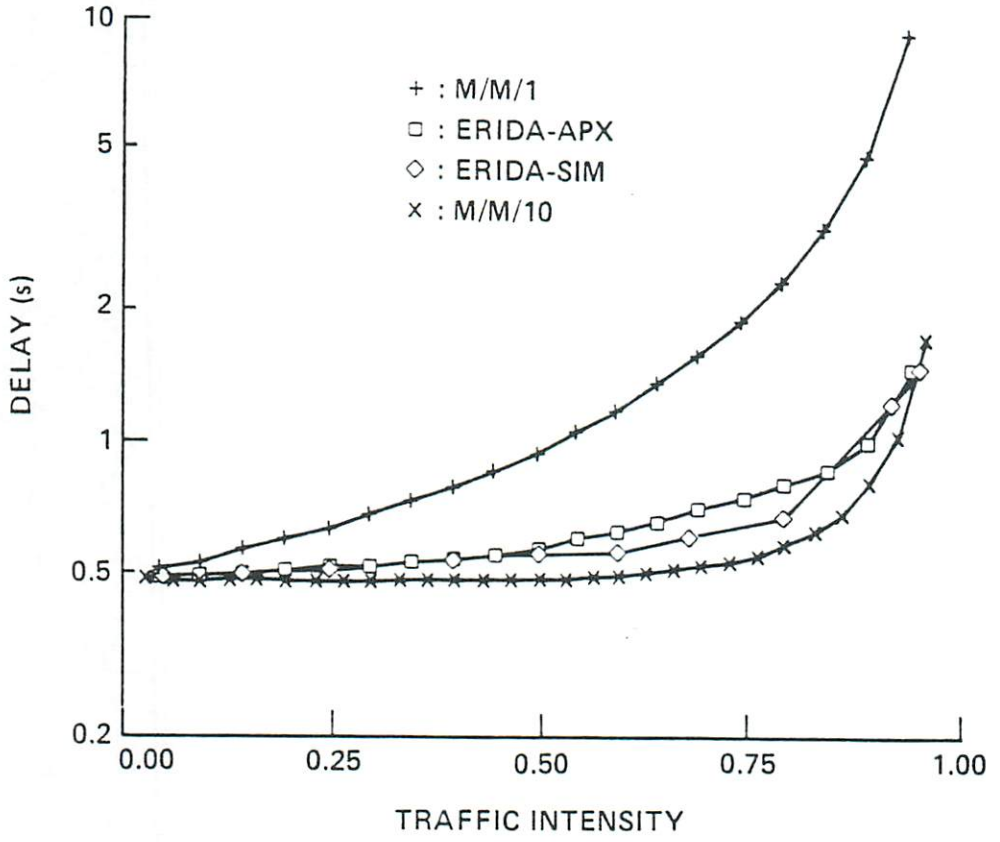


Figure 4.17: Average response time comparison among ERIDA DIT(ERIDA-APX: □), ERIDA simulation(ERIDA-SIM: ◇), centralized(M/M/20: X), and an independent system(M/M/1: +), each operating in a 20-server homogeneous system.

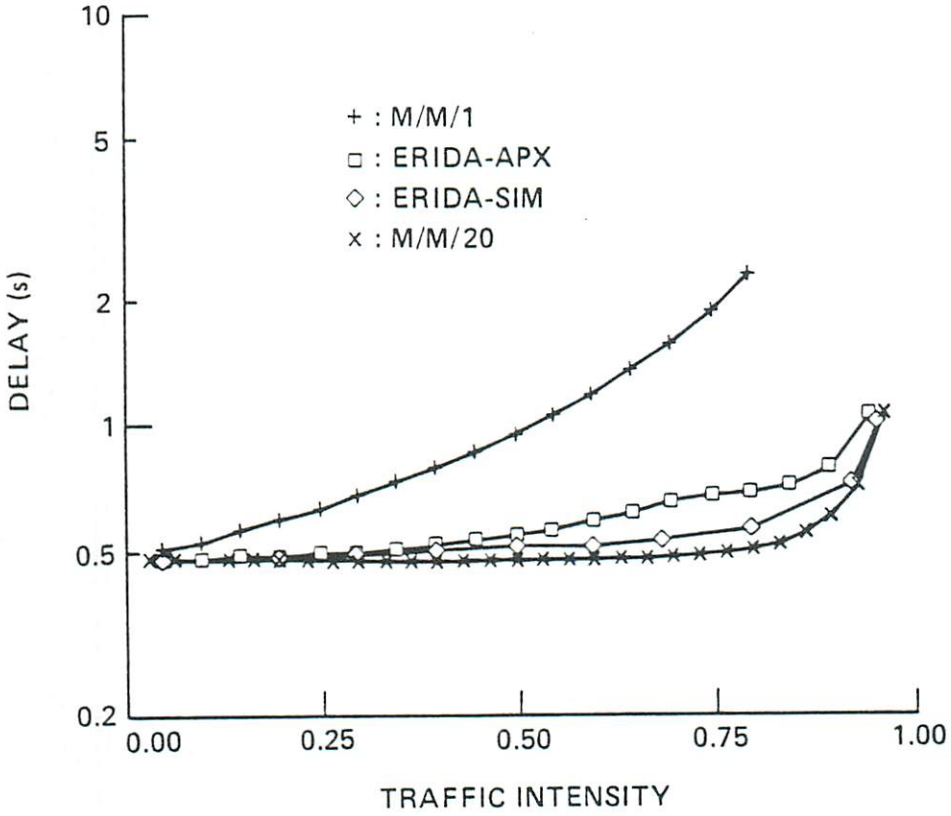
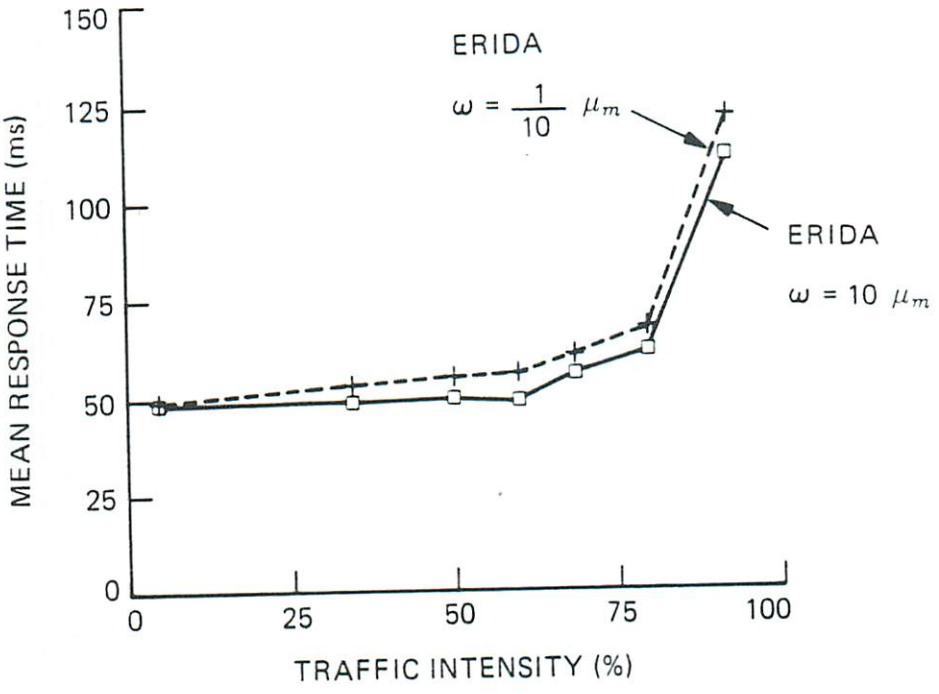


Figure 4.18: Average response time comparison among ERIDA with $\omega = \frac{1}{10}\mu_m$, ERIDA with $\omega = 10\mu_m$, and perfect load balancing scheme, M/M/10 each operating in a 10-server homogeneous system.



## 4.5 ERIDA Communication Overhead Study

In the network resource-sharing environment, the system status and transferred jobs will be available only after some delay. When the network is slow, this delay may be significant. In the previous section, the transmission delay of the transferred jobs was not included in the ERIDA analytical model because of its complexity. In this section, the overhead due to transferring of jobs in a dynamic resource-sharing environment is investigated.

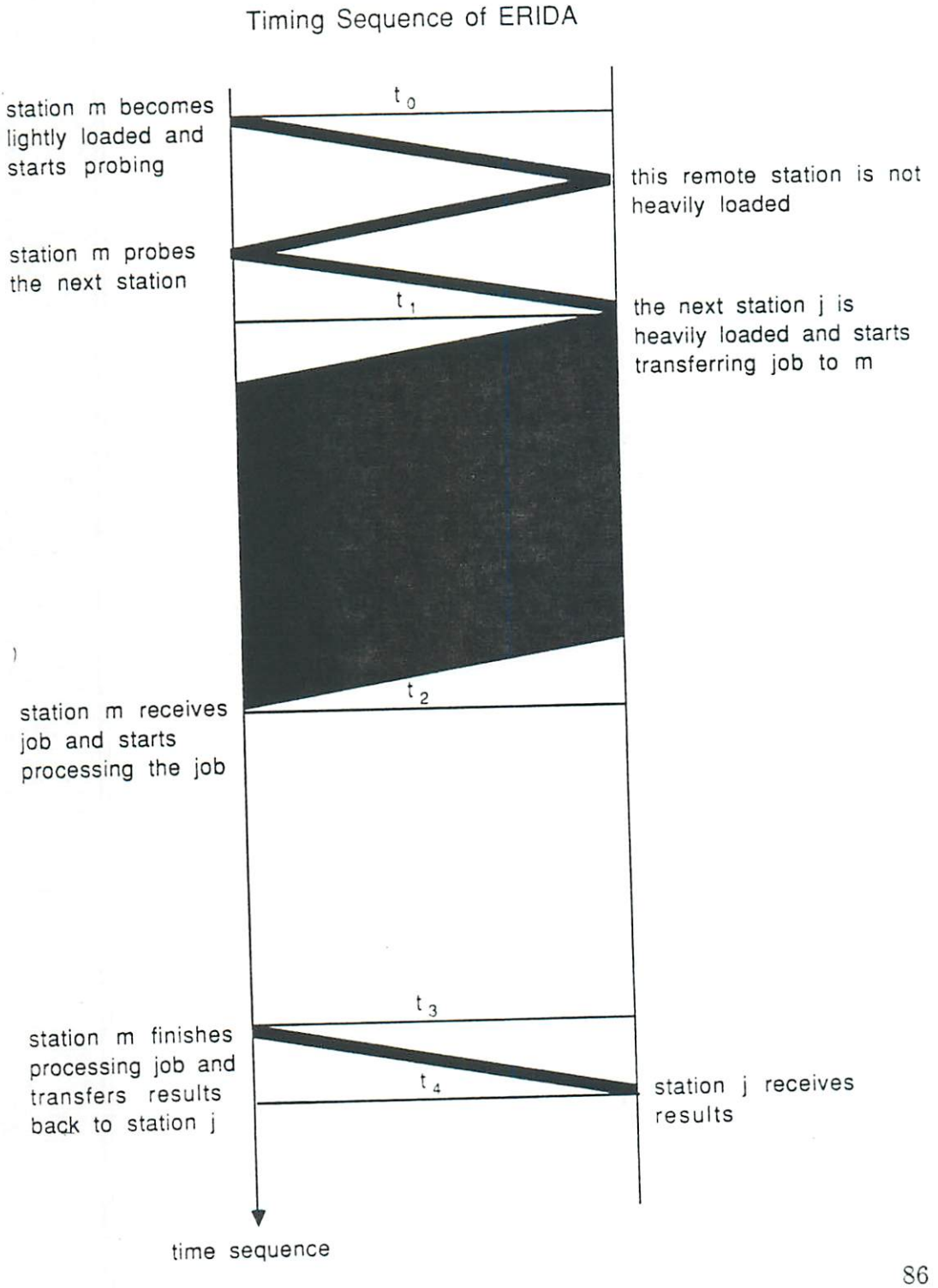
### 4.5.1 Communication Overhead Model Description

In the ERIDA, jobs are transferred from heavily loaded stations to lightly loaded ones over the local area network. Figure 4.19 shows the timing sequence of the ERIDA handshaking scheme between the job-requesting station $m$ and the remote overloaded station $j$. Time advances from the top to the bottom. The shaded bands between the two vertical lines illustrate the activities over the local area network (LAN).

At $t_0$, station $m$ becomes lightly loaded and starts probing the other remote stations to find a heavily loaded one. After one or more probes, station $j$ is identified as a heavily loaded station at $t_1$. Station $j$ then starts transferring the job and all the supporting files to station $m$. The transfer process is completed at $t_2$. The transferred job is then processed at station $m$ and is completed at $t_3$. The result of the processing is transferred back to station $j$. The result packets finish transferring at $t_4$.

The communication delay includes the packet transmission delay and the LAN media propagation delay. The propagation delay is much smaller than the transmission delay. The transmission delay is proportional to the amount of the transferred information. The probing request and the probing status

Figure 4.19: Timing sequence of ERIDA with communication overhead.



messages are small compared to the size of the job. In our communication overhead study, only job transfer delay is considered. In other words, only the delay between times $t_1$ and $t_2$ is considered.

The difference between this overhead study and Wah's study [47] is that Wah's approach can be used only in a homogeneous computer system.

### 4.5.2 Analytical Model for ERIDA with Communication Overhead

The analytical model is derived using the following assumptions:

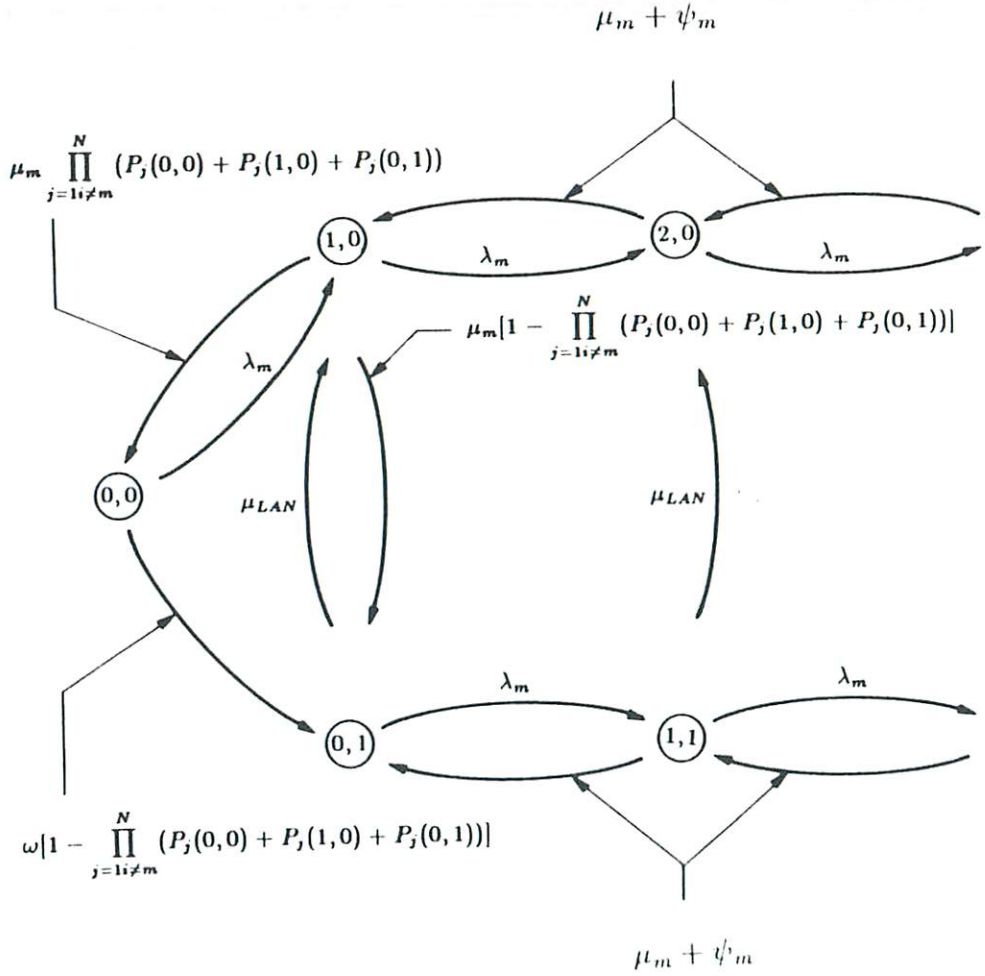
A. The network service time is exponentially distributed. The service rate for the local area network is $\mu_{LAN}$.

B. Each load-balancing job is serviced by the network once. The communication delays for probing status, status return, and job result transfer are negligible.

At each station $m$, the state of the system is defined by the tuple $(n_1, n_2)$, where $n_1 \geq 0$ denotes the number of jobs in the queue, and $n_2 \in (0, 1)$ denotes the job transfer state of the local area network. The job transfer state is either 0, indicating that there is no job being transferred to station $m$; or 1, indicating that there is a job being transferred to station $m$ via the LAN. The Markov chain for this system is shown in Figure 4.20.

In the idle state, $P_m(0,0)$, job transfers occur at a rate $\omega$ (the wakeup rate), provided some other station has more than one job, i.e., with probability $1 - \phi_m$, where $\phi_m$ is the probability that all stations except $m$ have 0 or 1 job, i.e., $\phi_m = \prod_{j=1 i \neq m}^{N} (P_j(0,0) + P_j(1,0) + P_j(0,1))$. Therefore, the state transition rate from idle state to state (0,1) is $\omega(1 - \phi_m)$. A job completes

Figure 4.20: Markov chain of ERIDA with communication overhead.



transfer over the LAN at the LAN transfer rate, $\mu_{LAN}$, and the system moves to state $(k,0)$ from state $(k-1,1)$ where $k \geq 1$.

For $k \geq 1$, transfer rates from state $(k,0)$ to $(k+1,0)$ or from $(k,1)$ to $(k+1,1)$ are due to external arrivals at a rate, $\lambda_m$.

If a station has only one job and is about to complete it, the remote stations will be polled in turn until a heavily loaded station is identified. The probability that a heavily loaded station exists and a job will be transferred is $1-\phi_m$. Thus, the transition rate from state (1,0) to state (0,1) is $\mu_m(1-\phi_m)$. Otherwise, the node will go idle at rate $\mu_m\phi_m$.

Downward transitions above state (1,0) or (0,1) in the Markov chain occur due to:

a. Service completions - rate $\mu_m$.

b. When the queue length is 2 or more, reduction of queue length through load sharing. We can estimate the rate through the system flow balance equation:

$$\sum_{j=0}^{\infty} P(j,1)\mu_{LAN} = \psi_m(1 - P(0,0) - P(1,0) - P(0,1)) \qquad (4.25)$$

Balancing equations, in the steady state, can be written by equating the rate of flow into each state to the rate of flow out of that state. The balancing equations are:

$$\begin{aligned}
(\lambda_m + \omega(1-\phi_m))P_j(0,0) &= \mu_m\phi_m P_j(1,0) \\
(\mu_m + \lambda_m)P_j(1,0) &= \lambda_m P_j(0,0) + (\mu_m + \psi_m)P_j(2,0) \\
&+ \mu_{LAN}P_j(0,1) \\
(\mu_{LAN} + \lambda_m)P_j(0.1) &= \omega(1-\phi_m)P_j(0,0) + (\mu_m + \psi_m)P_j(1,1) \\
&+ \mu_m(1-\phi_m)P_j(1,0)
\end{aligned}$$

$$
\begin{aligned}
(\mu_m + \psi_m + \lambda_m)P_j(n,0) &= \lambda_m P_j(n-1,0) + (\mu_m + \psi_m)P_j(n+1,0) \\
&+ \mu_{LAN} P_j(n-1,1) \text{ for } n \geq 2 \\
(\mu_m + \psi_m + \lambda_m + \mu_{LAN})P_j(n,1) &= \lambda_m P_j(n-1,1) + (\mu_m + \psi_m)P_j(n+1,1) \\
&\text{for } n \geq 2
\end{aligned}
$$

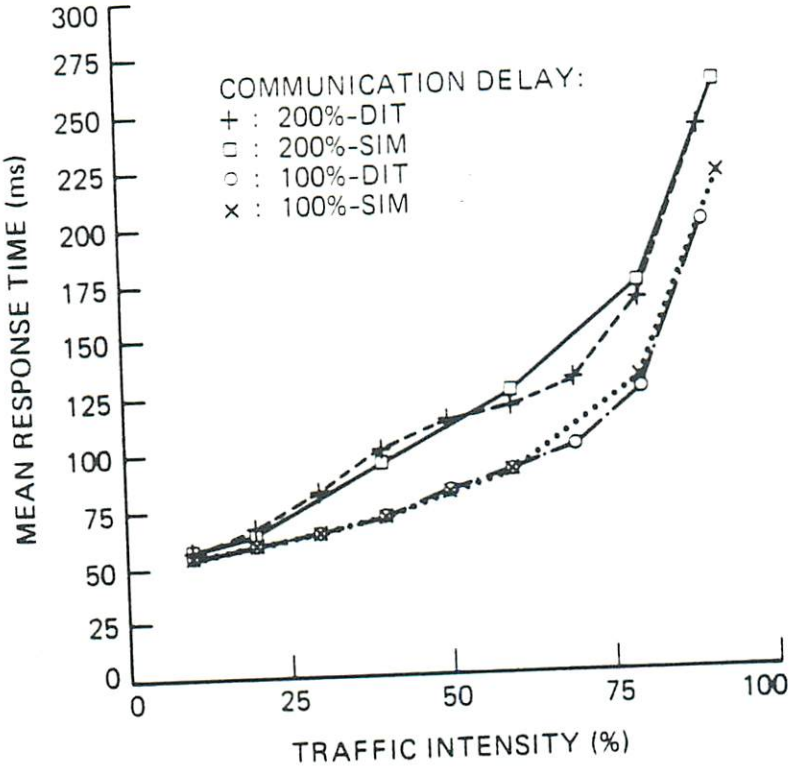
Given this model, we can immediately proceed to the iteration described in Chapter 2. The average time in system for the whole system can be obtained by Little's formula [22]:

$$
T = \frac{\sum_{m=1}^{N} \overline{N}_m}{\sum_{m=1}^{N} \lambda_m} \tag{4.26}
$$

### 4.5.3 Numerical Results of Communication Overhead Study

Figure 4.21 shows the average response times of the ten-homogeneous-station ERIDA with different network transmission times, say 10%, 90%, and 200% of the job processing time. When the network is slow, a significant delay is incurred in transferring the jobs. The slower the LAN, the longer the response time. It was also noticed that there is a kink at the low traffic arrival for large communication overhead cases. The transmission delay plays an important part in that load range. If the communication delay is significantly large, say 90% of job processing time, the ERIDA performs even worse than the M/M/1 no-load-sharing case. This observation warns that no load sharing scheme should be invoked at low load range if the communication overhead is about as large as the job processing rate.

Figure 4.21: Average response time for ERIDA with different ratios of communication delay.



# Chapter 5

# TMDA:another new resource-sharing scheme

In the previous chapter, a new load-balancing algorithm, ERIDA, was studied. The ERIDA performs very much like the receiver-initiated scheme. However, the receiver-initiated scheme is not as efficient as the sender-initiated scheme in the low-load range. In this chapter we try to combine the advantages of sender-initiated and receiver-initiated into one algorithm. Here, we propose a two-mode dynamic algorithm (TMDA) for efficient dynamic load sharing in a multiple-station distributed heterogeneous computer system. The TMDA uses the local queue length divided by the local service as the workload indicator. All station are initialized to use a sender initiated policy. When a job arrives at a station, the station evaluates the workload indicator. If the workload indicator is above a certain workload threshold, the node considers itself to be heavily loaded and attempts to transfer the job to any remote lightly loaded station. If the heavily loaded station finds that none of the remote stations is lightly loaded, it assumes that the whole system is heavily loaded and switches to the receiver-initiated mode. Thus TMDA performs

like a sender-initiated algorithm in light load and like a receiver-initiated algorithm at high load. This algorithm represents an improvement over the performance of the previously developed sender-initiated or receiver-initiated load-sharing policies in moderate loading conditions [6]. The TMDA dynamically adjusts to the load and does not generate extra overhead during high loading conditions. Therefore, it will not bring the system to an unstable state. An approximate queueing model validated by a GPSS/PC simulation is presented. These queueing and simulation models are used to evaluate the improvement in the average response time for a multiple-node system.

The TMDA fulfills all the criteria for an ideal resource-sharing algorithm: dynamic, decentralized, and heterogeneous, as described in Chapter 4. This algorithm also overcomes the shortcomings of the basic sender-initiated or receiver-initiated scheme, namely:

a. The sender-initiated scheme lacks stability at high loads. If the system is heavily loaded, all the stations are overloaded and each attempts to give away jobs, causing unproductive overhead which further saturates the overloaded system. [6]

b. The receiver-initiated scheme is efficient only for a heavily loaded system. Chang and Livny [3] also found that, when the arrival rate is low, the sender-initiated algorithm outperforms the receiver-initiated scheme in both the deadline miss ratio[1] and the mean lateness[2].

[1]The deadline miss ratio is defined as the percentage of jobs that do not meet their schedule deadlines.

[2]The mean lateness of the job is the mean difference between completion time and deadline if the job misses the deadline.

## 5.1 TMDA Algorithm Description

The TMDA uses the expected unfinished work, i.e., the local queue length divided by the local service rate at each host, as the workload indicator. In TMDA, all stations are initialized to use a sender initiated (SI) policy. When a job arrives at a station, the station evaluates the workload indicator. If the workload indicator is above a certain high workload threshold, $\vartheta_h$, the node considers itself to be heavily loaded and attempts to transfer the job to any remote lightly loaded station. If the heavily loaded station finds that none of the remote stations are lightly loaded, it assumes that the whole system is heavily loaded and switches to the receiver-initiated (RI) mode. When a job finishes service and leaves a station, the station checks the workload indicator. If the indicator is less than a certain low workload threshold, $\vartheta_l$, the lightly loaded station initiates a search for a busy station. If a station is found that has its workload indicator above a certain high threshold, a job from that busy station is transferred to the lightly loaded station.

To prevent any station from remaining idle while other stations have extra work, it is woken up periodically to search for jobs to take from other stations.

Thus, the TMDA attempts to adjust the load-balancing behavior as a function of load, resulting in an improvement in the response time and throughput performance of the total system.

The time duration between the start of a load-sharing request and the completion of the job transfer includes as a minimum: the propagation delay of the workload indicator probing; the status return from the remote station; and the transfer delay of sending a job from the source station to the destination station. During this delay, the source station and the destination station are still receiving and completing jobs. At the end of the status probing and job

transfer, the workload for each station might be quite different from what it was at the decision-making instant. If the workload of the receiving station is greater than that of the sending station at the end of the job transfer, the resource-sharing algorithm should not have transferred the job. We call this particular transfer a wrong transfer. To avoid a high probability of wrong transfers, the TMDA includes two techniques:

a. In the TMDA scheme, lightly loaded stations pull jobs from *any* heavily loaded stations rather than selecting only the most heavily loaded one as proposed in our ERIDA scheme of Chapter 4. This reduces the job-searching time significantly.

b. As soon as a source station and a destination station are committed to do a job transfer, both stations impose a 'lock' to prevent the transfer of other jobs into or out of those two stations.

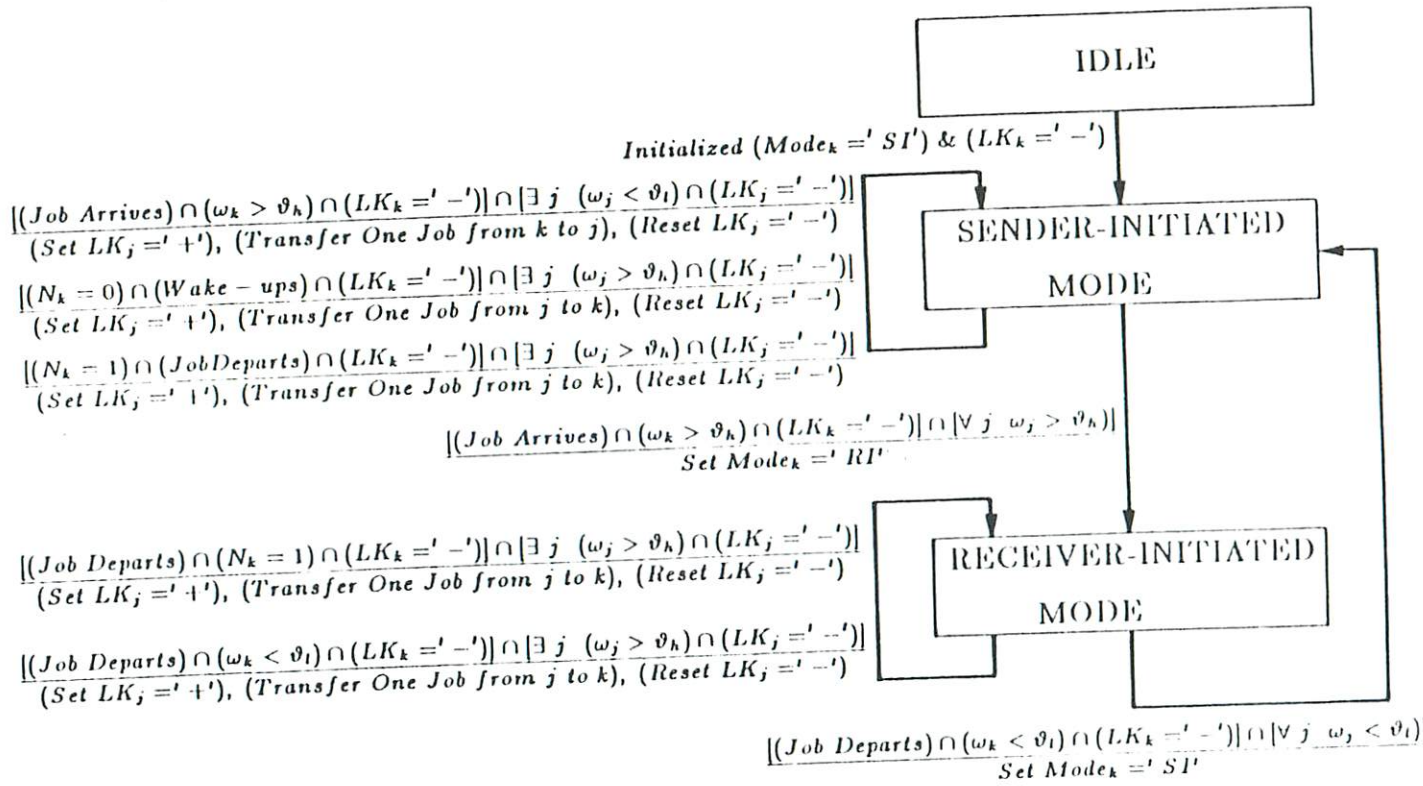
Figure 5.1 is a Yourdon state transition diagram [49] that illustrates the internal transitions in the TMDA. Each box represents a state. TMDA has three states: idle, sender-initiated mode, and receiver-initiated mode. In each of the symbolic statements, the expression above the bar shows the conditions which activate the transition. The expression below the bar shows the response taken by the TMDA for each transition. For example, if the TMDA is in SI mode and meets the condition $[(job\ arrives) \cap (\omega_k > \vartheta_h) \cap (LK_k = \text{`}-\text{'})] \cap [\forall\ j\ \ \omega_j > \vartheta_h)]$, the TMDA will take the action $[Set\ Mode_k = \text{`}RI\text{'}]$ and will switch from SI mode to RI mode. Thus the condition and action statement

$$\frac{[(Job\ Arrives) \cap (\omega_k > \vartheta_h) \cap (LK_k = \text{`}-\text{'})] \cap [\forall\ j\ \ \omega_j > \vartheta_h)]}{Set\ Mode_k = \text{`}RI\text{'}} \qquad (5.1)$$

appears beside the transition arrow from 'SI' to 'RI'.

The following is a symbolic description of the TMDA:

Figure 5.1: State transition diagram for the TMDA.



The Two-Mode Dynamic Algorithm (TMDA)

Given $N$ : total number of stations in the heterogeneous system;
$\mu_k$ : service rate of station $k$;
$N_k$ : queue length at station $k$;
$\omega_k$ : workload indicator of station $k$, $\omega_k = \frac{N_k}{\mu_k}$;
$Mode_k$ : resource-sharing mode of station $k$,
$Mode_k \in$ {SI (sender-initiated), RI (receiver-initiated)};
$k$ : station ID, $k \in \{1, \ldots, N\}$;
$\vartheta_h$ : high workload threshold; $\vartheta_h > 0$
$\vartheta_l$ : low workload threshold, $\vartheta_l > 0$ and $\vartheta_h \geq \vartheta_l$.
$LK_j$ : station status available indicator, $LK_j \in \{'+','-'\}$;
$LK_j$='+' means station $j$ is locked, i.e., committed to a job transfer, and its workload indicator is not available for probing;
$LK_j$='-' means the workload indicator of station $j$ is unlocked, i.e., available for probing.
$j$ : For a given local station $k$, $j$ represents all remote stations, i.e., $j = 1, \ldots, N, j \neq k$.

Initialize ($Mode_k$='SI') and ($LK_k$='-'), for $k = \{1, \ldots, N\}$;

while ($Mode_k$='SI') $\cap$ (job arrives) $\cap$ ($\omega_k > \vartheta_h$) $\cap$ ($LK_k$='-')
do
  $\exists j$ so that ($LK_j$='-') $\cap$ ($\omega_j < \vartheta_l$), then
  do
    set $LK_j$='+';
    transfer 1 job from end of queue in station $k$ to station $j$;
    reset $LK_j$='-';
    job is processed at station $j$ and the results are returned to station $k$;
    exit;
  end;
  else
  if $\omega_j > \vartheta_h$ for $\forall j$, then
    set $Mode_k$='RI';
end;

while ($Mode_k$='RI') $\cap$($\omega_k < \vartheta_l$) $\cap$ (job departs) $\cap$ ($LK_k$='-')
do

$\exists j$ so that $(LK_j$='-') $\cap$ $(\omega_j > \vartheta_h)$, then
do
set $LK_j$='+';
transfer 1 job from end of queue in station $j$ to station $k$;
reset $LK_j$='-';
job is processed at station $k$ and the results are returned to station $j$;
exit;
end;
else
if $\omega_j < \vartheta_l$ for $\forall j$, then
set $Mode_k$='SI';
end.

while $(N_k = 1) \cap$ (job departs) $\cap$ $(LK_k$='-')
do
$\exists j$ so that $(LK_j$='-') $\cap$ $(\omega_j > \vartheta_h)$, then
do
set $LK_j$='+';
transfer 1 job from end of queue in station $j$ to station $k$;
reset $LK_j$='-';
job is processed at station $k$ and the results are returned to station $j$;
exit;
end;
end;

while $(N_k = 0) \cap$ (wakeups) $\cap$ $(LK_k$='-')
do
$\exists j$ so that $(LK_j$='-') $\cap$ $(\omega_j > \vartheta_h)$, then
do
set $LK_j$='+';
transfer 1 job from end of queue in station $j$ to station $k$;
reset $LK_j$='-';
job is processed at station $k$ and the results are returned to station $j$;
exit;
end;
end.

## 5.2 Analytical Performance Model

Evaluation of the TMDA performance is a typical state-dependent interactive queue problem. This problem is solved by the DIT approximate queueing technique described in Chapter 3. In this section we show how the approximate analytical model can be used to find the average response time of a distributed system with the TMDA load-balancing scheme.

Let us review the main assumptions:

a. There is only one class of tasks. New tasks arrive at node $i$ at a rate $\lambda_i$, and the service rate at node $i$ is $\mu_i$. Both the arrival and service times are exponentially distributed.

b. Each job needs only one resource (i.e., the processor).

c. For simplicity, the queue length rather than the expected unfinished work is used as the workload indicator.

d. The network transmission delay in propagating requests, probing the status, and returning results is negligible. This assumption is valid if the transmission bandwidth is large compared to the traffic on the network.

e. All processing overhead for probing, packing and unpacking data, request, and result transfer is ignored. This assumption is valid if the processing required to pack and unpack the job is significantly less than the processing required to process the job.

Overhead was not considered in the TMDA analytical model because of the additional complexity. Later in this chapter, the overhead due to communication delay and packing/unpacking processing is investigated by simulation.

An exact analytic solution of dynamic load-sharing algorithms such as TMDA with multiple interaction queues is quite complex, since an $N$-dimensional

Markov chain is needed to show the interactions between queues. An exact solution for such systems with more than two stations becomes intractable. Instead of describing the number of jobs of all the stations in one state representation, our DIT approximation model concentrates on the behavior of one queue at a time. It attempts to express the interaction in the state transition rates of the Markov chain of the queue in each station. This approach characterizes the interaction between the various queues in terms of their steady-state occupancy probabilities. Iteration is then used to update estimates of the interaction.

The Markov chains for TMDA with different thresholds are different. The derivations of TMDA performance with various threshold combinations are very similar and are presented in the following sections.

### 5.2.1 TMDA State Transition Rates

In this section, we present the analytical model for a homogeneous system. Extension to the heterogeneous case is straightforward but tedious. The state of the system is defined by the ordered 2-tuple $(n_m, \gamma_m)$, where $n_m$ denotes the number of jobs in server $m$, and $\gamma_m \in \{S, R\}$ denotes that station $m$ is in sender-initiated or receiver-initiated mode.

We also define the following notation:

$P_m(l, \gamma)$ = estimate of the steady-state probability that the queue length at station $m$ is equal to $l$ $(m = 1, \ldots, N)$ and the mode is $\gamma \in \{R, S\}$.

$\omega$=average wakeup rate for each idle station.

$\mu_m$=service rate at station $m$ (can be extended to state dependent servers).

$\vartheta_h$ =high workload threshold.

$\vartheta_l$ =low workload threshold.

$\theta_{h,m}$ =high queue-length threshold for station $m$, $\theta_{h,m} = \vartheta_h \ \mu_m$.

$\theta_{l,m}$ =low queue-length threshold for station $m$, $\theta_{l,m} = \vartheta_l \ \mu_m$.

$N_m$=queue size at station $m$.

$\overline{N}_m$ =average number of customers (queue length) at station $m$.

$T$ =average system response time.

Upward Transitions

Upward transitions can be split into 5 regions:

A. Idle:

If station $m$ is idle, the arrival rate which we will call $\delta_m$ is

$$\begin{aligned} \delta_m &\equiv \lambda_m + \frac{\sum_{j=1 \ j\neq m}^{N} P_j(\theta_{h,j}, S)\lambda_j}{1 + N_{light,m}} \\ &+ (1 - P_{light,m})\omega \end{aligned} \tag{5.2}$$

$$\text{where } P_{light,m} = \prod_{j=1 \ j\neq m}^{N} [\sum_{i=0}^{\theta_{h,j}} \sum_{\gamma=S,R} P_j(i,\gamma)] \tag{5.3}$$

$$\text{and } N_{light,m} = \sum_{j=1 \ j\neq m}^{N} [\sum_{i=0}^{\theta_{l,j}-1} \sum_{\gamma=S,R} P_j(i,\gamma)] \tag{5.4}$$

In Equation 5.2, $\lambda_m$ corresponds to external arrivals at station $m$. $\sum_{j=1 \ j\neq m}^{N} P_j(\theta_{h,j}, S)\lambda_j$ is the total arrival rate due to overloaded SI stations. These excess jobs will be distributed over station $m$ and any other underloaded stations. $N_{light,m}$ is the mean number of stations other than

$m$ that are underloaded. $(1 - P_{light,m})\omega$ corresponds to job transfers occurring at the wakeup rate $\omega$ while system $m$ is idle, provided some other station is heavily loaded, i.e., with probability $1 - P_{light,m}$, where $P_{light,m}$ is the probability that all stations except $m$ are lightly loaded.

B. $0 < N_m < (\theta_{l,m} - 1)$ and in SI:

For queue length $0 < N_m < (\theta_{l,m} - 1)$, the arrival rate is the same as in Equation 5.2 except that there is no need for the periodic wakeup term. We call this term $\eta_m$:

$$\eta_m \equiv \lambda_m + \frac{\sum_{j=1\ j\neq m}^{N} P_j(\theta_{h,j}, S)\lambda_j}{1 + N_{light,m}} \quad \text{for } 0 < N_m < (\theta_{l,m} - 1) \tag{5.5}$$

$$\text{where } N_{light,m} = \sum_{j=1\ j\neq m}^{N} [\sum_{i=0}^{\theta_{l,j}-1} \sum_{\gamma=S,R} P_j(i,\gamma)] \tag{5.6}$$

C. $N_m = \theta_{h,m}$ and in SI:

External jobs will not be accepted at the overloaded sender-initiated station unless all of the other stations in the system are also overloaded. Therefore, the arrival rate to this state, $\tau_m$, is

$$\tau_m \equiv \lambda_m P_{heavy,m} \tag{5.7}$$

$$\text{where } P_{heavy,m} = \prod_{j=1\ j\neq m}^{N} [\sum_{i=\theta_{l,j}}^{\infty} \sum_{\gamma=S,R} P_j(i,\gamma)] \tag{5.8}$$

In our TMDA scheme, the node switches from sender-initiated mode to receiver-initiated mode if it is above the high threshold and all the remote stations are also above the high threshold, i.e., $\tau_m$ is the rate that the node switches from the SI mode to the RI mode.

D. $\theta_{l,m} \geq N_m > \theta_{l,m}$ and in SI:

In this range, all external arrivals are locally queued and the upward state transitions are given by $\lambda_m$.

E. $N_m \geq \theta_{l,m}$ and in RI:

Once the node is in RI mode, all external arrivals are locally queued and the upward state transitions are given by $\lambda_m$.

Downward Transitions

The downward transitions are considered in 5 regions:

A. $N_m = 1$ and in SI mode:

After serving the last job, station $m$ will depart from state $(1, S)$ to state $(0, S)$ if all of other stations in the system are lightly loaded. The downward transition rate, $\xi_m$, is therefore

$$\xi_m \equiv \mu_m P_{light,m} \tag{5.9}$$

$$\text{where } P_{light,m} = \prod_{j=1\ j\neq m}^{N} [\sum_{i=0}^{\theta_{h,j}} \sum_{\gamma=S,R} P_j(i,\gamma)] \tag{5.10}$$

B. $1 < N_m \leq \theta_{h,m}$ and in SI mode:

For $1 < N_m \leq \theta_{h,m}$ in the sender-initiated mode, the downward transition rate is simply $\mu_m$.

C. $N_m = \theta_{l,m}$ and in RI mode:

For a receiver-initiated station at the low threshold, the station will not depart from this state after serving a job unless none of the remote stations is overloaded (otherwise it picks up one of their jobs). The

downward transition rate, $\xi_m$, is same as in region A.

$$\xi_m \equiv \mu_m P_{light,m} \tag{5.11}$$

$$\text{where } P_{light,m} = \prod_{j=1\ j\neq m}^{N} [\sum_{i=0\ \gamma=S,R}^{\theta_{h,j}} P_j(i,\gamma)] \tag{5.12}$$

The TMDA scheme will switch from receiver-initiated mode to sender-initiated mode if the local receiver-initiated station $m$ is at the low threshold and all the remote stations are also under the low threshold. Therefore, $\xi_m$ is also the transfer rate from the RI mode to the SI mode.

D. $\theta_{l,m} < N_m \leq \theta_{h,m}$ and in RI mode:

For $\theta_{l,m} < N_m \leq \theta_{h,m}$ in either mode, the downward transition rate is also $\mu_m$ since it is still under high threshold.

E. $N_m > \theta_{h,m}$ and in RI mode

All the stations above the high threshold gain some extra load-balancing processing power contributed by receiver-initiated stations that are at low threshold and idle stations. Therefore, while $N_m > \theta_{h,m}$, downward transitions in the Markov chain occur as a result of:

a. Service completions at a rate $\mu_m$.

b. Load sharing by remote lightly loaded stations.

c. Periodical polling by the remote idle stations.

The increase in service rate provided by items b and c can be evaluated by the following system flow balance equation:

$$\left( \frac{\sum_{j=1\ j\neq m}^{N} P_j(\theta_{h,j}, S)\lambda_j}{1 + N_{light,m}} + (1 - P_{light,m})\omega \right) P_j(0, S)$$

$$+ \left( \frac{\sum_{j=1\ j\neq m}^{N} P_j(\theta_{h,j}, S)\lambda_j}{1 + N_{light,m}} \right) \sum_{i=1}^{\theta_{h,j}-1} P_j(i, S) + (1 - P_{light,m})\mu_m P_{l,R}$$

$$= (1 - P_{heavy,m})\lambda_m P_{h,S} + \varphi_m \sum_{i=\theta_{h,m}+1}^{\infty} P_j(i, R) \quad (5.13)$$

$$\text{where } P_{light,m} = \prod_{j=1\ j\neq m}^{N} [\sum_{i=0\ \gamma=S,R}^{\theta_{h,j}} P_j(i, \gamma)]$$

$$\text{and } N_{light,m} = \sum_{j=1\ j\neq m}^{N} [\sum_{i=0\ \gamma=S,R}^{\theta_{l,j}-1} P_j(i, \gamma)]$$

In this system flow balance equation, Equation 5.13 the left-hand side of the equation represents the total load-balancing incoming flow from the remote stations to the local station $m$, and the right-hand side of the equation represents the total load-balancing outgoing flow from the local station $m$ to the remote stations.

Therefore, the service rate for those receiver-initiated heavily loaded stations, $\psi_m$, is

$$\psi_m \equiv \mu_m + \varphi_m \quad (5.14)$$

Steady state balance equations can be written equating the rate of flow into a state to the rate of flow out of that state. The steady-state probabilities for state $(N_m, R)$, with $N_m \geq (\theta_{h,m} + 2)$, can be expressed in terms of state $(\theta_{h,m} + 1, R)$ explicitly:

$$P_{n,R} = (\frac{\lambda_m}{\psi_m})^{n-(\theta_{h,m}+1)} P_{\theta_{h,m}+1,R} \quad \text{for } n \geq (\theta_{h,m} + 2) \quad (5.15)$$

The global balance equations for the rest of the states are different with different threshold combinations. Global balance equations for four different threshold combinations are presented in the following four sections:

### 5.2.2 TMDA with $\theta_{h,m} = 4$ and $\theta_{l,m} = 3$

The global balance equations for the states in the trapezoidal arrangement in Figure 5.2 can be represented by their neighbor states and summarized into the following seven equations:

$$\delta_m P_{0,S} = \xi_m P_{1,S} \tag{5.16}$$

$$(\xi_m + \eta_m) P_{1,S} = \delta_m P_{0,S} + \mu_m P_{2,S} \tag{5.17}$$

$$(\mu_m + \eta_m) P_{2,S} = \eta_m P_{1,S} + \xi_m P_{3,R} + \mu_m P_{3,S} \tag{5.18}$$

$$(\mu_m + \lambda_m) P_{3,S} = \eta_m P_{2,S} + \mu_m P_{4,S} \tag{5.19}$$

$$(\mu_m + \tau_m) P_{4,S} = \lambda_m P_{3,S} \tag{5.20}$$

$$(\xi_m + \lambda_m) P_{3,R} = \mu_m P_{4,R} \tag{5.21}$$

$$(\mu_m + \lambda_m) P_{4,R} = \lambda_m P_{3,R} + \psi_m P_{5,R} \tag{5.22}$$

### 5.2.3 TMDA with $\theta_{h,m} = 3$ and $\theta_{l,m} = 2$

The Markov chain for $\theta_{h,m} = 3$ and $\theta_{l,m} = 2$ is shown in Figure 5.3. The balance equations for the states in the trapezoidal arrangement are:

$$\delta_m P_{0,S} = \mu_m P_{light,m} P_{1,S} \tag{5.23}$$

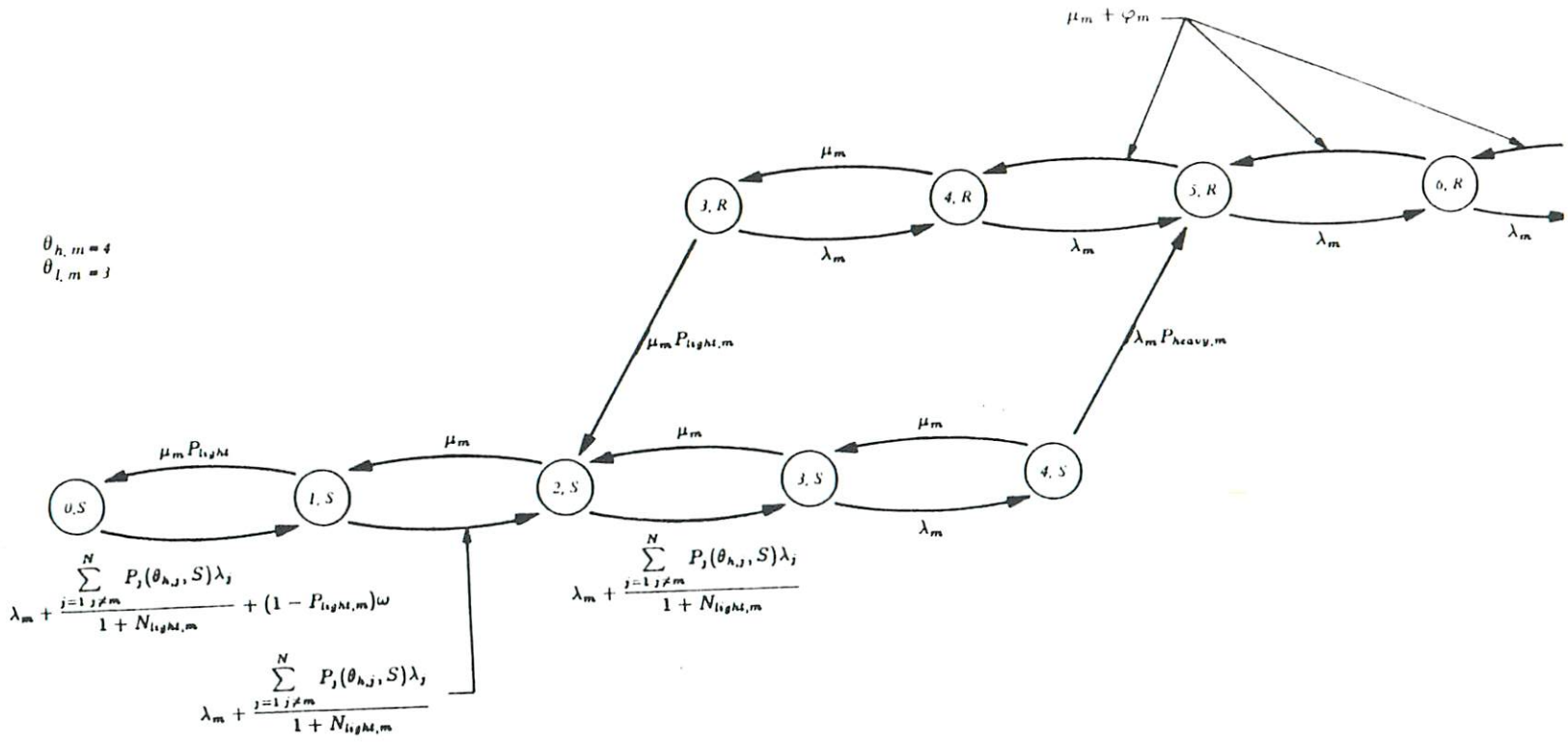
$$(\mu_m P_{light} + \eta_m) P_{1,S} = \delta_m P_{0,S} + \mu_m P_{light} P_{2,R} + \mu_m P_{2,S} \tag{5.24}$$

$$(\mu_m + \lambda_m) P_{2,S} = \eta_m P_{1,S} + \mu_m P_{3,S} \tag{5.25}$$

$$(\mu_m + \lambda_m P_{heavy}) P_{3,S} = \lambda_m P_{2,S} \tag{5.26}$$

$$(\mu_m P_{light} + \lambda_m) P_{2,R} = \mu_m P_{3,R} \tag{5.27}$$

Figure 5.2: Markov model for the TMDA with $\theta_{h,m} = 4, \theta_{l,m} = 3$.



$$(\mu_m + \lambda_m)P_{3,R} = \mu_m P_{2,R} + \psi_m P_{4,R} \quad (5.28)$$

### 5.2.4 TMDA with $\theta_{h,m} = 2$ and $\theta_{l,m} = 1$

The Markov chain for $\theta_{h,m} = 2$ and $\theta_{l,m} = 1$ is shown in Figure 5.4. The balance equations for the states in the trapezoidal arrangement are:

$$\delta_m P_{0,S} = \mu_m P_{light,m} P_{1,S} + \mu_m P_{light,m} P_{1,R} \quad (5.29)$$

$$(\mu_m P_{light,m} + \eta_m)P_{1,S} = \delta_m P_{0,S} + \mu_m P_{2,S} \quad (5.30)$$

$$(\mu_m + \lambda_m P_{heavy,m})P_{2,S} = \eta_m P_{1,S} \quad (5.31)$$

$$(\mu_m P_{light,m} + \lambda_m)P_{1,R} = \mu_m P_{2,R} \quad (5.32)$$

$$(\mu_m + \lambda_m)P_{2,R} = \lambda_m P_{1,R} + \psi_m P_{3,R} \quad (5.33)$$

### 5.2.5 TMDA with $\theta_{h,m} = 1$ and $\theta_{l,m} = 1$

The Markov chain for $\theta_{h,m} = 1$ and $\theta_{l,m} = 1$ is shown in Figure 5.5. The balance equations for the states in the trapezoidal arrangement are:

$$\delta_m P_{0,S} = \mu_m P_{light,m} P_{1,S} + \mu_m P_{light} P_{1,R}$$

$$(\mu_m P_{light} + \lambda_m P_{heavy})P_{1,S} = \delta_m P_{0,S}$$

$$(\mu_m P_{light} + \lambda_m)P_{1,R} = \psi_m P_{2,R}$$

These equations and the boundary conditions can be used to find the steady state probabilities, $\Pi$.

The average number of customers at each station can be obtained by

$$\overline{N_m} = \sum_{n=1}^{\theta_{h,m}} n P_{n,S} + \sum_{n=\theta_{l,m}}^{\infty} n P_{n,R} \quad (5.34)$$

Figure 5.3: Markov model for the TMDA with $\theta_{h,m} = 3, \theta_{l,m} = 2$.

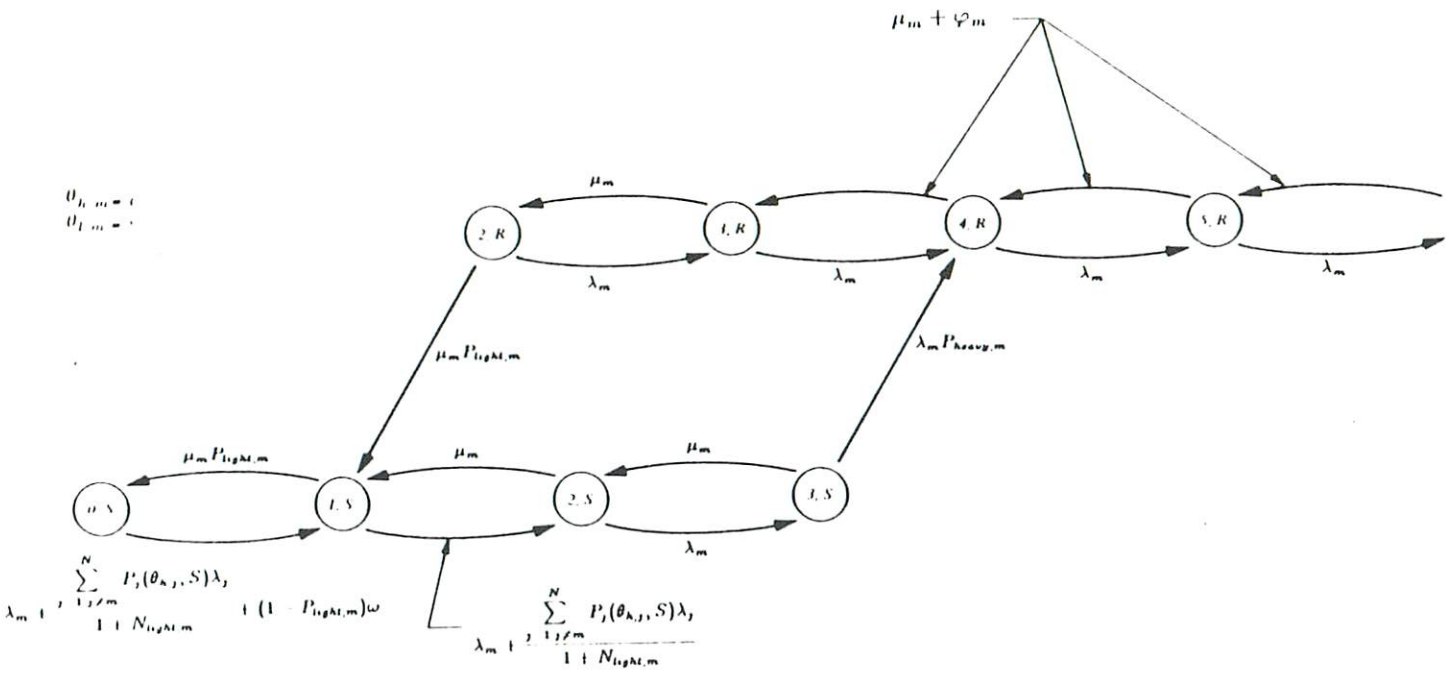


Figure 5.4: Markov model for the TMDA with $\theta_{h,m} = 2, \theta_{l,m} = 1$.

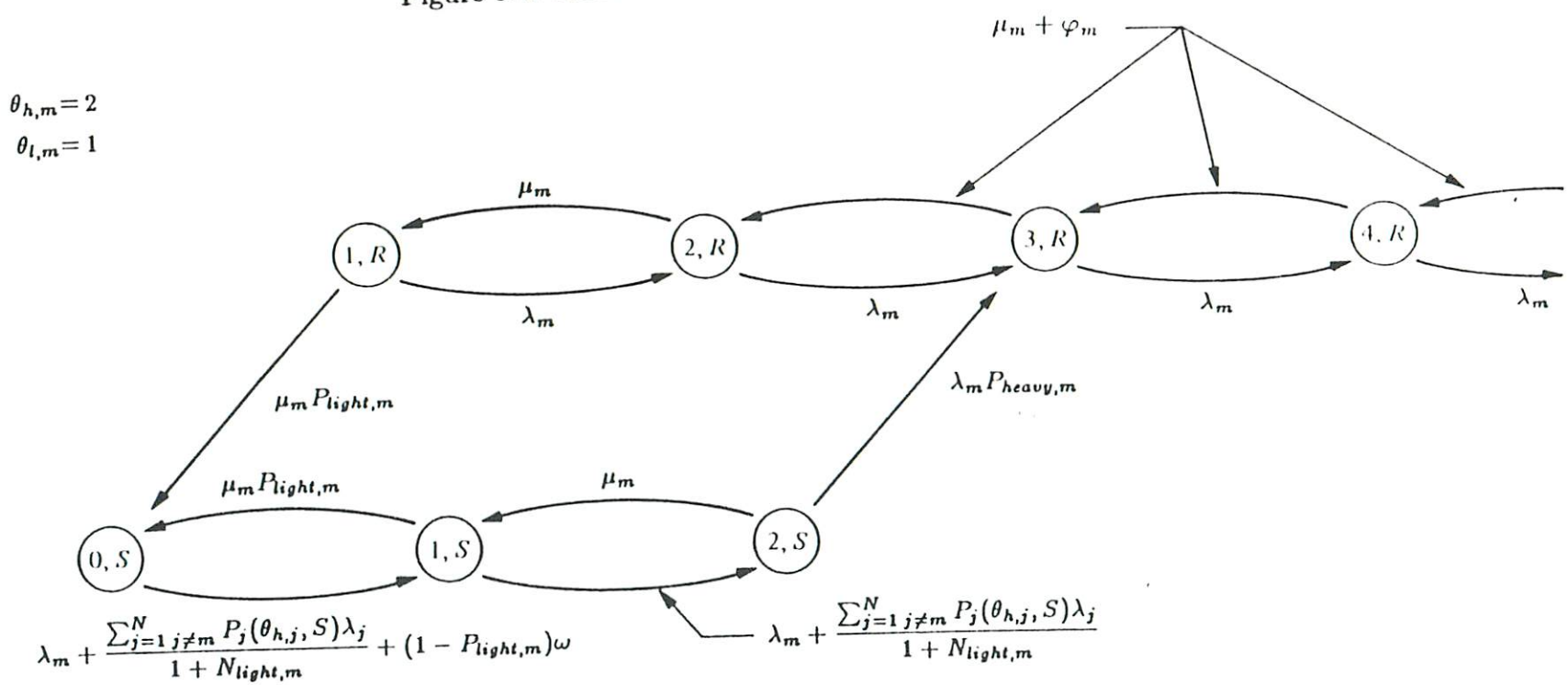
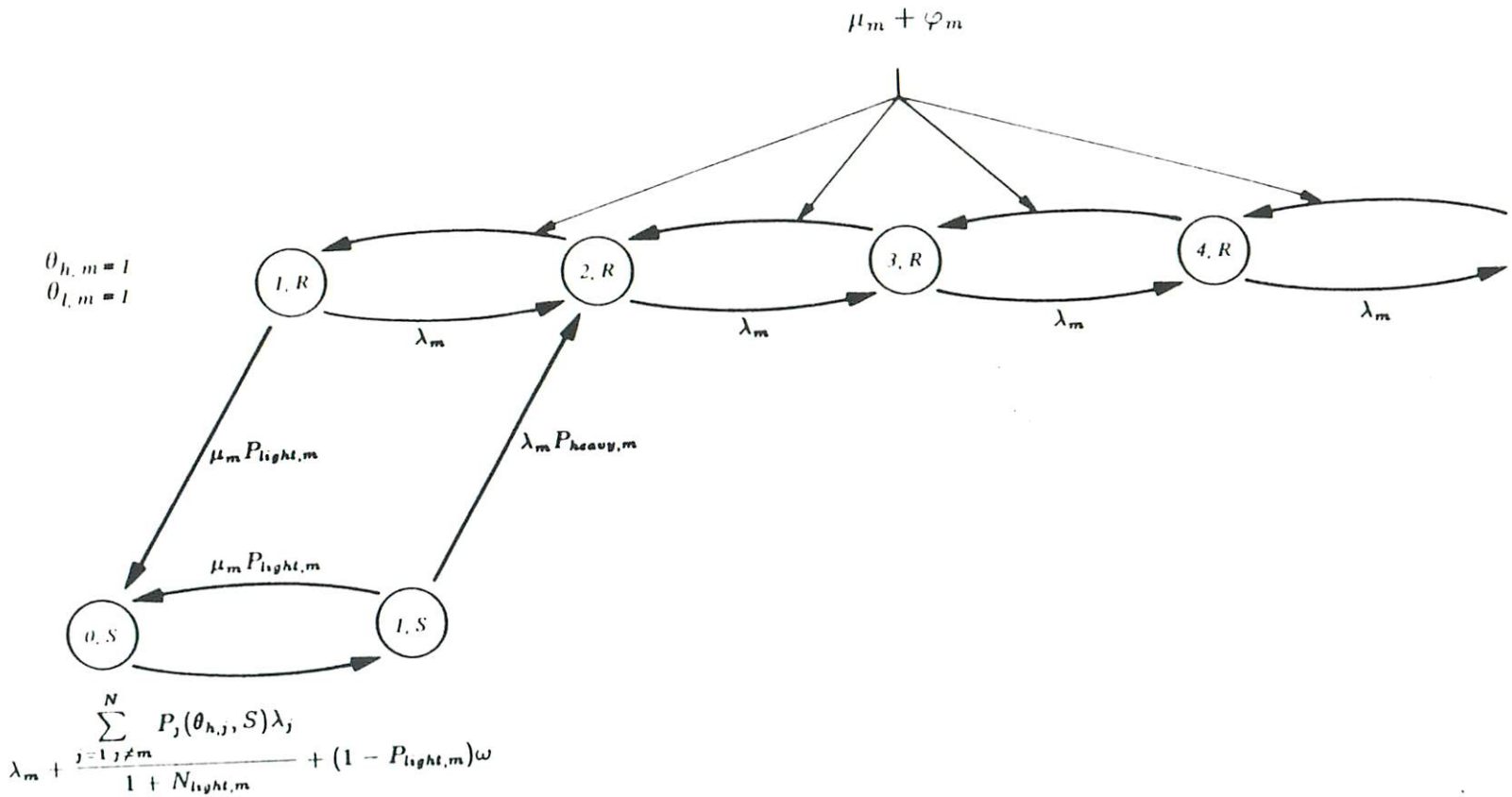


Figure 5.5: Markov model for the TMDA with $\theta_{h,m} = 1, \theta_{l,m} = 1$.



For example, the average number of customers at each station for TMDA with $\theta_{h,m} = 4$ and $\theta_{l,m} = 3$ is:

$$\begin{aligned}\overline{N_m} &= P_{1,S} + 2P_{2,S} + 3(P_{3,S} + P_{3,R}) + 4(P_{4,S} + P_{4,R}) \\ &+ P_{5,R}(\frac{\psi_m}{\lambda_m})^5[\frac{\frac{\lambda_m}{\psi_m}}{(1-\frac{\lambda_m}{\psi_m})^2} - \frac{\lambda_m}{\psi_m} - 2(\frac{\lambda_m}{\psi_m})^2 - 3(\frac{\lambda_m}{\psi_m})^3 - 4(\frac{\lambda_m}{\psi_m})^4]\end{aligned}$$

We see that the transition rates, and hence the steady-state solution, for node $m$ depend on the steady-state probabilities of the other queues in the system.

Given this model we can immediately proceed to the iteration described in [29]. The iteration procedure is used to generate $P_m(l, \gamma)$ for each station. Hence, the average response time for the whole system can be obtained by Little's formula [27]:

$$T = \frac{\sum_{m=1}^{N} \overline{N}_m}{\sum_{m=1}^{N} \lambda_m} \tag{5.35}$$

## 5.3 Numerical Results

### 5.3.1 TMDA Performance: No-Overhead Case

In this section we present results from the approximate model described in Section 5.2 and a comparison with simulation. We also compare the TMDA result to no load sharing and to a centralized queuing system.

The system configuration considered here is a homogeneous ten-node system. Each service station has the same service rate, 5 MIPS. The mean size of each job is 240,000 instructions. Therefore, the mean service time for each job is 48 ms. The arrival rates at each station are also assumed to be equal.

### Comparison Between Analytical Performance Model and GPSS/PC Simulation

The approximate TMDA queueing model is validated by a GPSS/PC simulation [19]. Each data point was run a number of times with different random number seeds until a 95% confidence interval on the mean was less than 15% of the mean itself. The comparison of results from the average response time generated by the approximate queuing model and the results from the simulation exercise are shown in Figures 5.6, 5.7, 5.8, and 5.9 for ($\theta_{h,m} = 4,\ \theta_{l,m} = 3$), ($\theta_{h,m} = 3,\ \theta_{l,m} = 2$), ($\theta_{h,m} = 2,\ \theta_{l,m} = 1$), and ($\theta_{h,m} = 1,\ \theta_{l,m} = 1$), respectively. We see that our approximate queueing model agrees very closely with simulation.

### Threshold Selection

In this section we try to find the optimal queue length threshold for the TMDA.

First of all, the low threshold must be greater than zero. If $\theta_{l,m} = 0$, the 'RI' transfer will never be activated after heavy traffic pushes the stations from 'SI' mode to 'RI' mode. Second, the high threshold must be at least as high as the low threshold. Otherwise, a 'ping-pong' problem will be encountered, as illustrated in Figure 5.10 [31].

Figure 5.11 compares the average delay performance of the TMDA with different thresholds. The result shows that the higher the threshold the less the improvement in delay performance. This is because the lower the threshold the more frequently the TMDA scheme will be invoked to perform load balancing. If the threshold is set too high, the TMDA will not be activated even when the load among the stations is imbalanced. This phenomenon can be illustrated by the job probing ratio and job transfer ratio, discussed in the next section.

For the TMDA to operate efficiently, it is important to use the *workload*

Figure 5.6: Average response time comparison between TMDA DIT analysis and TMDA GPSS/PC simulation in a ten-server homogeneous system. $\theta_{h,m} = 4, \theta_{l,m} = 3$. No overhead is considered.

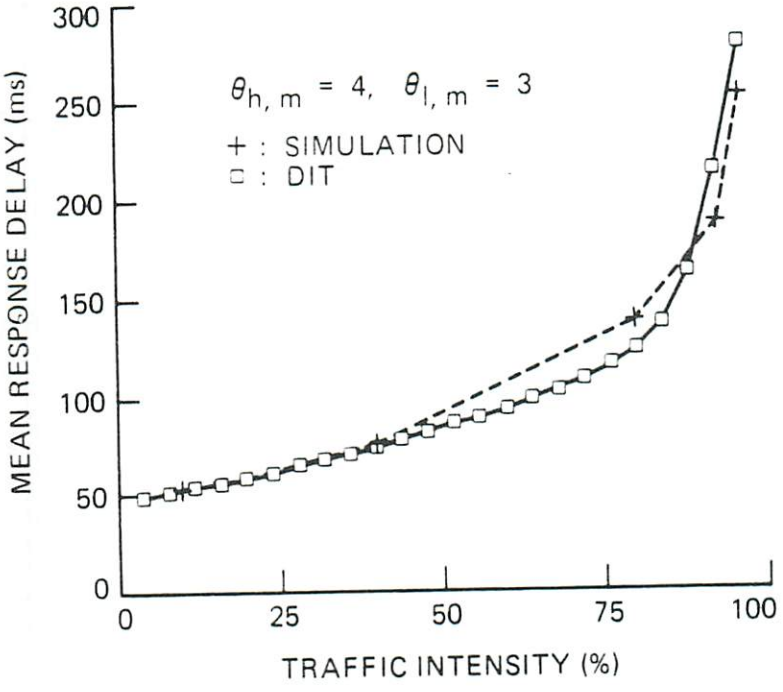


Figure 5.7: Average response time comparison between TMDA DIT analysis and simulation in a ten-server homogeneous system. $\theta_{h,m} = 3, \theta_{l,m} = 2$. No overhead is considered.

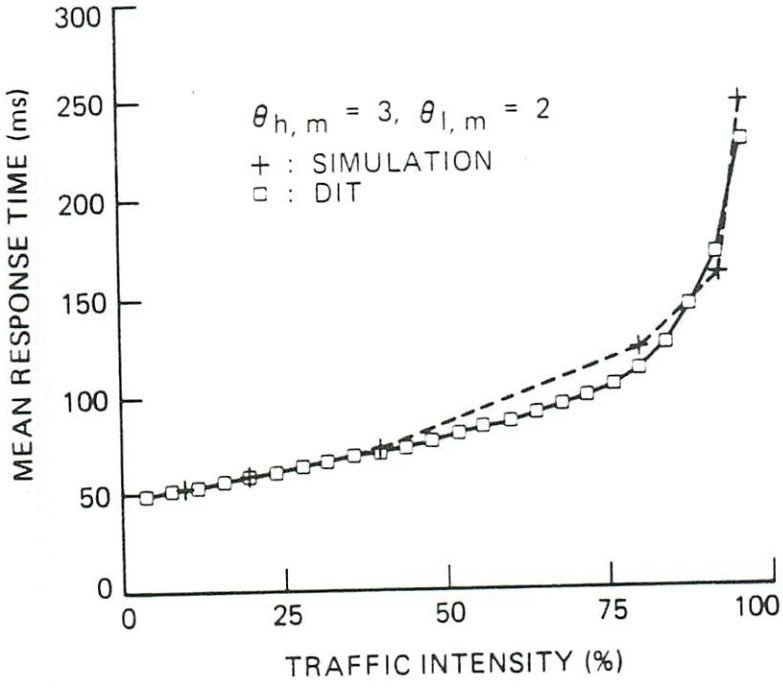


Figure 5.8: Average response time comparison between TMDA DIT analysis and simulation in a ten-server homogeneous system. $\theta_{h,m} = 2, \theta_{l,m} = 1$. No overhead is considered.

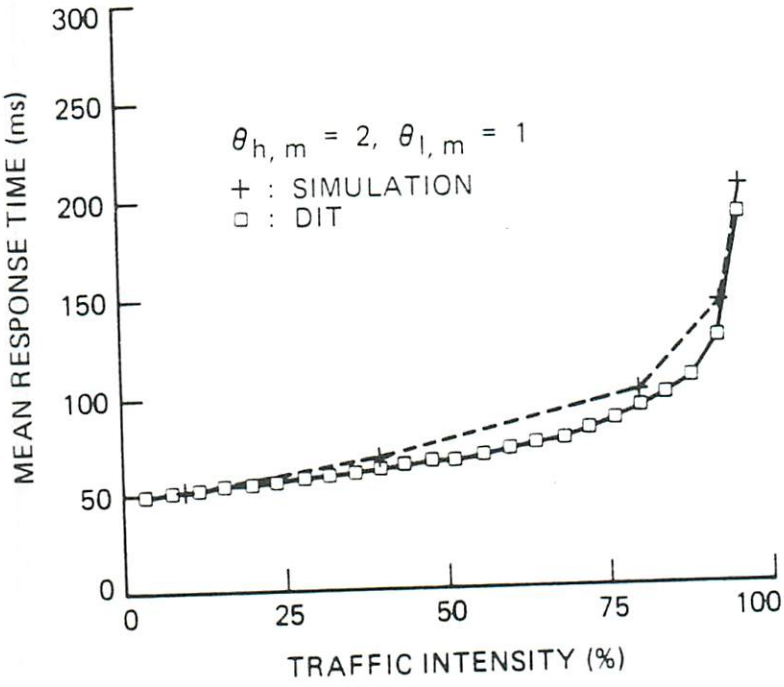


Figure 5.9: Average response time comparison between TMDA DIT analysis and simulation in a ten-server homogeneous system. $\theta_{h,m} = 1, \theta_{l,m} = 1$. No overhead is considered.

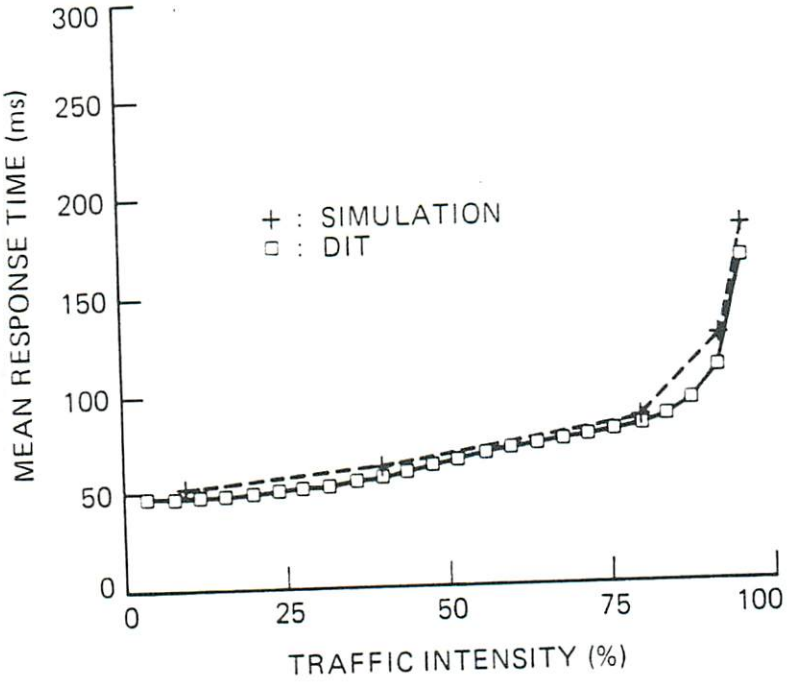


Figure 5.10: TMDA ping-pong problem caused by setting $\theta_{l,m} > \theta_{h,m}$.

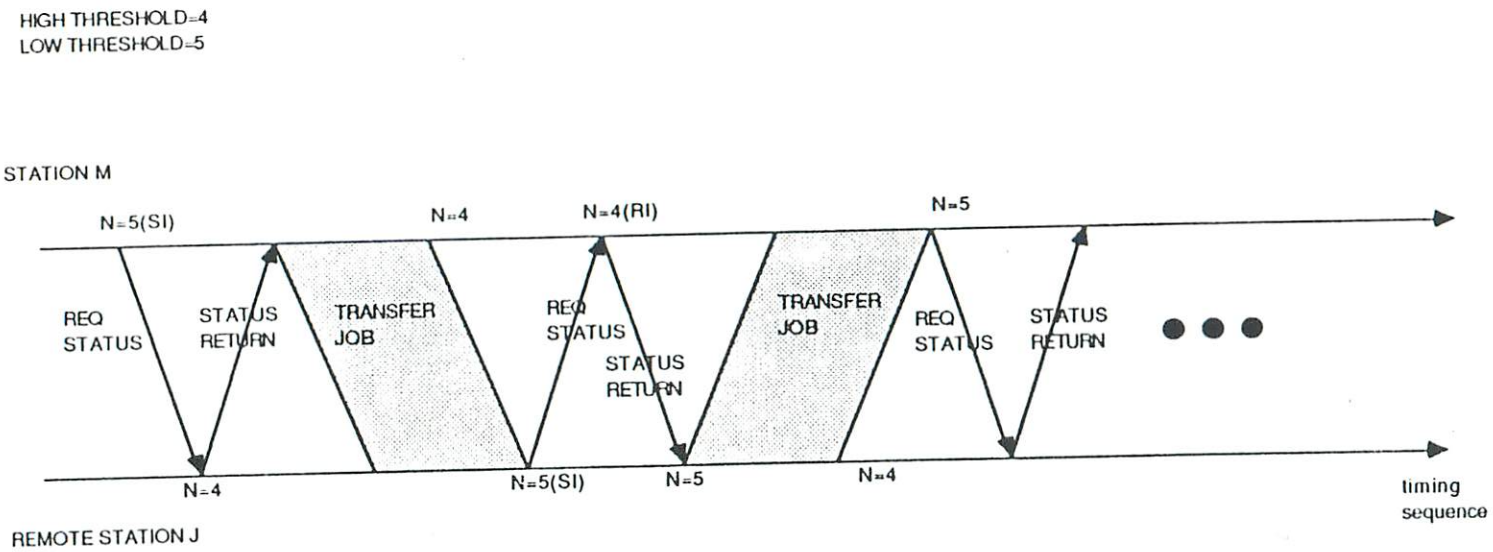
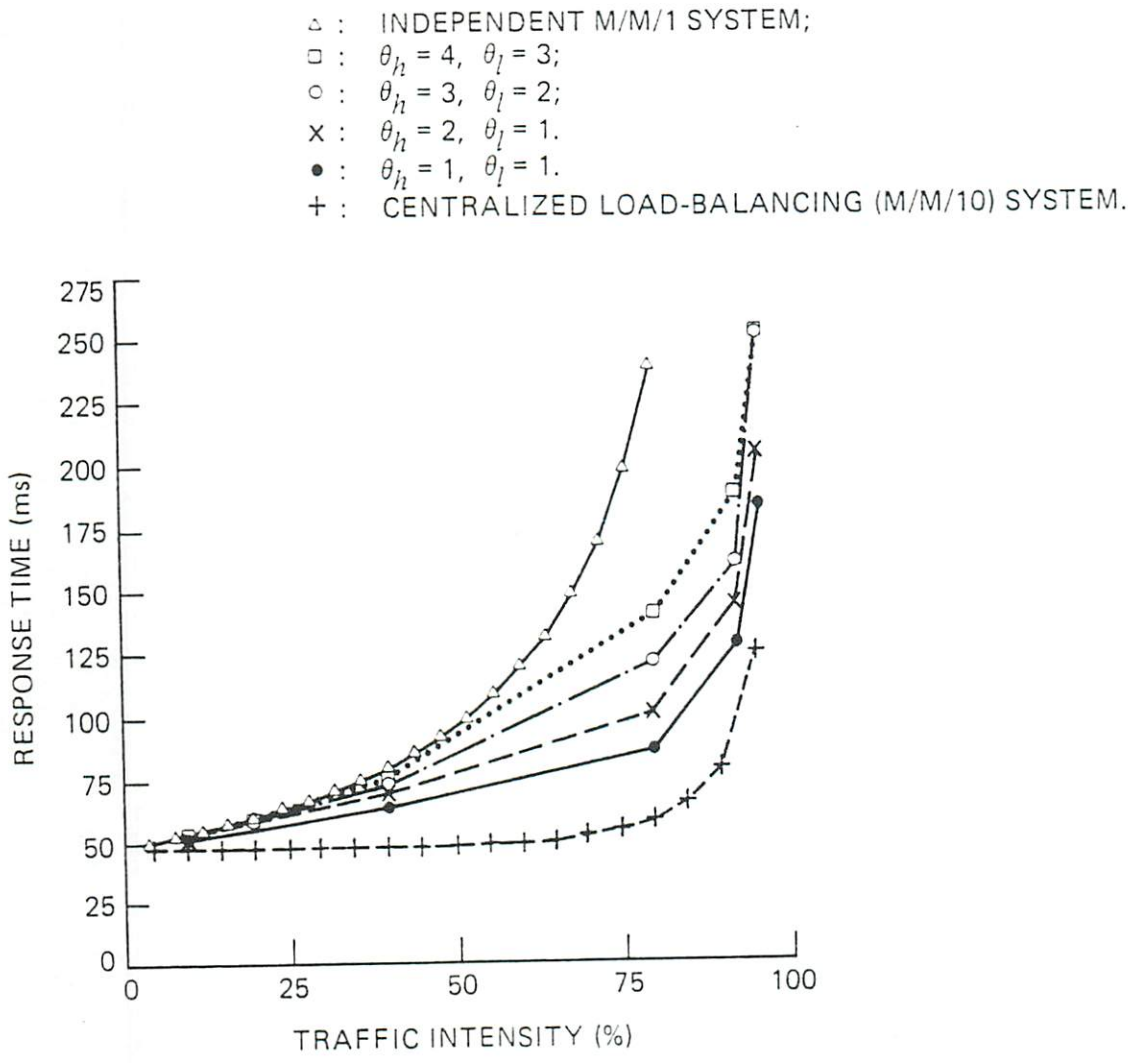


Figure 5.11: Average response times of TMDA GPSS/PC simulation with four different threshold settings, compared with centralized load balancing (M/M/10) and an independent (M/M/1) schemes in a ten-server homogeneous system. The arrival rate is evenly distributed. No overhead is considered.



instead of the *queue length* as the loading indicator in a heterogeneous computer system. Figure 5.12 demonstrates how that choice affects the response time performance. It is assumed that the service rate of station A is twice that of station B. Using the queue length as the loading indicator will initiate a job transfer from the fast station to the slow station, resulting in a slow completion time for station B. Using the workload (queue length divided by service rate) will prevent this bad transfer.

From these results it can be concluded that a reasonable strategy for selecting the queue-length threshold for low-overhead TMDA is to define both high and low queue-length thresholds to be the floor of the local service time divided by the minimum service rate in the system. The algorithm to select the threshold can be represented as follows:

Given $N$ : total number of stations in the heterogeneous system;
$\mu_k$ : service rate of station $k$;
$k$ : station ID, $k \in \{1, \ldots, N\}$;

begin

$$\mu_i = MIN_{k=1\ldots,N}\{\mu_k\}$$
$$\forall k, \quad \theta_{h,k} = \theta_{l,k} = \lfloor \frac{\mu_k}{\mu_i} \rfloor$$

end.

In Figure 5.11 the results of the TMDA are also compared against the case without load sharing, i.e., the case of multiple independent M/M/1 queues. We find that the average response time is always improved when the TMDA resource-sharing scheme is applied. Obviously, the independent M/M/1 queue system is inefficient, since there may be jobs queued up in front of one of the stations while another station is idle. The TMDA is less effective when the traffic intensity is low, since almost no queues are developed at any processor.

Comparing the TMDA to a centralized resource-sharing system (M/M/10),

Figure 5.12: Load-balancing in a heterogeneous computer system by queue length threshold or by workload threshold.

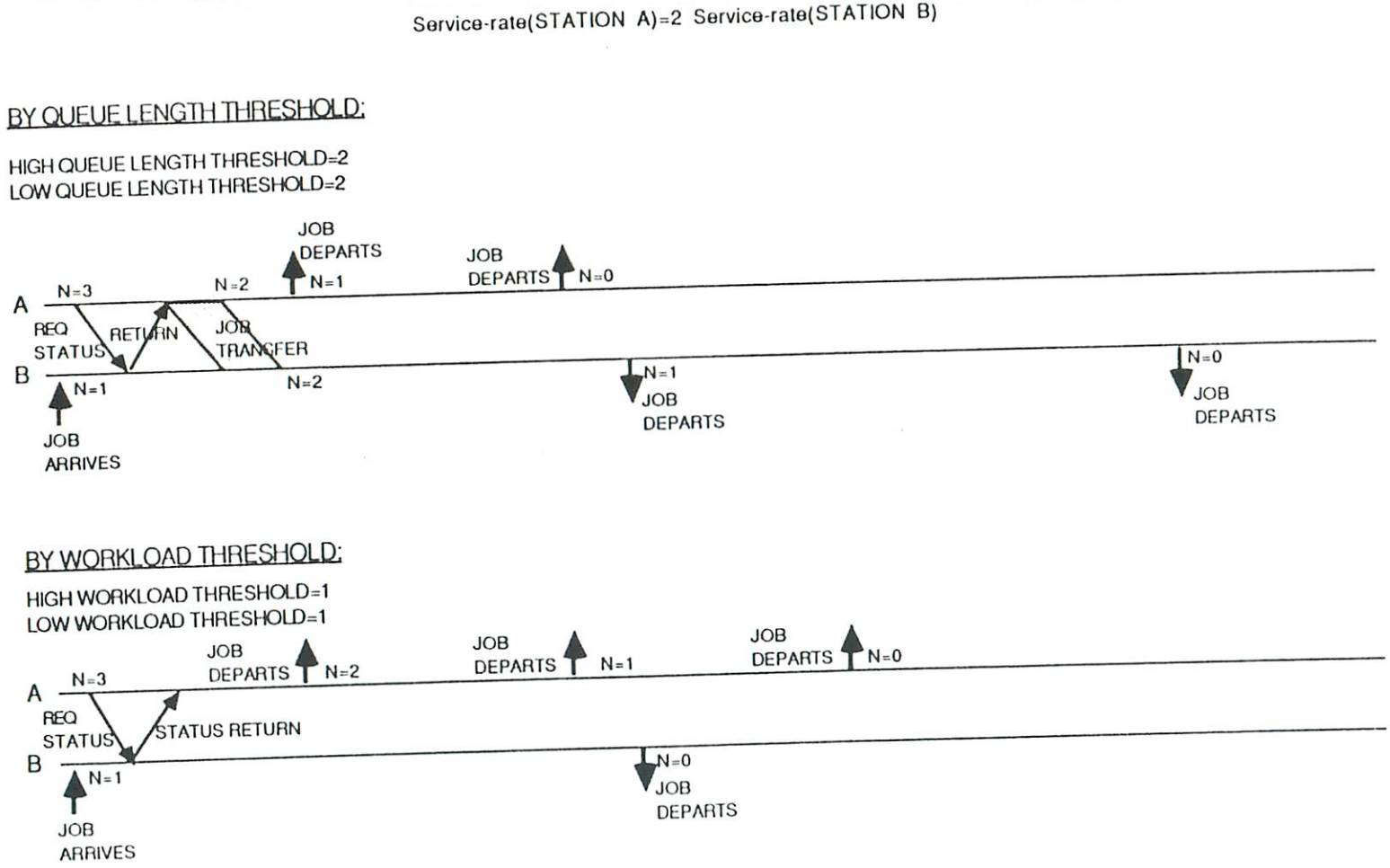


Figure 5.11 also shows that the TMDA performance is worse than the centralized M/M/10 system. The inefficiency of the TMDA can be illustrated by the following scenario: If one of the stations finishes the jobs in its queue and sees that all the other stations are not heavily loaded, this station will go idle and remain so until either another local job arrives or until its wakeup timeout. This station will not be able to share the load of other stations that become overloaded during its idle period. For this homogeneous system configuration, we can conclude that a centralized resource-sharing scheme provides the best performance, since no job will wait if any station is idle (although this scheme is generally impractical to implement).

### Job Transfer Ratio and Status Probing Ratio

Figure 5.13 shows the TMDA job transfer ratio for four different threshold combinations at various traffic intensities. The job transfer ratio is calculated by dividing the total number of jobs transferred between stations by the total number of jobs served in the system. The result shows that the lower the thresholds the more frequently a job will be transferred to balance the overall system load. Therefore, the more accurately the system will be balanced and the better the overall system response time will be. The job transfer ratio is relatively high during medium traffic intensity, since most servers are either idle at low traffic intensity or busy at high traffic intensity. This is consistent with the imbalance probability behavior [29].

Figure 5.14 shows the TMDA job status probing ratio for four different threshold combinations at various traffic intensities. The job probing ratio is defined as the average number of remote workload status requests per job arriving at the local station. The figure shows that the lower the threshold the more frequently the remote station will be polled. The job probing ratio

Figure 5.13: Job transfer ratio for TMDA with different combinations of thresholds. No overhead is considered.

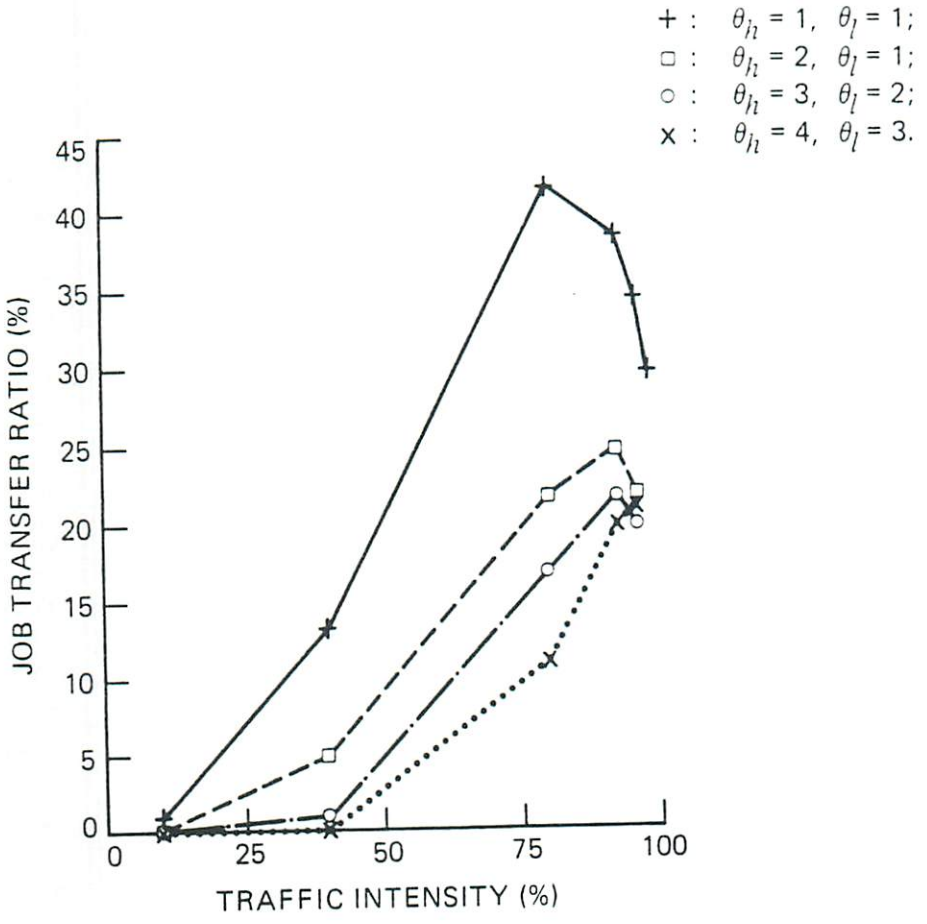
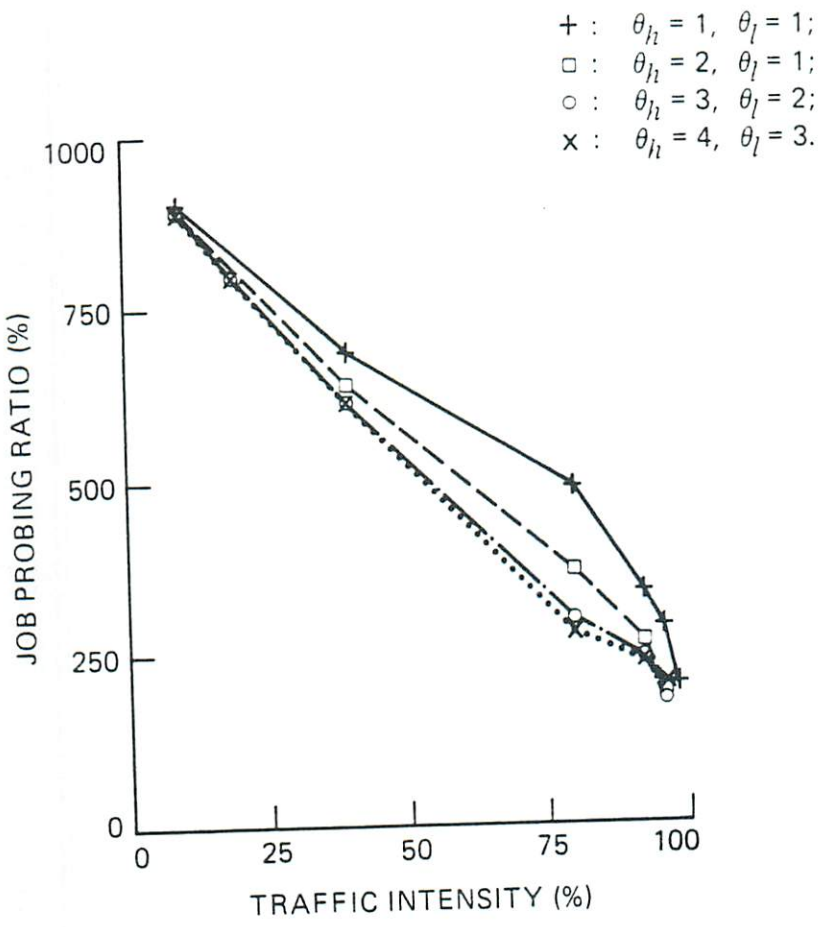


Figure 5.14: Job status probing ratio for TMDA with different combinations of thresholds. No overhead is considered.



is high at low traffic intensity. This is due partly to the fact that the TMDA will query the job status of the remote stations as soon as it finishes the last job. TMDA will also periodically wake up all the idle stations to do status polling. Furthermore, at low traffic intensity most stations will be polled before a heavily loaded station is found, since most stations are lightly loaded.

In the high traffic intensity range, more stations will switch to the receiver-initiated mode, in which the TMDA will not poll status unless a station is idle. Since few stations will be idle, few probes will be conducted. Furthermore, since most stations will be busy, TMDA will be able to identify a heavily loaded station in the first few polls. Therefore, the job status probing ratio is low.

Figure 5.15 shows the TMDA job transfer to job status probing ratio. This ratio is derived by dividing the total number of job transfers by the total number of status probes. This figure shows that the TMDA with $\theta_{h,m} = 1$ and $\theta_{l,m} = 1$ is more efficient than other threshold combinations, since it can obtain more jobs per status request.

### Comparison of TMDA with Sender-Initiated and Receiver-Initiated Schemes - without Overhead

Table 5.1 compares the mean system response time for the TMDA scheme with the sender-initiated and receiver-initiated schemes studied by Eager et al.[6] The threshold is set to one in all cases. These three schemes have fairly similar delay performance.

The sender-initiated scheme outperforms the receiver-initiated scheme when the system is lightly loaded. As illustrated in Figure 5.16, a job arriving at an overloaded station in the sender-initiated scheme can be transferred to a lightly loaded station as soon as it comes in, whereas the receiver-initiated

Figure 5.15: Job transfer to job status probing ratio for TMDA with different combinations of thresholds. No overhead is considered.

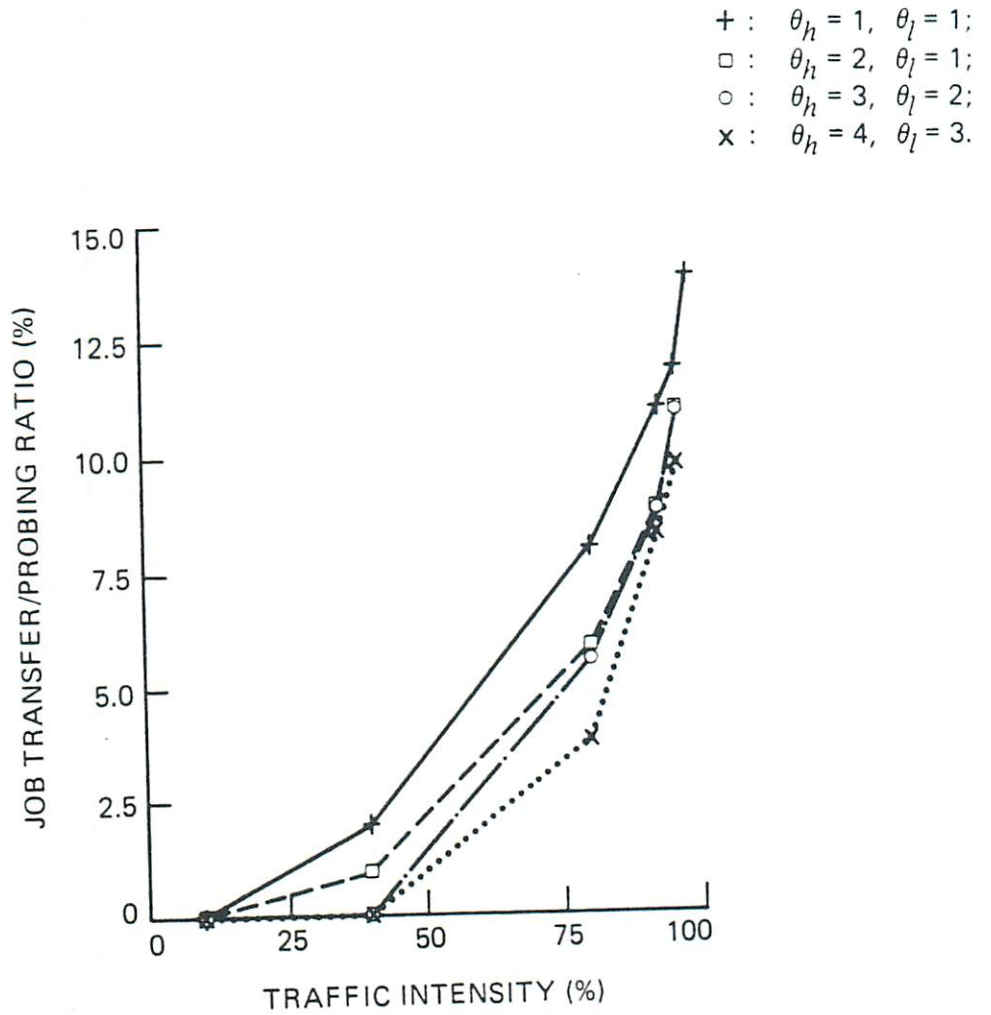


Table 5.1: Average response time comparison among TMDA simulation, sender-initiated simulation, and receiver-initiated simulation. No overhead is considered.

| Resource-Sharing Scheme | Delay(ms) at Various Traffic Intensities | | | | | |
|---|---|---|---|---|---|---|
| | 96% | 92.3% | 80% | 40% | 20% | 10% |
| TMDA $\theta_h = 1, \theta_l = 1$ | 182.75 ±11.7 | 127.33 ±9.3 | 85.00 ±0.0 | 63.00 ±0.0 | 56.01 ±0.7 | 51.67 ±0.7 |
| Sender-Initiated | 202.67 ±25.7 | 151.67 ±7.9 | 88.00 ±0.7 | 57.67 ±0.0 | 55.33 ±0.7 | 51.67 ±0.6 |
| Receiver-Initiated | 180.86 ±12.5 | 133.00 ±2.5 | 89.43 ±0.6 | 64.67 ±0.7 | 56.33 ±0.7 | 51.67 ±0.6 |

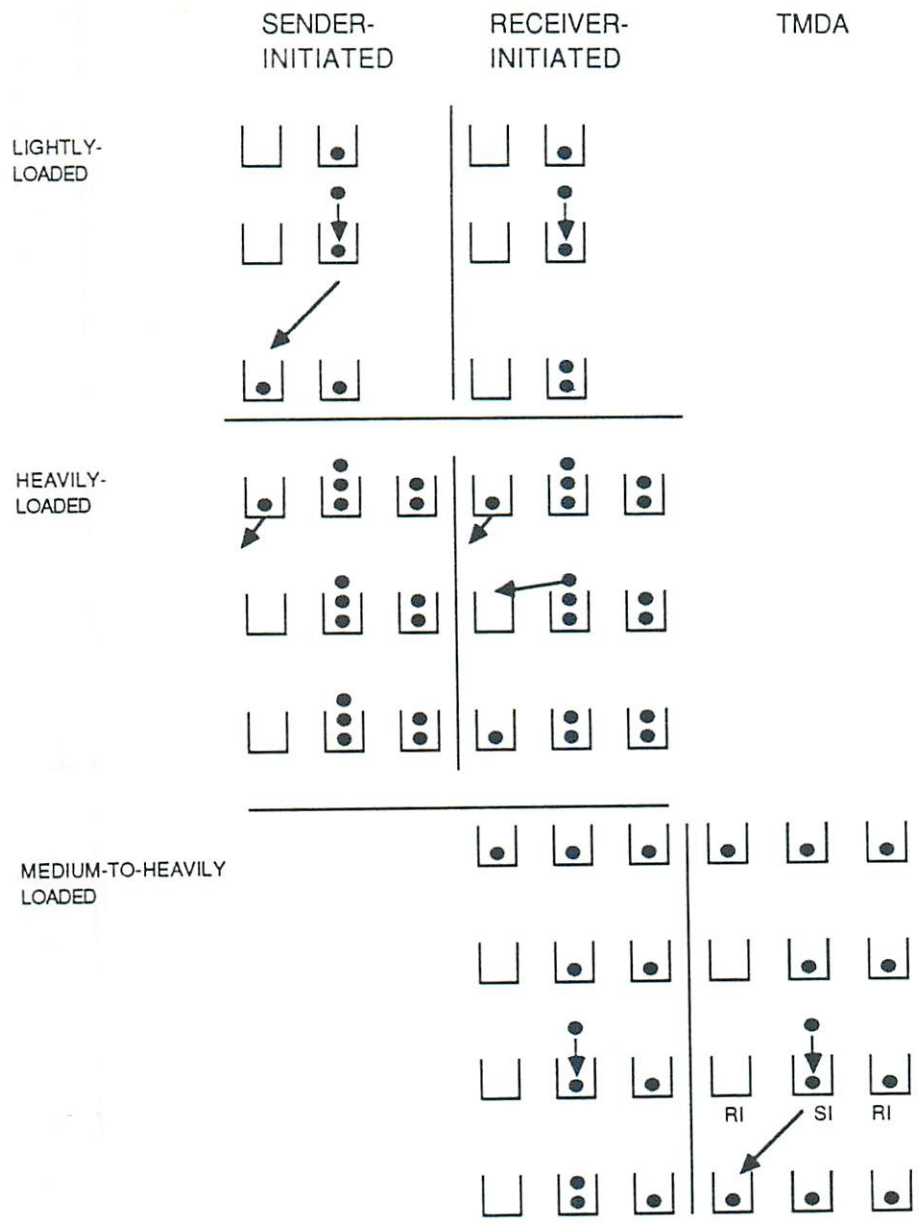
scheme, even if there are idle stations at the moment, cannot transfer that job unless a station becomes lightly loaded at some later time. Thus the receiver-initiated scheme wastes the resources of any idle stations during that time period.

On the other hand, the receiver-initiated scheme has better performance when the system is heavily loaded. As soon as a station becomes idle, it can pull jobs from heavily loaded stations, whereas in the sender-initiated scheme the job transfer will occur only when the next job arrives at the heavily loaded station. Furthermore, it is much easier for the lightly loaded station to find a heavily loaded station, since most of the stations will be heavily loaded. These behavior comparisons between sender-initiated and receiver-initiated schemes are consistent with Eager's report [6].

The behavior of the TMDA is very much like the receiver-initiated scheme when the system load is heavy and very much like the sender-initiated scheme when the load is light. TMDA performs even better at 80% and 92% traffic intensities. This behavior, shown in Figure 5.16, is explained as follows: One of the stations finishes its last job and searches for a heavily loaded station. It does not find any overloaded stations at that moment, so it goes idle. In the receiver-initiated scheme, that idle station cannot be accessed by any other stations that become overloaded. However, in the TMDA scheme, the idle station will be used immediately by the incoming jobs of any heavily loaded stations in sender-initiated mode.

In Figure 5.17 we see that the job transfer rate of the sender-initiated scheme drops dramatically at high load, while the TMDA and the receiver-initiated schemes continue to maintain activity on a certain level. Lacking job transfer, loads cannot be shared among processors to improve the performance. We conclude that the instability of the sender-initiated scheme at high loads

Figure 5.16: Comparison of sender-initiated, receiver-initiated, and TMDA schemes at different loadings.



is caused by both the high polling rate at a time when system resources are scarce and the lack of job transfers.

### 5.3.2 TMDA Performance: With Overhead

In the resource-sharing environment, two types of overhead should be considered:

a. the network transmission delay in transferring polling status and jobs from one station to another and in transferring the results back. Each sender-initiated poll requires the exchange of one polling message, while each receiver-initiated poll requires the exchange of a polling message and a reply message. A significant delay is incurred in transferring jobs and status if the network is slow.

b. the processing overhead for probing, packing and unpacking data, and request and result transfer. The overhead will significantly affect performance if the processing required to pack and unpack the transferring job is of the same order as the processing required to complete the job.

In this section, the overhead due to communication delay and packing/unpacking processing are investigated by simulation.

#### TMDA Delay with Different Processing Overhead

The GPSS/PC model includes the following overhead assumptions:

a. The network service time is exponentially distributed, and its mean value is 10 $\mu$s. Each load-balancing job is serviced by the network once.

b. The processing overhead can be varied from 1% to 50% of the job processing required.

Figure 5.17: Job transfer ratio comparison among TMDA, SI, and RI schemes. Thresholds are set to 1. No overhead is considered.

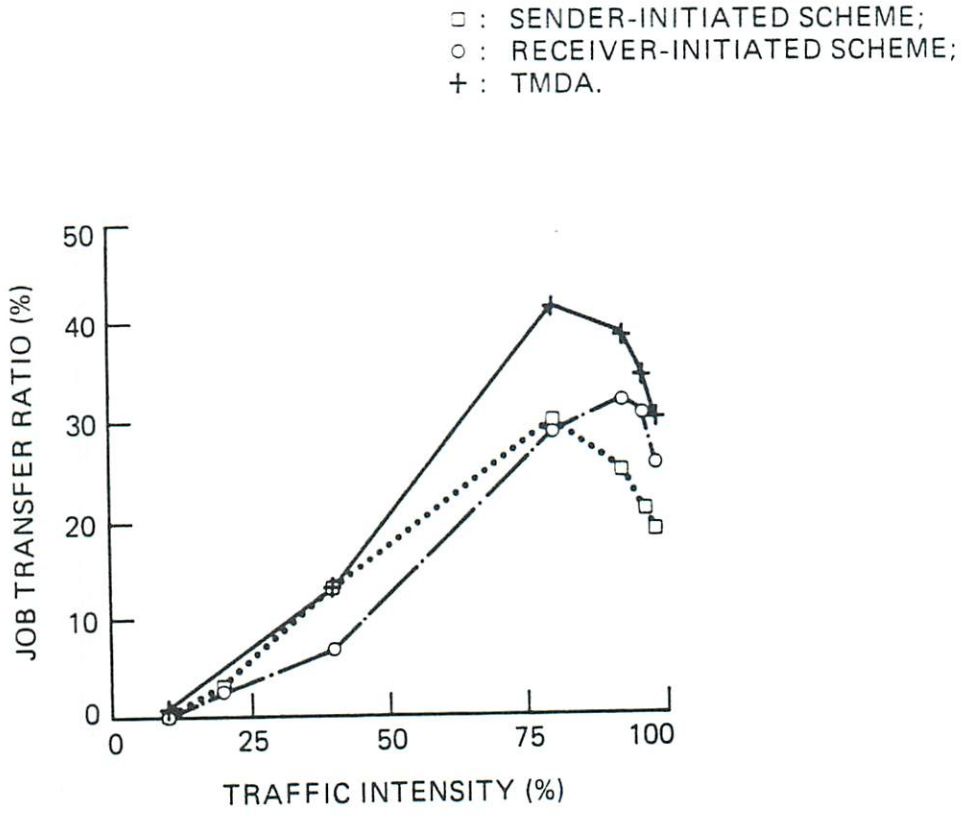


Figure 5.18: Job status probing ratio comparison among TMDA, SI, and RI schemes. Thresholds are set to 1. No overhead is considered.

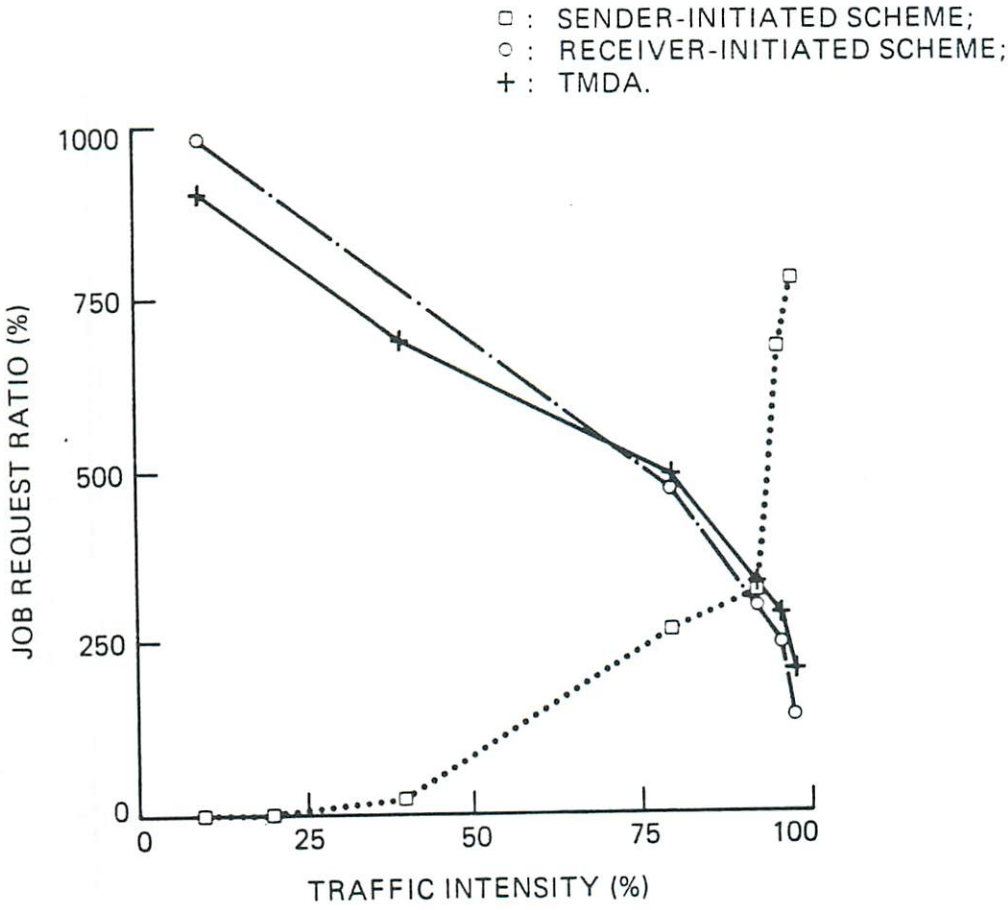
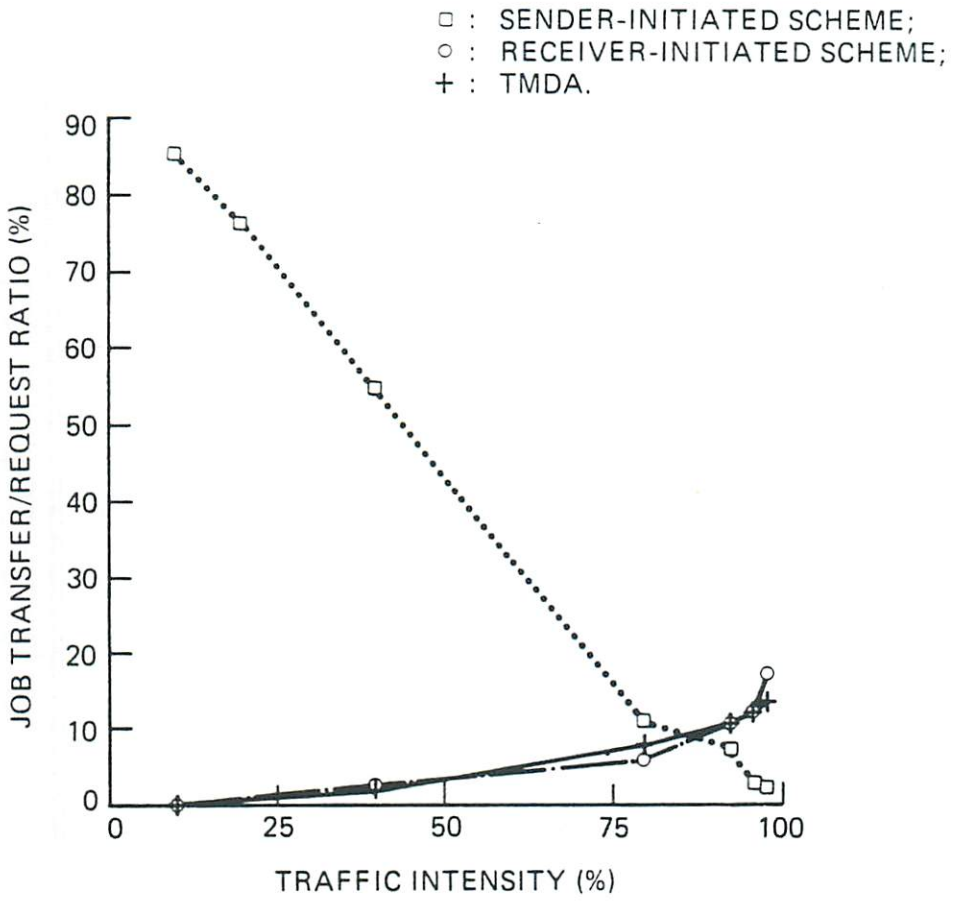


Figure 5.19: Job transfer to job status probing ratio comparison among TMDA, SI, and RI schemes. Thresholds are set to 1. No overhead is considered.



At some heavily loaded operational point, say 96%, the average response time of the TMDA with different packing/unpacking processing overheads is illustrated in Figure 5.20. The figure shows that the delay increases as the processing overhead increases. At 50% processing overhead, the delay is increased by a factor of four. However, the delay performance is still better than the no-resource-sharing M/M/1 case. The delay for M/M/1 is 6.5 times that of the no-overhead TMDA.

### Various Combinations of Thresholds with Overhead

With the assumption that the processing overhead is 10% and the communication delay is 10 ms, the delay performance of the TMDA with various combinations of high and low thresholds is investigated in this section. Figure 5.21 shows the TMDA delay behavior with four different threshold combinations: ($\theta_{h,m} = 4,\ \theta_{l,m} = 3$), ($\theta_{h,m} = 3,\ \theta_{l,m} = 2$), ($\theta_{h,m} = 2,\ \theta_{l,m} = 1$), and ($\theta_{h,m} = 1,\ \theta_{l,m} = 1$). The relationship among these four curves is the same as the case without overhead. It is interesting to note that the delay improvement for the case of ($\theta_{h,m} = 1,\ \theta_{l,m} = 1$) is reduced. This is because the high job transfer ratio and high status request ratio result in high overhead.

There are conflicting factors for the workload thresholds, $\theta_{h,m}$ and $\theta_{l,m}$. On the one hand, the workload threshold needs to be set as low as possible to allow frequent checking of the load status of each station and a sufficient number of job transfers between stations in order to accurately balance their loads. On the other hand, the workload threshold needs to set high to prevent an excessive number of job transfers resulting in high system overhead.

The tradeoff of different thresholds with various overheads for the mean response time is illustrated in Figures 5.22, 5.23, and 5.24. For low-overhead case, the mean response time of the system decreases with thresholds, whereas

Figure 5.20: Average response time for TMDA with different ratios of processing overhead compared to the job processing required. Threshold setting: $\theta_{h,m} = 1$, $\theta_{l,m} = 1$. Traffic intensity=96%.

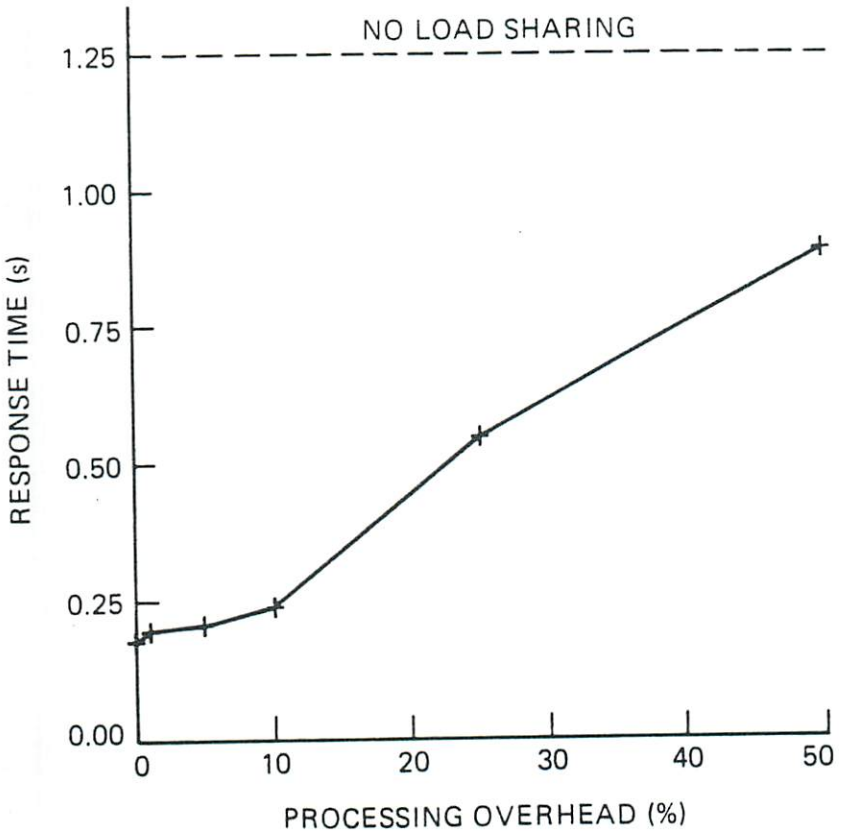
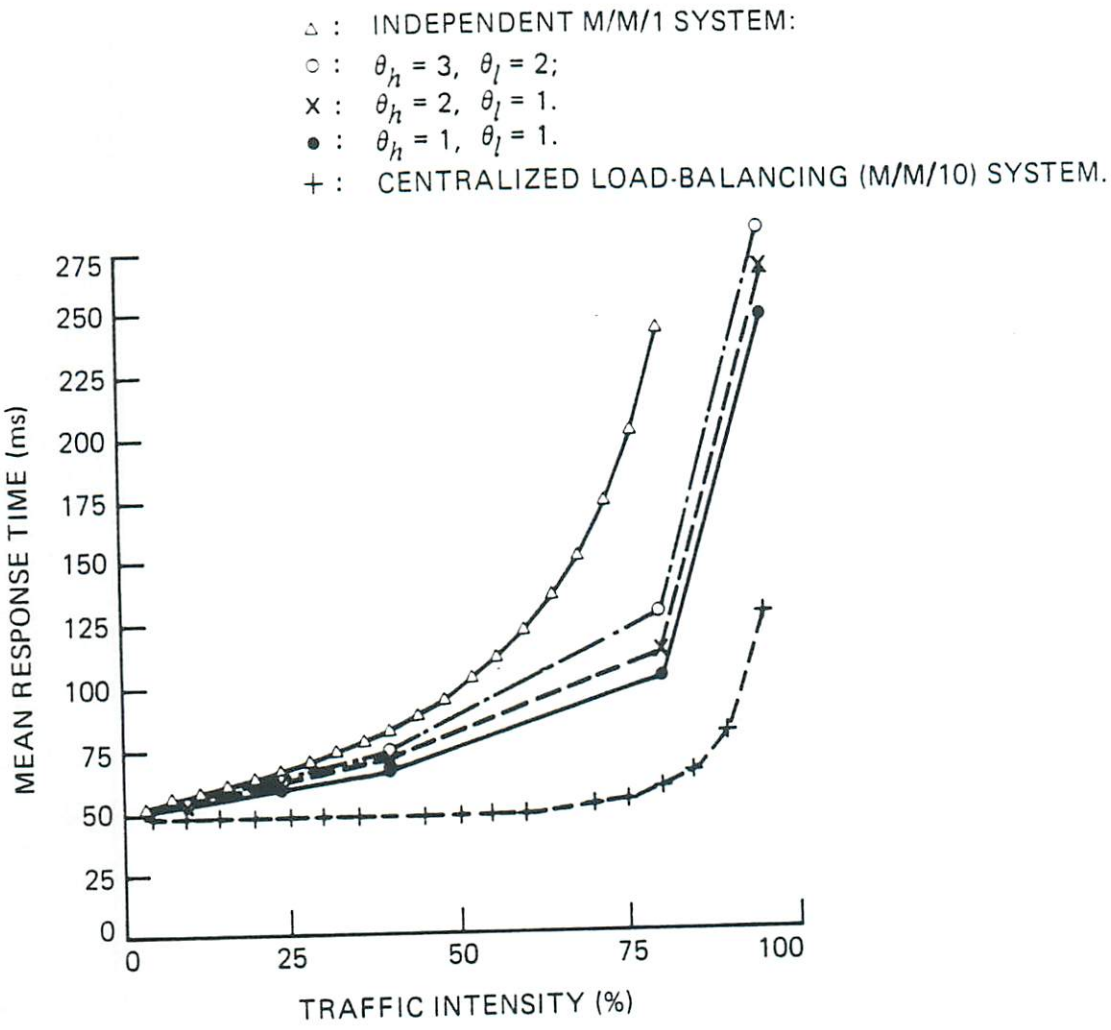


Figure 5.21: Average response times of TMDA GPSS/PC with several different threshold settings, compared with centralized (M/M/10) and an independent (M/M/1) schemes in a ten-server homogeneous system. 10% processing overhead and 10-ms communication delay are assumed.



the situation is reversed for the high overhead case. If all the stations are subject to heavy workload, thresholds should be set relatively high to avoid frequently invoking unproductive system overhead. In this situation, the performance gains obtained by load balancing will be negated by the large system overhead created by the job transfers.

As shown in Figure 5.23, it is also interesting to note that the thresholds should be dynamically adjusted as the system load conditions change, in order to make best use of the performance gains of load balancing.

### Comparison of TMDA with Sender-Initiated and Receiver-Initiated Schemes with Overhead

With the assumption that the processing overhead is 10% and the communication delay is 10 ms, the TMDA performance studies conducted in the previous section are repeated in this section. Figure 5.25 shows the comparison of TMDA with sender-initiated and receiver-initiated schemes with thresholds equal to one. Each scheme has the same overhead assumption. The TMDA scheme generally has better delay performance than the sender-initiated and receiver-initiated schemes, but does not outperform them by as much as it does in the no-overhead case. This is due to its high job transfer ratio, as shown in Figure 5.26. The job probing ratio and job transfer/probing ratio comparison for those three schemes are shown in Figures 5.27 and 5.28. These performance comparisons are very much like those for the no-overhead case as presented in the previous section.

Figure 5.22: Average response times of TMDA GPSS/PC with several different threshold settings, $\theta_{h,m} = \theta_{l,m} = 3$ and $\theta_{h,m} = \theta_{l,m} = 1$, in a ten-server homogeneous system. 10% processing overhead and no communication delay are assumed.

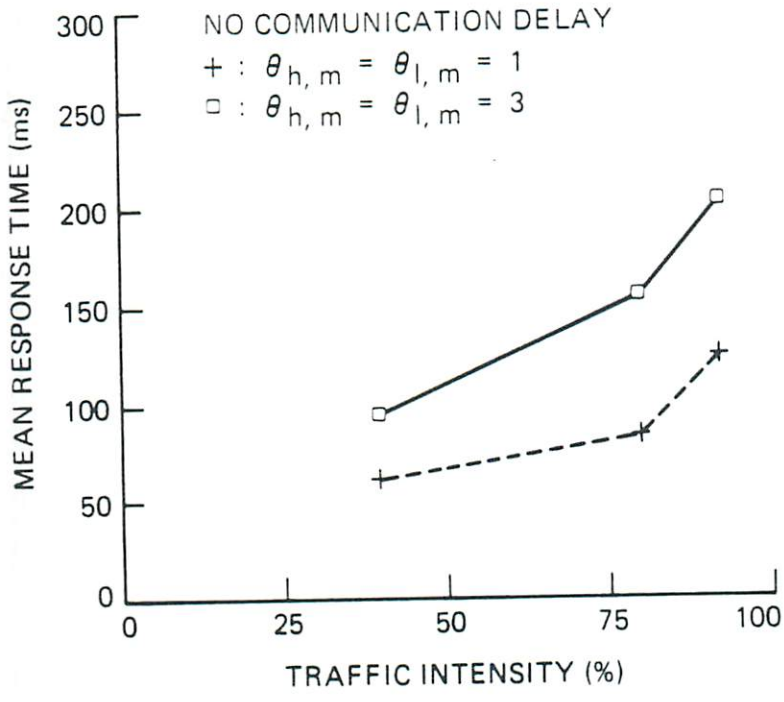


Figure 5.23: Average response times of TMDA GPSS/PC with several different threshold settings, $\theta_{h,m} = \theta_{l,m} = 3$ and $\theta_{h,m} = \theta_{l,m} = 1$, in a ten-server homogeneous system. 10% processing overhead and 100% communication delay are assumed.

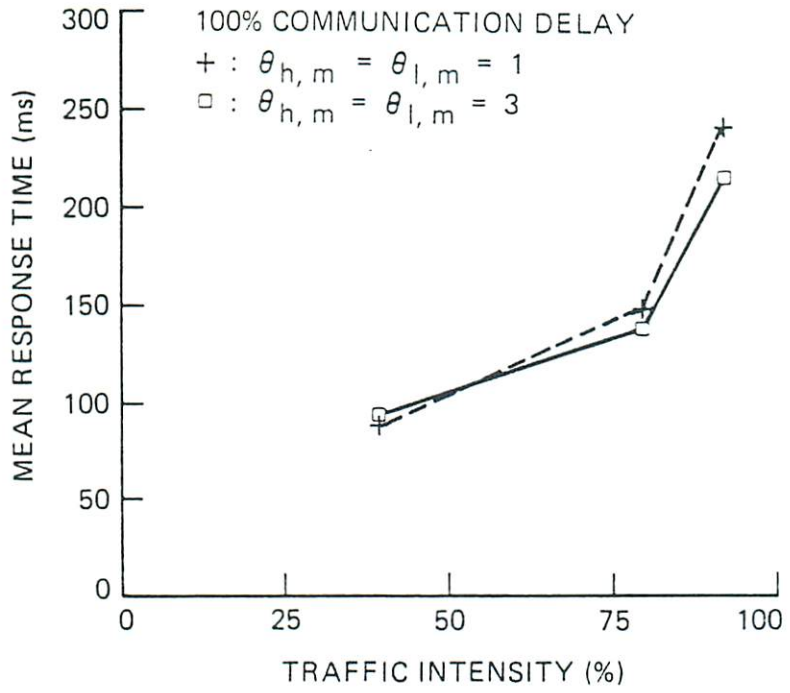


Figure 5.24: Average response times of TMDA GPSS/PC with several different threshold settings, $\theta_{h,m} = \theta_{l,m} = 3$ and $\theta_{h,m} = \theta_{l,m} = 1$, in a ten-server homogeneous system. 10% processing overhead and 500% communication delay are assumed.

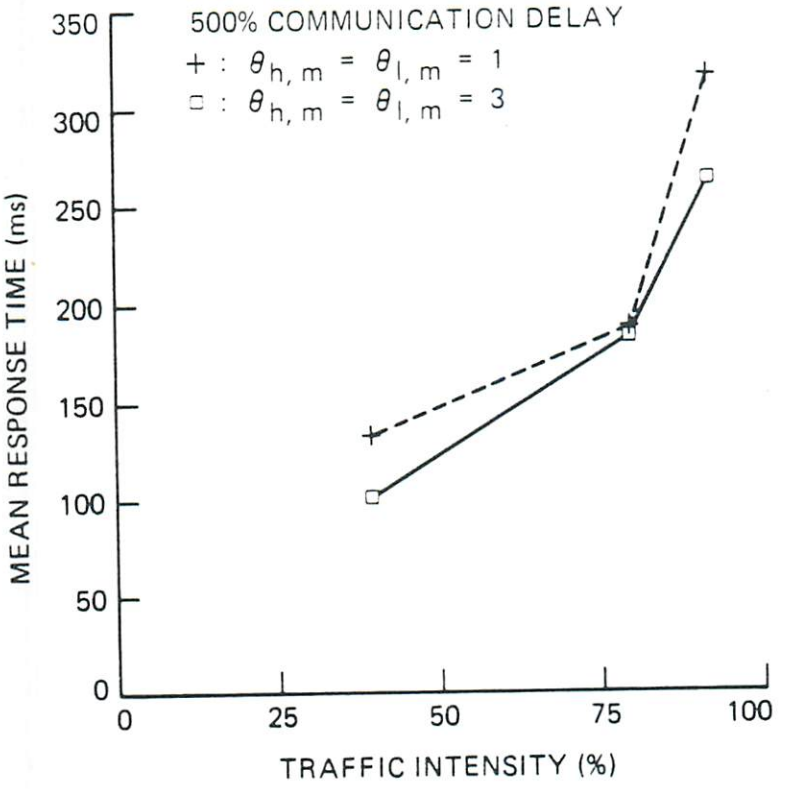


Figure 5.25: Average response time comparison among TMDA simulation, sender-initiated simulation, and receiver-initiated simulation. 10% processing overhead and 10-ms communication delay are assumed.

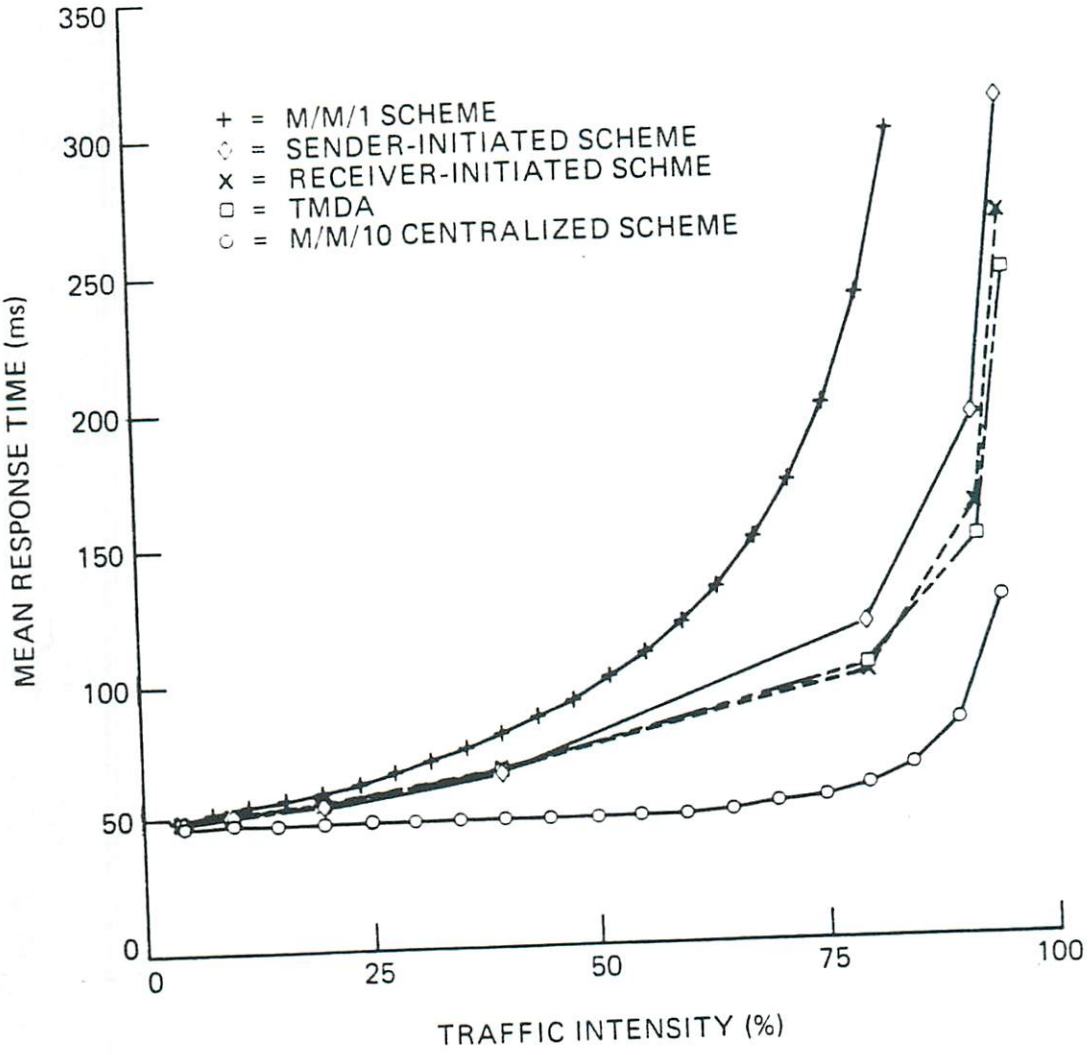


Figure 5.26: Job transfer ratio comparison among TMDA, sender-initiated, and receiver-initiated schemes. Thresholds are set to 1. 10% processing overhead and 10-ms communication delay are assumed.

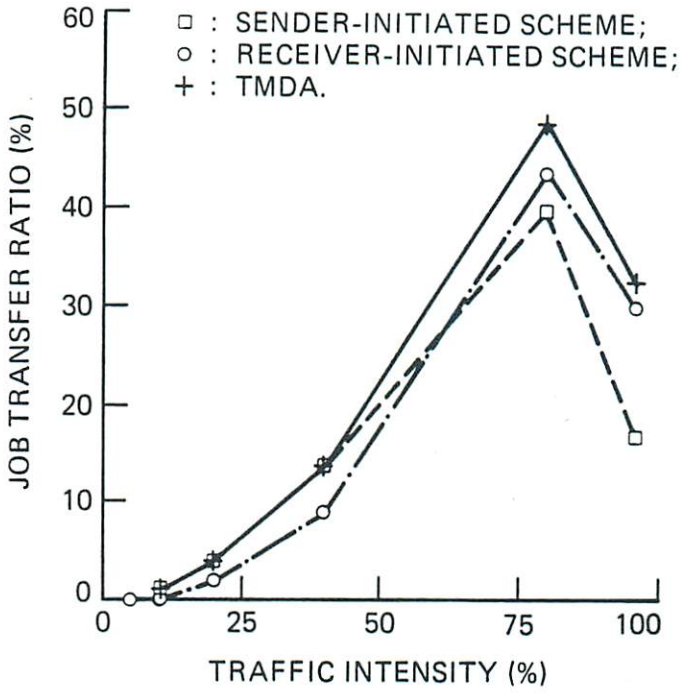


Figure 5.27: Job status probing ratio comparison among TMDA, sender-initiated, and receiver-initiated schemes. Thresholds are set to 1. 10% processing overhead and 10-ms communication delay are assumed.

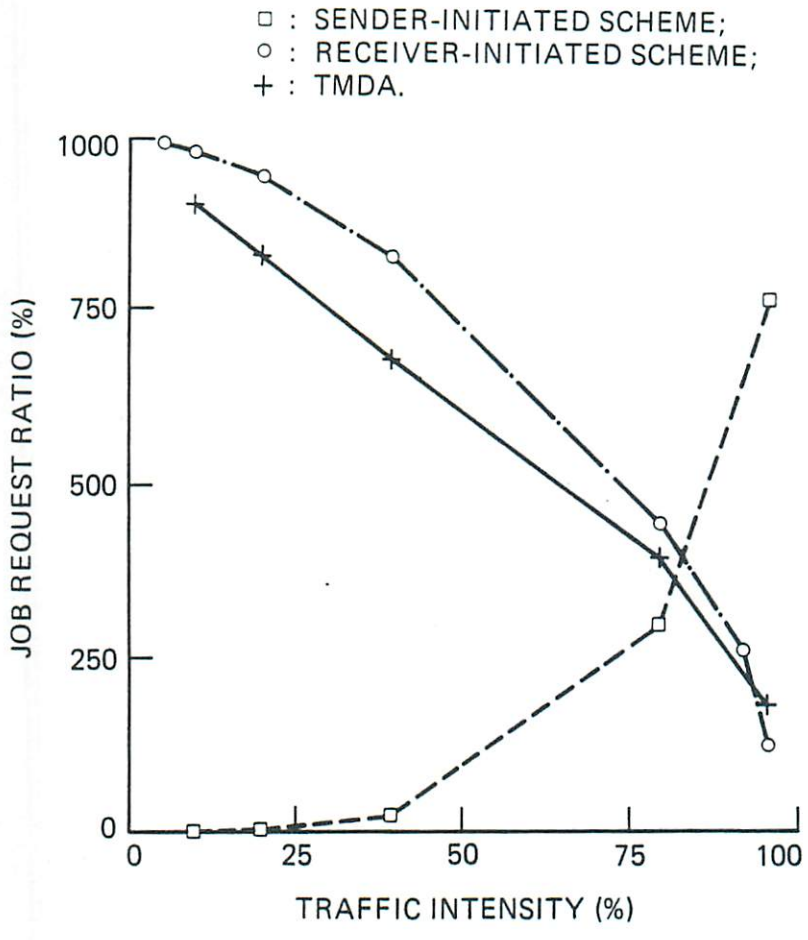
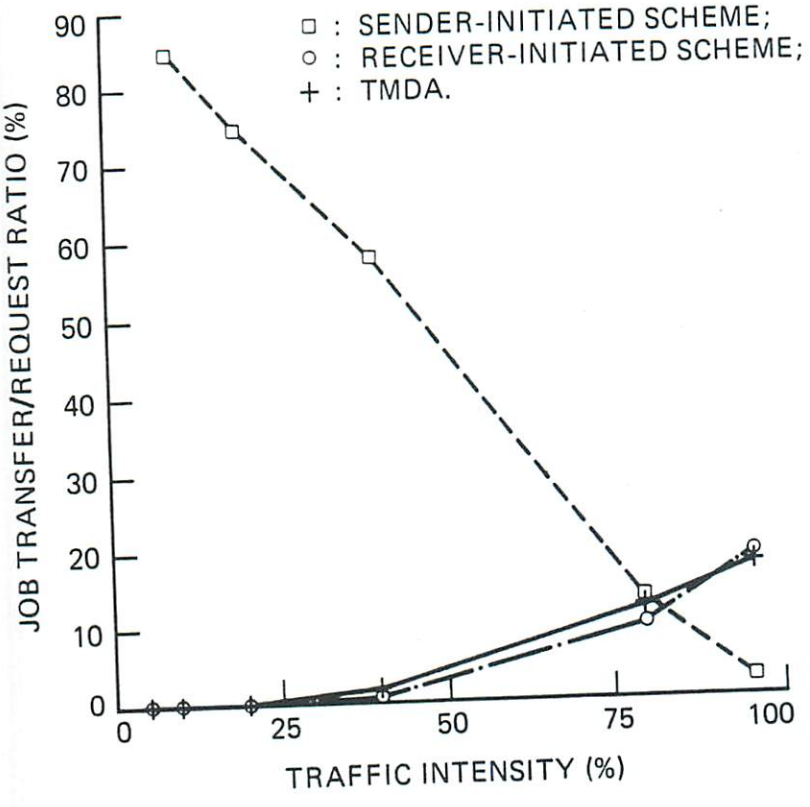


Figure 5.28: Job transfer to job status probing ratio comparison among TMDA, sender-, and receiver-initiated schemes. Thresholds are set to 1. 10% processing overhead and 10-ms communication delay are assumed.



## 5.4 TMDA Implementation Issues

Wah and Yuang presented the timing and memory issues for implementing window protocol on Ethernet interfaces for load sharing [46]. Eager et. al. also discussed some other implementation issues such as the threshold selection, instability, transfer cost for adaptive load sharing scheme in homogeneous distributed systems [7]. To ensure efficiency in implementation of the TMDA, solutions to several potential problems are presented in this section. A Pascal language implementation of the TMDA incorporating solutions to those issues is presented in Appendix B. Transaction details of the sender-initiated operation and receiver-initiated operation are shown in Figures 5.29 and 5.30.

A. Overhead evaluation before job transfer:

Nonnegligible processing and communication overhead impacts the correctness of the load-balancing decisions. Job transfer should be discouraged by significant overhead. To reduce the potential for wrong transfer, the sending station should estimate the expected finishing time if the transfer is done. This is accomplished by adding the time taken to: dequeue the job from the sending station; perform frame-packing of the job instructions and its supporting files; download those frames to the LAN; transmit to the destination station; upload at the destination host; perform frame unpacking; queue the job to the end of the destination waiting queue; await service (FIFO); complete service by the destination host; and finally to send the results back over the LAN to the originating station.

In correct-transfer policy, a valid job transfer should ensure that the expected finishing time with transfer is less than the expected finishing time without transfer. The expected finishing time without transfer is just the

Figure 5.29: TMDA sender-initiated operation.

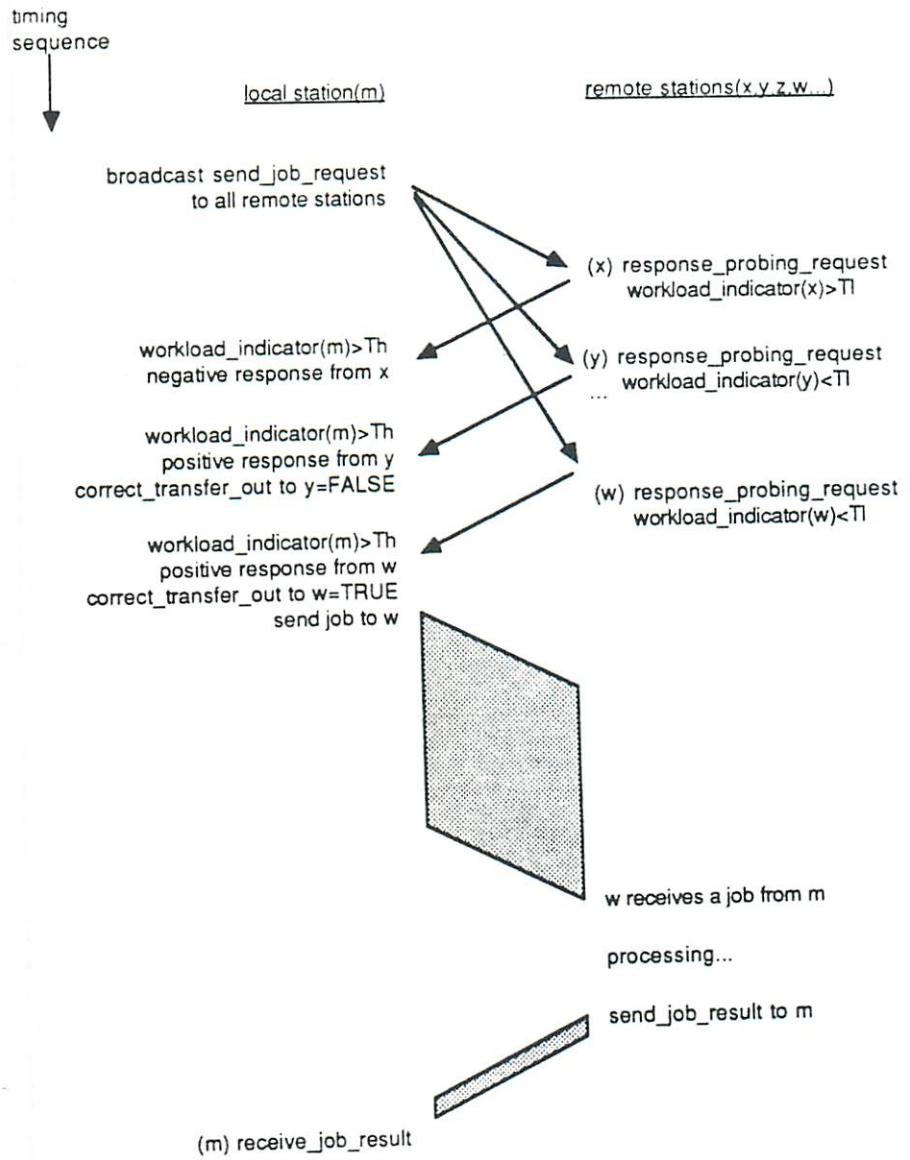
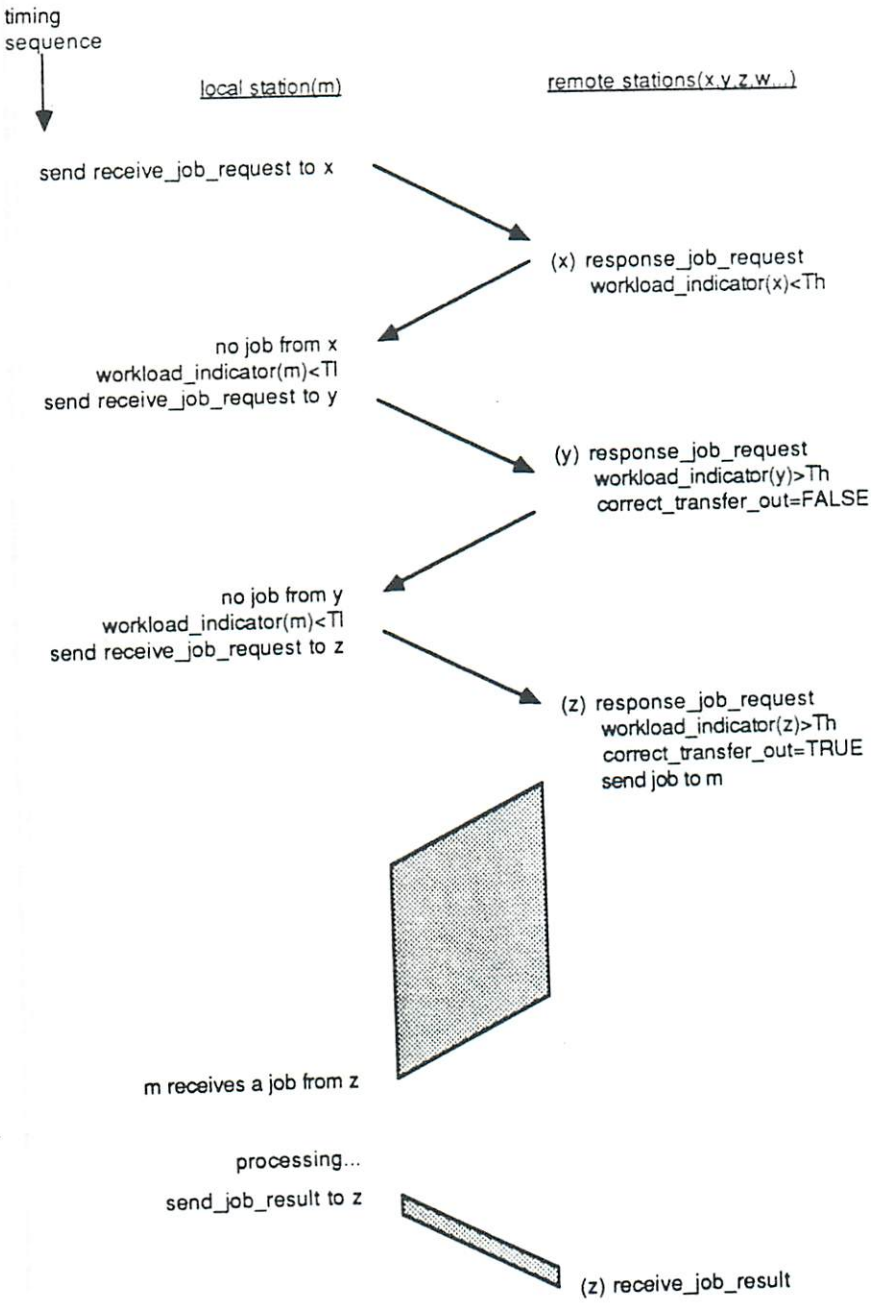


Figure 5.30: TMDA receiver-initiated operation.



mean waiting time in the local FIFO queue and the time for processing by the local station. If the expected finishing time with transfer is longer then the expected finishing time without transfer, the sending station should not initiate the transfer, even though the threshold conditions are met on both sides. To persist in the job transfer, ignoring the potential overhead, would delay the completion time of the transferred job, increase the overall response time of the system, and waste the processing and communication resources of the system.

This checking can be expressed by the 'correct-transfer evaluation formula':

$$\frac{N_{sender}I}{\mu_{sender}} - \left\{ \frac{\xi_{pk}\lceil \frac{L_{job}+L_{support\ files}}{L_{frame}} \rceil}{\mu_{sender}} + \frac{L_{job}+L_{support\ files}}{B_{lan}} + \frac{D_{lan}}{V_{lan}} + \frac{\xi_{upk}\lceil \frac{L_{job}+L_{support\ files}}{L_{frame}} \rceil}{\mu_{receiver}} + \frac{(N_{receiver}+1)I}{\mu_{receiver}} + \frac{\xi_{pk}\lceil \frac{L_{result}}{L_{frame}} \rceil}{\mu_{sender}} + \frac{L_{result}}{B_{lan}} + \frac{D_{lan}}{V_{lan}} + \frac{\xi_{upk}\lceil \frac{L_{result}}{L_{frame}} \rceil}{\mu_{receiver}} \right\} > \Delta_t \qquad (5.36)$$

where

$N_{sender}$=the queue length of the job-sending station;

$N_{receiver}$=the queue length of the job-receiving station;

$\mu_{sender}$=the service rate of the sending station, in instructions/sec;

$\mu_{receiver}$=the service rate of the receiving station, in instructions/sec;

$I$=the mean number of instructions per job;

$\xi_{pk}$=the processing overhead for frame packing, in instructions/frame;

$\xi_{upk}$=the processing overhead for frame unpacking, in instructions/frame;

$L_{job}$=file length of the job commands, in bytes;

$L_{result}$=file length of the job execution result, in bytes;

$L_{support\ file}$=the length of all the supporting files for job execution, such as run time libraries and data files, in bytes;

$L_{frame}$=LAN frame length, in bytes;

$B_{lan}$=network bandwidth (or data rate), in bytes/second;

$D_{lan}$=the LAN median distance, in meters;

$V_{lan}$=propagation velocity of the LAN medium, which is nearly constant among most media of interest. A good approximation for propagation velocity is about two-thirds of the speed of light, or 2 x $10^8$ m/s;

$\Delta_t$=time threshold for the job transfer, in seconds.

In the overhead evaluation formula, $\mu_{sender}$, $\mu_{receiver}$, $\xi_{pk}$, $\xi_{upk}$, $L_{frame}$, $I$, $B_{lan}$, $D_{lan}$, and $V_{lan}$ are all system configuration parameters that can be downloaded at system generation time. Depending on whether the particular transaction is sender initiated or receiver initiated, $N_{receiver}$ or $N_{sender}$ is provided by the probing response message. $N_{sender}$ or $N_{receiver}$ will be provided locally. $L_{job}$ and $L_{support\ files}$ can be obtained by two approaches: They can either be evaluated by checking the instruction map of the transferring job's task control block or approximated by the mean job and supporting file length. $L_{result}$ is typically small and can be ignored. $\Delta_t$ is an adaptable fine-tuning parameter.

The TMDA cannot rely solely on the correct-transfer evaluation formula (5.36) to determine how to transfer jobs. The proper workload thresholds

should also be checked. This is done to prevent frequent job transfers from a very heavily loaded station to an already heavily loaded one even through the correct-transfer evaluation formula may indicate a significant benefit from doing a job transfer. However, balancing loads between a very heavily loaded station and a heavily loaded station may not gain any performance benefit, since both of them are already busy.

B. Broadcast issues:

In a sender-initiated transaction, broadcasting the status-probing message is more time efficient than probing each remote station one by one. While the requested status messages coming back from each remote station are waiting in the status-probing response queue, the local station can be evaluating the status messages already received. As soon as it finds one remote station that fulfills the load-balancing requirements, the local station will transfer a job and purge the rest of the status response messages in the queue. The broadcasting scheme is not recommended for receiver-initiated transactions. If the local lightly loaded station broadcasts a requesting-job message, all the heavily loaded remote stations will transfer jobs simultaneously and the local lightly loaded station could suddenly become overloaded. Furthermore, the LAN may become heavily congested with messages trying to get through.

C. Controlling the sequence of job service:

A good load-sharing algorithm should maintain fairness of service. If sent to a remote lightly loaded station with fast speed and a short waiting line, the next job in the queue can be completed sooner than if processed locally. However, if either the remote station or the communication line is very slow, the completion time of that job will be delayed longer than if processed locally. In this case, the tail of the queue should be transferred to maintain fairness.

The correct-transfer evaluation formula (5.36) can be modified to yield a formula for selecting the job to be transferred. The modified formula becomes a 'job selection formula', expressed as follows:

$$\begin{aligned}
\text{If } \frac{(K-1)I}{\mu_{sender}} \leq & \left\{ \frac{(N_{receiver}+1)I}{\mu_{receiver}} + \frac{\xi_{pk}\lceil \frac{L_{job}+L_{support\ files}}{L_{frame}} \rceil}{\mu_{sender}} \right. \\
& + \frac{L_{job}+L_{support\ files}}{B_{lan}} + \frac{D_{lan}}{V_{lan}} + \frac{\xi_{upk}\lceil \frac{L_{job}+L_{support\ files}}{L_{frame}} \rceil}{\mu_{receiver}} \\
& \left. + \frac{\xi_{pk}\lceil \frac{L_{result}}{L_{frame}} \rceil}{\mu_{sender}} + \frac{L_{result}}{B_{lan}} + \frac{D_{lan}}{V_{lan}} + \frac{\xi_{upk}\lceil \frac{L_{result}}{L_{frame}} \rceil}{\mu_{receiver}} \right\} \qquad (5.37) \\
< & \frac{KI}{\mu_{sender}}
\end{aligned}$$

$$\text{then} \qquad \text{transfer the Kth job from queue.} \qquad (5.38)$$

The transferred job should be queued at the end of the waiting line in the receiving station, so as not to disturb the service sequence in the receiving station.

D. No hot potato transfer:

Each station is committed to serve a job transferred to it. Allowing a job to be retransferred could mean multiple handoffs from station to station and could delay the completion of the job indefinitely. This situation, while it might appear to improve the overall efficiency of the system, could lead to instability and large delay various.

E. Processing priority for resource sharing:

Research by Wah, et al.[46], classified four types of tasks to be served in a system using load-balancing scheme. They are (1) regular message transfer, (2) result return, (3) job migration, and (4) identification of the processors with the heaviest and the lightest loads. Of these, they assign the highest priority to the task of regular message transfer and a lower priority to the tasks involved in load-balancing process. This conservative approach can result in serious inhibition of the load-balancing process, since a heavily loaded processor may not be able to share its load with more lightly loaded stations because it is totally committed to processing the jobs in its queue.

By assigning the highest priority to the tasks of identifying load status and transferring jobs, we can assure that desirable job transfers can be effected in a timely manner, i.e., before the load status of either station changes significantly. Eager et al. [6] also make a similar suggestion. They suggested giving the transferring load-balancing tasks higher priority over the regular processing of tasks.

# Chapter 6

# Conclusions and future research

## 6.1 Conclusions

This thesis presents a study of load balancing in a distributed system consisting of a number of heterogeneous hosts connected by a LAN. We presented the motivation of resource sharing by showing the imbalance probability. We also reviewed previous research on resource sharing as related to our present work. An approximate queueing technique, the decoupled iterative technique (DIT), is proposed to evaluate the mean response time performance of a large network with heterogeneous interactive queues. This approximate model provides a solution to the problem of modeling a system with a dynamically changing job arrival rate and various service rates depending on the populations of the remote stations. To illustrate the effectiveness and efficiency of DIT, we include simple solutions for several complex examples. A GPSS/PC simulation is used to validate our approximation technique. Another validation method used is comparison with known solutions to similar problems. We proposed a new dynamic load-balancing scheme, called an enhanced receiver-initiated dynamic algorithm (ERIDA). The impact of the communication overhead is

discussed. We also proposed another effective scheme for load balancing, a two-mode dynamic algorithm (TMDA). The performance profile of the algorithm shows that the TMDA has better delay performance in the medium-to-high load range than either the sender-initiated scheme or the receiver-initiated scheme. We also developed a TMDA threshold selection strategy. DIT and extensive simulation are applied to both ERIDA and TMDA to evaluate the mean response time performance. In all the load-balancing schemes, the potential wrong-transfer probability introduced by the nonnegligible overhead was investigated. We suggested ways to reduce that potential problem. Numerical results are compared with other known schemes. This algorithm represents an improvement over the performance of the previously developed sender-initiated or receiver-initiated load-sharing policy [6]. These two new algorithms dynamically adjust to the traffic load and do not generate extra overhead during high traffic loading conditions. Therefore, they will not bring the system to an unstable state.

## 6.2 Future Research

Following are suggestions for further research on load balancing:

A. In this thesis, we have examined only the CPU load-balancing problem. The I/O load-balancing problem should be studied. As hardware costs decrease, the CPU load-balancing problem will become less important; however, the I/O problem will be relevant in distributed computer systems. We speculate that the I/O load-balancing problem is also NP-complete.

B. Laboratory implementation of distributed load balancing should be undertaken. This would provide experimental results on the performance

increase achievable with a proper load-balancing scheme. It would show the critical factors that affect performance.

C. The analytical model needs to be generalized. The resources modeled in this paper were limited to computer processing power and communication bandwidth. It should be possible to construct a more complete model that would include storage speed, update rate of files, printer allocation, gateway traffic, name server request, network traffic load, and any other special-purpose hardware resources.

# Bibliography


[1] Baran, S. and Dorsey, A., "Adaptive control of two competing queues," *IEEE Conference*, 1983, pp.427-435.

[2] Baumgartner, K.M. and Wah, B.W., "The Effects of Load Balancing on Response Time for Local Computer Systems with A Multiaccess Network," *IEEE International Comm. Conf.*, 1985, pp.10.1.1-10.1.5.

[3] Chang, H.Y. and Livny, M., "Distributed Scheduling under Deadline Constraints: a Comparison of Sender-Initiated and Receiver-Initiated Approaches," *Real-Time Symposium*, 1986.

[4] Chow, Y.C. and Kohler, W.H., "Models for Dynamic Load Balancing in a Heterogeneous Multiple Processor System," *IEEE Trans. on Computers*, Vol. C-28, No. 5, pp. 345-361, May 1979.

[5] Cooper, R., "Queues with Ordered Servers that Work at Different Rates," *Opsearch*, Vol. 13, No. 2, pp. 69-77, 1976.

[6] Eager, D.L., Lazowska, E.D., and Zahorjan, J., *A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing,* Technical Report 85-04-01, University of Washington, October. 1984.

[7] Eager, D.L., Lazowska, E.D., and Zahorjan, J., "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. on Software Eng.*, Vol. SE-12, No 5, May 1986.

[8] Foschini, G.J. and Salz, J., "A basic dynamic routing problem with diffusion," *IEEE Trans. Commun.*, Vol. COM-26, pp. 320-327, March 1978.

[9] Foschini, G.J., "Equilibria for diffusion models of pairs of communicating computers-symmetric case," *IEEE Trans. Inform. Theory,* Vol. IT-18, pp. 273-284, 1982.

[10] Flatto, L. and McKean, H., "Two queues in parallel," *Commun. Pure Appl. Math.,* Vol. 30, pp. 255-263, 1977.

[11] Goldberg, A. and Popek, G., "A validated distributed system performance model," *Performance,* 1983.

[12] Gross, D. and Harris, C. M., *Fundamentals of Queueing Theory,* John Wiley and Sons, Inc., New York, 1985.

[13] Grosh, H.A., "High speed arithmetic: The digital computer as a research tool," *J. Opt. Soc. Am.,* Vol 43, No. 4 (Apr.), pp. 11-16, 1953.

[14] Hwang, K. and Ni, L.M., "Optimal load balancing strategies for a multiple process system," *Proc. of Intl. Conf. on Parallel Processing,* August 1981.

[15] Hwang, K. and Croft, W.J., et al., "A Unix-based local computer network with load balancing," IEEE Computer magazine, April 1982.

[16] Hwang, C. and Liu, J.S., "Dymanic load balancing algorithms in homogeneous distributed systems," *Proc. of 6th International Conference on Distributed Computing Systems,* May 1986.

[17] Iglehart, D. and Whitt, W., "Multiple channel queues in heavy traffic," *Adv. Appl. Prob.,* vol. 2, pp. 150-177, 1970.

[18] Juang, J.Y., *Resource allocation in computer networks,* Ph.D. Dissertation, Purdue Univ., West Lafayette, IN, August 1985.

[19] Karian, Z.A., "GPSS/PC : Discrete-event simulation on the IBM PC," *Byte,* October 1985.

[20] King, J.L., *Centralization vs. decentralization of computing: an empirical assessment in city governments,* Public Policy Research Organization, University of California, Irvine, 1984.

[21] Knessel, C., Matkowsky, B., Schuss, Z., and Tier, C., "Two parallel queues with dynamic routing," *IEEE Trans. Commun.,* Vol. COM-34, No. 12, December 1986.

[22] Kleinrock, L., *Queueing Systems, Volume I: Theory,* John Wiley and Sons. Inc., New York, 1975.

[23] Kratzer, A. and Hammerstrom, D., "A study of load leveling," *COMPCON Symposium,* Spring 1980.

[24] Kurose, J. and Singh, S., "A distributed algorithm for optimal static load balancing in distributed computer systems," *IEEE Infocom,* April 1986.

[25] Lee, I. and Silvester, J.A., "An iterative schemes for performance modeling of slotted aloha packet radio network," *Proceeding of IEEE International Conference on Communications,* June 1982.

[26] Lin, W. and Kumar, P.R., "Optimal control of a queueing system with two heterogeneous servers," *IEEE Trans. on Automatic Control,* Vol AC-29, No. 8, August 1984.

[27] Little, J.D.C., "A Proof of the Queueing Formula $L = \lambda W$," *Operations Research,* 9, pp. 383-387, 1961.

[28] Livny, M. and Melman, M., "Load Balancing in Homogeneous Broadcast Distributed Systems," *ACM Computer Network Performance Symposium,* April 1982.

[29] Liu, H. and Silvester, J., *A New Resource Sharing Scheme in Distributed Heterogeneous Computer Systems,* Computer Research Institute Technical Report CRI-86-38, November 1986.

[30] Liu, H. and Silvester, J., *Resource Sharing Scheme in 2-server Heterogeneous Systems,* Computer Research Institute Technical Report CRI-87-45, August 1987.

[31] Liu, H. and Silvester, J., *Overhead Study for Resource Sharing Scheme in Distributed Heterogeneous Computer Systems,* Computer Research Institute Technical Report CRI-87-56, October 1987.

[32] Liu, H., *Optimal Resource Allocation in EOS,* Proceedings of the Space Station Workshop, JPL D-4149, March, 1987.

[33] Liu, H. and Silvester, J., *A New Dynamic Resource Allocation Scheme for Distributed Heterogeneous Computer Systems,* NASA Tech Briefs Journal, NPO-17197, to be published.

[34] Liu, H. and Silvester, J., "An Approximate Performance Model for Load-Dependent Interactive Queues in Distributed Systems," *IEEE Infocom,* New Orleans, L.A. March 1988.

[35] Liu, H. and Silvester, J., *A Two-Mode Dynamic Algorithm (TMDA) for Load Balancing in Distributed System,* Computer Research Institute Technical Report CRI-87-55, October 1987.

[36] Ni, L.M., "A distributed drafting algorithm for load balancing," *IEEE Trans. on software Eng.,* Vol.SE-11, No.10, October 1985.

[37] Rubinovitch, M., "The slow server problem: a queue with stalling," *J. Applied Probability* Vol. 22, pp. 879-892, 1985.

[38] Silva, E.S. and Gerla, M., "Load balancing in distributed systems with multiple classes and site constraints," *Performance 84* (North Holland), pp. 17-33, 1984.

[39] Stone, H.S., "Critical Load Factors in Two-Processor Distributed Systems," *IEEE, Trans. on Software Engineering,* Vol. SE-4, No. 3, May 1978, pp. 254-258.

[40] Towsley, D. and Lee, K.J., "On the analysis of a decentralized load sharing policy in heterogeneous distributed systems," submitted to *IEEE Trans. on Computers.*

[41] Towsley, D. and Lee, K.J., "A Comparison of Priority-Based Decentralized Load Balancing Policies," *ACM Performance Evaluation Review,* Vol 14, No. 1, May 1986.

[42] Mirchandaney, R., Towsley, D., and Stankovic, J.A., "Analysis of the Effects of Delays on Load Sharing," submitted to *Trans. on Computers,* February 1987.

[43] Mirchandaney, R., Towsley, D., and Stankovic, J.A., "Effects of Delays on Receiver-Initiated Load Sharing Policies," submitted to *Journal on Parallel and Distributed Systems,* February 1987.

[44] Tantawi, A.N. and Towsley, D., "Optimal Static Load Balancing in Distributed Computers," *JACM,* Vol. 32, No. 2, April 1985.

[45] Trivedi, K.S., *Probability and statistics with reliability, queuing and computer science applications,* Prentice-Hall, Inc., New York, 1982.

[46] Wah, B. and Yuang, J.Y., "Resource Scheduling for Local Computer Systems with a Multiaccess Network," *IEEE Trans. on Computers,* Vol. C-34, No, 12, December 1985.

[47] Wah, B. and Lien, Y.N., "Design of distributed databases on local computer systems with a multiaccess network," *IEEE trans. on Software Eng.,* Vol. SE-11, No. 7, July 1985.

[48] Wang, Y.T. and Morris, R.J.T., "Load sharing in distributed systems," *IEEE Trans. on Computers,* Vol. C-34, pp. 204-217, March 1985.

[49] Ward, P.T. and Mellor, S.J., *Structured development for real-time systems, Volume II: Implementation Modeling Techniques* Yourdon Press, New York, England, 1985.

[50] Zahorjan, J. and Lazowska, E. D., "Incorporating Load Dependent Servers in Approximate Mean Value Analysis," *Performance Evaluation Review,* Special Issue, Vol. 12, No. 3, August 1984.

# Appendix A

# Pascal implementation for ERIDA

```
{*****************************************************************
*                    ERIDA Program                                *
*****************************************************************}
PROGRAM ERIDA {Enhanced Receiver-Initiated Dynamic Algorithm};

TYPE
    priority_type=(high,low);              {priority of job}

    msg_type     =(probing_message,
                   probing_response,
                   job_request,
                   job_sending,
                   job_result);            {type of message}

    job_fmt =                              {job information}
    RECORD
          ins :REAL;                       {# of insts of the job}
          msg_length  :INTEGER;            {job message length}
          body:array[1..500] OF REAL;      {job body}
    END;

    msg_fmt =                              {format of LAN message}
    RECORD
          message_type   :msg_type;
          message_id     :INTEGER;         {2 byte}
          source_id      :INTEGER;         {the ID of each node}
          destination_id :INTEGER;         {the ID of each node
                                            or "broadcast_mode"}
          priority       :priority_type;   {1 byte}
          workload       :REAL;            {2 byte}
{         data           :array[1..1485] OF byte;  }
                                            {IEEE 802.3 standard}
```

```
        data            :string[15];
        queue_length    :INTEGER;
        job             :job_fmt;
    END;

  q_length_array = array [0..255] OF INTEGER;
                                        {q length for node}

  service_array = array [0..255] OF REAL;
                                        {service rate}


VAR
  low_workload_threshold :INTEGER;      {low workload threshold}
  station_number         :INTEGER;      {total no of nodes}
  local_id               :INTEGER;      {local node ID}
  dist_lan               :REAL;         {LAN medium length}
  mu_lan                 :REAL;         {LAN service rate (B/s)}
  oh_pk_ratio            :REAL;         {processing overhead ratio
                                        for packing(instructions/
                                        byte message length)}
  oh_upk_ratio           :REAL;         {processing overhead ratio
                                        for unpackaging(insts/
                                        byte message length)}
  workload_delta         :REAL;         {workload difference for
                                        correct job transfer}
  q_length               :q_length_array;
                                        {q length for each node}
  mu                     :service_array;{service rate for each
                                        node (ins/sec)}
  probing_message_queue  :REAL;
  mean_job_ins           :REAL;         {mean # of instructions}
  queue_length           :INTEGER;
  ii,jj,kk               :INTEGER;      {index}


PROCEDURE LAN_trans_service(VAR send_msg);
BEGIN
END;

PROCEDURE receive_any(VAR return_msg);
BEGIN
END;

PROCEDURE MATCH_RESULT;
BEGIN
END;

PROCEDURE deque(VAR job);
```

```
BEGIN
END;

PROCEDURE wait(VAR job);
BEGIN
END;

PROCEDURE purge(queue:REAL);
BEGIN
END;

FUNCTION end_of_probing_msg_q:BOOLEAN;
BEGIN
END;


{*****************************************************************
* Procedure INITIALIZE initialize all the system parameters:    *
* local_id   :local node ID.                                    *
* mu_lan     :LAN service rate (in bytes/sec)                   *
* mu[0..255] :service rate for each node (in insts/sec)         *
* proc_oh_pk :processing overhead to do packing at the sen_id   *
*             node (in number of instruction/1024 byte)         *
* proc_oh_upk:processing overhead to do unpacking at the rec_id *
*             node (in number of instruction/1024 byte)         *
*****************************************************************}
PROCEDURE initialize;
BEGIN
     local_id:=32;
     mu_lan:=4E6/8;                    {Token ring (4 Mbit/sec)}
     dist_lan:=5E2;                    {500 meters}
     low_workload_threshold:=3;
     node_number:=99;
     workload_delta:=1;                {difference for correct
                                        transfer}
     FOR ii:=0 TO 255 DO
     BEGIN
         mu[ii]:=5E6;                  {5 MIPS machine}
     END;
     mean_job_ins:=2.4E5;              {mean job insts is 240,000}
     oh_pk_ratio:=30;                  {packing processing
                                        overhead (ins/frame)}
     oh_upk_ratio:=20;                 {unpacking processing
                                        overhead (ins/frame)}
END; {of initialize}


{*****************************************************************
*WORKLOAD_INDICATOR calculates the workload indicator of a node  *
```

```
*****************************************************************}
FUNCTION workload_indicator(node_queue_length:INTEGER;
                            node_id:INTEGER)          :REAL;

VAR
     mu:service_array;

BEGIN
     workload_indicator:=node_queue_length/mu[node_id];
END; {of workload_indicator}



{*****************************************************************
*                   receive job request                          *
*****************************************************************}
PROCEDURE receive_job_request
                     (dest_id:INTEGER; VAR send_msg:msg_fmt);

VAR
  local_queue_length:INTEGER;

BEGIN
  send_msg.message_type:=job_request;
  send_msg.source_id:=local_id;
  send_msg.destination_id:=dest_id;
  send_msg.priority:=high;
  send_msg.queue_length:=local_queue_length;
  LAN_trans_service(send_msg);
END; {of send_job_request}


{*****************************************************************
*                   send job                                     *
*****************************************************************}
PROCEDURE send_job(dest_id:INTEGER; VAR send_msg:msg_fmt);

VAR
job :job_fmt;

BEGIN
  send_msg.message_type:=job_sending;
  send_msg.source_id:=local_id;
  send_msg.destination_id:=dest_id;
  send_msg.priority:=high;
  deque(job);                              {deque head of q or tail
                                            of q}
  send_msg.job:=job;
  LAN_trans_service(send_msg);
```

```
END; {of send_job}


{*****************************************************************
*                    send job result                             *
*****************************************************************}
PROCEDURE send_job_result(dest_id:INTEGER; VAR send_msg:msg_fmt);

VAR
  results:string[15];

BEGIN
  send_msg.message_type:=job_result;
  send_msg.source_id:=local_id;
  send_msg.destination_id:=dest_id;
  send_msg.priority:=high;
  send_msg.data:=results;
  LAN_trans_service(send_msg);
END; {of send_job_result}


{*****************************************************************
*                    receive job result                          *
*****************************************************************}
PROCEDURE receive_job_result;

VAR return_msg : msg_fmt;

BEGIN
  receive_any(return_msg);              {return_result?}
  MATCH_RESULT;                         {match result to the
                                         original via message_id
                                         attribute of the
                                         return_msg}
END; {of receive_job_result}


{*****************************************************************
*                        receive job                             *
*****************************************************************}
PROCEDURE receive_job;

VAR return_msg : msg_fmt;

BEGIN
  receive_any(return_msg);              {return_result?}
END; {of receive_job}
```

```
{*****************************************************************
*correct_transfer_out evaluates the correctness of transferring a*
*job from local node, local_id, to a remote node.                *
*****************************************************************}
FUNCTION correct_transfer_out(rm_id:INTEGER;  {remote node id}
                              q_length_rm_id:INTEGER;
                                             {remote q length}
                              q_length_local_id:INTEGER;
                              local_job:job_fmt)        :BOOLEAN;

VAR
   mu:service_array;

BEGIN
     IF ((local_job.msg_length/1500*oh_pk_ratio)/mu[local_id]
                                        {packing processing
                                        overhead}
     +(local_job.msg_length/mu_lan)     {transmission delay}
     +dist_lan/(2E8)                    {propagation delay}
     +((q_length_rm_id+1)*mean_job_ins)/mu[rm_id]
                                        {remote node service time}
     +(local_job.msg_length/1500*oh_upk_ratio)/mu[rm_id])
                                        {unpacking processing
                                         overhead}
     -(q_length_local_id/mu[local_id]) > workload_delta
     THEN
        correct_transfer_OUT:=TRUE
     ELSE
        correct_transfer_OUT:=FALSE;
END; {of correct_transfer_out}


{*****************************************************************
*                   response job request                         *
*****************************************************************}
PROCEDURE response_job_request;

VAR
   job_req_msg,job_resp_msg : msg_fmt;
   local_queue_length       : INTEGER;
   job                      :job_fmt;

BEGIN
  receive_any(job_req_msg);{"job request?"}
  IF workload_indicator(local_queue_length,local_id)
                           > low_workload_threshold  THEN
  BEGIN
       IF correct_transfer_out(job_req_msg.source_id,
                               job_req_msg.queue_length,
```

```
                              local_queue_length,
                              job)                    = true THEN
        BEGIN
          job_resp_msg.message_type:=job_sending;
          job_resp_msg.source_id:=local_id;
          job_resp_msg.destination_id:=job_req_msg.source_id;
          job_resp_msg.priority:=high;
          deque(job);                    {deque from the head or
                                          tail of the waiting q}
          job_resp_msg.job:=job;
          LAN_trans_service(job_resp_msg);
                                          {send the job to the
                                           requesting node}
        END
        ELSE
        BEGIN
          job_resp_msg.message_type:=probing_response;
          job_resp_msg.source_id:=local_id;
          job_resp_msg.destination_id:=job_req_msg.source_id;
          job_resp_msg.priority:=high;
          job_resp_msg.data:='not available';
          LAN_trans_service(job_resp_msg);
        END;
 END
 ELSE
   BEGIN
          job_resp_msg.message_type:=probing_response;
          job_resp_msg.source_id:=local_id;
          job_resp_msg.destination_id:=job_req_msg.source_id;
          job_resp_msg.priority:=high;
          job_resp_msg.data:='not available';
          LAN_trans_service(job_resp_msg);
  END;
END; {of response_job_request}


{*****************************************************************
*                  job transfer                                  *
*****************************************************************}
{* Procedure job_transfer;                                       *
*      wait for job departs                                       *
*      WHILE local workload indicator < threshold                 *
*           send receive job request to each one of remote nodes*
*           IF workload indicator of remote node>high threshold *
*           THEN                                                  *
*                check if it is a 'correct transfer'              *
*                receive job                                      *
*                finish job                                       *
```

```
*                  return job result                                    *
*********************************************************************}
PROCEDURE job_transfer;
VAR
    job_departure_event        : BOOLEAN;
    job_finish_event           : BOOLEAN;
    wakeup_timeout             : BOOLEAN;
    send_msg                   : msg_fmt;
    job_response_msg           : msg_fmt;
    return_msg                 : msg_fmt;
    system_maybe_lightly_loaded: BOOLEAN;   {system loading guess
                                             from local node}
    system_maybe_heavily_loaded: BOOLEAN;   {system loading guess
                                             from local node}

   local_job                   : job_fmt;

BEGIN
{      IF (q_length[local_id]=0 and wakeup_timeout) or
       (job_departure_event)
       THEN
       BEGIN}
       WHILE workload_indicator(q_length[local_id],local_id)
             < low_workload_threshold
          BEGIN
             system_maybe_lightly_loaded:= TRUE;
          FOR kk:=1 TO node_number DO
          BEGIN
              receive_job_request(kk,send_msg); {send the workload
                                                 req to all the
                                                 other nodes}
             receive_any(return_msg);
             IF workload_indicator(return_msg.queue_length,
                                   return_msg.source_id) >
                                   low_workload_threshold THEN
                     BEGIN
                        system_maybe_lightly_loaded:=FALSE;
                        IF correct_transfer_out(return_msg.source_id,
                          return_msg.queue_length,q_length[local_id],
                          return_msg.job)  = TRUE   THEN
                          BEGIN
                              receive_job;
                              send_job_result(return_msg.source_id
                                              ,send_msg);
                              exit;
                          END;
                     END {of work_load_indicator}
          END; {of while not EOQ}
     END; {of receiver_initiated}
```

```
END;{of job_transfer}


{*****************************************************************
*                            ERIDA main program                   *
*****************************************************************}
BEGIN
  {CONCURRENT}
  {LOOP}
      response_job_request;
  {FOREVER}
  {LOOP}
      receive_job_result;
  {FOREVER}
  {LOOP}
      job_transfer;
  {FOREVER}
  {LOOP}
     {process normal process at low priority}
  {FOREVER}
{ENDCONCURRENT}
END. {ERIDA:Enhanced Receiver-Initiated Dynamic Algorithm}
```

# Appendix B

# Pascal implementation for TMDA

```
{*****************************************************************
*                  TMDA Program  for USC token ring network     *
*****************************************************************}
PROGRAM TMDA {Two-Mode Dynamic Algorithm};

TYPE
    priority_type=(high,low);           {priority of job}

    msg_type     =(probing_message,
                   probing_response,
                   job_request,
                   job_sending,
                   job_result);         {type of message}

    job_fmt      =                      {job information}
    RECORD
          ins :REAL;                    {# of insts of the job}
          msg_length  :INTEGER;         {job msg length}
          body:array[1..500] OF REAL;   {job body}
    END;

    msg_fmt      =                      {format of the LAN msg}
    RECORD
          message_type   :msg_type;
          message_id     :INTEGER;      {2 byte}
          source_id      :INTEGER;      {the ID of each node}
          destination_id :INTEGER;      {the ID of each node or
                                         "broadcast_mode"}
          priority       :priority_type;{1 byte}
          workload       :REAL;         {2 byte}
{         data           :array[1..1485] OF byte;  }
                                        {IEEE 802.3 standard}
```

```
        data            :string[15];
        q_length        :INTEGER;
        job             :job_fmt;
    END;

  q_length_array = array [0..255] OF INTEGER;
                                        {q length for each node}

  service_array = array [0..255] OF REAL;{service rate for each node}

  mode = (sender_initiated,receiver_initiated);


VAR
  low_workload_threshold :INTEGER;      {low workload threshold}
  high_workload_threshold:INTEGER;      {high workload threshold}
  TMDA_mode              :mode;         {sender_ or receiver_ini}
  node_number            :INTEGER;      {total # of nodes}
  local_id               :INTEGER;      {local node ID}
  dist_lan               :REAL;         {LAN medium length}
  mu_lan                 :REAL;         {LAN service rate (byte/s)}
  oh_pk_ratio            :REAL;         {processing overhead ratio
                                         for packing(insts/
                                        byte msg length)}
  oh_upk_ratio           :REAL;         {processing overhead ratio
                                         for unpackaging (insts/
                                        byte msg length)}
  workload_delta         :REAL;         {workload difference for
                                         correct job transfer}
  q_length               :q_length_array;{q length for each node}
  mu                     :service_array;{service rate for each node
                                        (ins/sec)}
  probing_msg_q  :REAL;
  mean_job_ins           :REAL;         {mean # of insts}
  q_length               :INTEGER;
  ii,jj,kk               :INTEGER;      {index}


PROCEDURE LAN_trans_service(VAR send_msg);
BEGIN
END;

PROCEDURE receive_any(VAR return_msg);
BEGIN
END;

PROCEDURE MATCH_RESULT;
BEGIN
END;
```

```
PROCEDURE deque(VAR job);
BEGIN
END;

PROCEDURE wait(VAR job);
BEGIN
END;

PROCEDURE purge(q:REAL);
BEGIN
END;

FUNCTION end_of_probing_msg_q:BOOLEAN;
BEGIN
END;


{*****************************************************************
* Procedure INITIALIZE initialize all the system parameters:     *
* local_id   :local node ID.                                     *
* mu_lan     :LAN service rate (in bytes/sec)                    *
* mu[0..255] :service rate for each node (in insts/sec)          *
* proc_oh_pk :processing overhead to do packing at the sen_id    *
*             node (in # of instruction/1024 byte)               *
* proc_oh_upk:processing overhead to do unpacking at the rec_id  *
*             node (in # of instruction/1024 byte)               *
*****************************************************************}
PROCEDURE initialize;
BEGIN
     local_id:=32;
     mu_lan:=4E6/8;                    {Token ring (4 Mbit/sec)}
     dist_lan:=5E2;                    {500 meters}
     high_workload_threshold:=4;
     low_workload_threshold:=3;
     node_number:=99;
     workload_delta:=1;                {diff for correct transfer}
     FOR ii:=0 TO 255 DO
     BEGIN
         mu[ii]:=5E6;                  {5 MIPS machine}
     END;
     mean_job_ins:=2.4E5;              {mean job inst is 240,000}
     oh_pk_ratio:=30;                  {packing processing
                                        overhead (ins/frame)}
     oh_upk_ratio:=20;                 {unpacking processing
                                        overhead (ins/frame)}
END; {of initialize}
```

```
{*****************************************************************
*WORKLOAD_INDICATOR calculates the workload indicator of a node  *
*****************************************************************}
FUNCTION workload_indicator(node_q_length:INTEGER;
                            node_id:INTEGER)          :REAL;

VAR
     mu:service_array;

BEGIN
     workload_indicator:=node_q_length/mu[node_id];
END; {of workload_indicator}



{*****************************************************************
*                   send job request                             *
*****************************************************************}
PROCEDURE send_job_request(dest_id:INTEGER; VAR send_msg:msg_fmt);

VAR
  local_q_length:INTEGER;

BEGIN
  send_msg.msg_type:=job_request;
  send_msg.source_id:=local_id;
  send_msg.destination_id:=dest_id;     {if destination_id=0,
                                         broadcast msg}
  send_msg.priority:=high;
  send_msg.q_length:=local_q_length;
  LAN_trans_service(send_msg);
END; {of send_job_request}


{*****************************************************************
*                   receive job request                          *
*****************************************************************}
PROCEDURE receive_job_request
                         (dest_id:INTEGER; VAR send_msg:msg_fmt);

VAR
  local_q_length:INTEGER;

BEGIN
  send_msg.msg_type:=job_request;
  send_msg.source_id:=local_id;
  send_msg.destination_id:=dest_id;
  send_msg.priority:=high;
  send_msg.q_length:=local_q_length;
```

```
  LAN_trans_service(send_msg);
END; {of send_job_request}


{*****************************************************************
*                   send job                                     *
*****************************************************************}
PROCEDURE send_job(dest_id:INTEGER; VAR send_msg:msg_fmt);

VAR
job :job_fmt;

BEGIN
  send_msg.msg_type:=job_sending;
  send_msg.source_id:=local_id;
  send_msg.destination_id:=dest_id;
  send_msg.priority:=high;
  deque(job);                              {deque head of q or tail
                                            of q}
  send_msg.job:=job;
  LAN_trans_service(send_msg);
END; {of send_job}


{*****************************************************************
*                   send job result                              *
*****************************************************************}
PROCEDURE send_job_result(dest_id:INTEGER; VAR send_msg:msg_fmt);

VAR
  results:string[15];

BEGIN
  send_msg.msg_type:=job_result;
  send_msg.source_id:=local_id;
  send_msg.destination_id:=dest_id;
  send_msg.priority:=high;
  send_msg.data:=results;
  LAN_trans_service(send_msg);
END; {of send_job_result}


{*****************************************************************
*                        response probing request                *
*****************************************************************}
PROCEDURE response_probing_request;

VAR
  q_length  : INTEGER;
```

```
  probing_msg   : msg_fmt;

BEGIN
  receive_any(probing_msg);{"probing msg?");
  probing_resp_msg.msg_type:=probing_response;
  probing_resp_msg.msg_id:=probing_msg.msg_id;
  probing_resp_msg.source_id:=local_id;
  probing_resp_msg.destination_id:=probing_msg.source_id;
  probing_resp_msg.priority:=high;
  probing_resp_msg.workload:=workload_indicator(q_length,local_id);
  LAN_trans_service(probing_resp_msg);  {send the workload indicator
                                         back to the input buffer
                                         q of requesting node}
END; {of response_probing_request}


{*****************************************************************
*                 receive job result                             *
*****************************************************************}
PROCEDURE receive_job_result;

VAR return_msg : msg_fmt;

BEGIN
  receive_any(return_msg);              {return_result?}
  MATCH_RESULT;                         {match result to the original
                                         job via msg_id attribute
                                         of the return_msg}
END; {of receive_job_result}


{*****************************************************************
*                        receive job                             *
*****************************************************************}
PROCEDURE receive_job;

VAR return_msg : msg_fmt;

BEGIN
  receive_any(return_msg);              {return_result?}
END; {of receive_job}


{*****************************************************************
*correct_transfer_out evaluates the correctness of transferring *
*a job from local node, local_id, to a remote node.             *
*****************************************************************}
FUNCTION correct_transfer_out(rm_id:INTEGER;         {remote node id}
                              q_length_rm_id:INTEGER;{remote q length}
```

```
                              q_length_local_id:INTEGER;
                              local_job:job_fmt)        :BOOLEAN;

VAR
   mu:service_array;

BEGIN
     IF ((local_job.msg_length/1500*oh_pk_ratio)/mu[local_id]
                                        {packing processing
                                        overhead}
     +(local_job.msg_length/mu_lan)     {transmission delay}
     +dist_lan/(2E8)                    {propagation delay}
     +((q_length_rm_id+1)*mean_job_ins)/mu[rm_id]
                                        {remote node service time}
     +(local_job.msg_length/1500*oh_upk_ratio)/mu[rm_id])
                                        {unpacking processing oh}
     -(q_length_local_id/mu[local_id]) > workload_delta
     THEN
        correct_transfer_OUT:=TRUE
     ELSE
        correct_transfer_OUT:=FALSE;
END; {of correct_transfer_out}


{*****************************************************************
*                  response job request                          *
*****************************************************************}
PROCEDURE response_job_request;

VAR
   job_req_msg,job_resp_msg : msg_fmt;
   local_q_length       : INTEGER;
   job                      :job_fmt;

BEGIN
  receive_any(job_req_msg);{"job request?"}
  IF workload_indicator(local_q_length,local_id)
                           > high_workload_threshold  THEN
  BEGIN
       IF correct_transfer_out(job_req_msg.source_id,
                               job_req_msg.q_length,
                               local_q_length,
                               job)                  = true THEN
       BEGIN
         job_resp_msg.msg_type:=job_sending;
         job_resp_msg.source_id:=local_id;
         job_resp_msg.destination_id:=job_req_msg.source_id;
         job_resp_msg.priority:=high;
         deque(job);                    {deque from the head or
```

```
                                        tail of the waiting q}
        job_resp_msg.job:=job;
        LAN_trans_service(job_resp_msg);
                                        {send the job to the
                                         requesting node}
      END
      ELSE
      BEGIN
        job_resp_msg.msg_type:=probing_response;
        job_resp_msg.source_id:=local_id;
        job_resp_msg.destination_id:=job_req_msg.source_id;
        job_resp_msg.priority:=high;
        job_resp_msg.data:='not available';
        LAN_trans_service(job_resp_msg);
      END;
 END
 ELSE
   BEGIN
        job_resp_msg.msg_type:=probing_response;
        job_resp_msg.source_id:=local_id;
        job_resp_msg.destination_id:=job_req_msg.source_id;
        job_resp_msg.priority:=high;
        job_resp_msg.data:='not available';
        LAN_trans_service(job_resp_msg);
  END;
END; {of response_job_request}


{*****************************************************************
*                  job transfer                                  *
*****************************************************************}
{* Procedure job_transfer;                                       *
*   IF TMDA_mode = receiver initiated THEN                       *
*      wait for job departs                                      *
*      WHILE local workload indicator < low_threshold            *
*            send receive job request to each one of remote nodes*
*            IF workload indicator of remote node>high threshold *
*            THEN                                               *
*                 check if it is a 'correct transfer'           *
*                 receive job                                   *
*                 finish job                                    *
*                 return job result                             *
*            ELSE                                               *
*                 switch mode                                   *
*                                                               *
*  IF TMDA_mode = sender initiated THEN                         *
*     wait for job arrives or idle                              *
*     WHILE local workload indicator > high_threshold           *
```

```
*           broadcast the workload request to all the other nodes*
*           deque the probing response msg                        *
*           IF workload indicator of remote node < low threshold *
*           THEN                                                  *
*                 check if it is a 'correct transfer'             *
*                 transfer job                                    *
*                 receive job result                              *
*           ELSE                                                  *
*                 switch mode                                     *
*****************************************************************}
PROCEDURE job_transfer;
VAR
    job_departure_event        : BOOLEAN;
    job_arrival_event          : BOOLEAN;
    job_finish_event           : BOOLEAN;
    wakeup_timeout             : BOOLEAN;
    send_msg                   : msg_fmt;
    job_response_msg           : msg_fmt;
    return_msg                 : msg_fmt;
    system_maybe_lightly_loaded: BOOLEAN;   {system loading guess
                                             from local node}
    system_maybe_heavily_loaded: BOOLEAN;   {system loading guess
                                             from local node}
   local_job                   : job_fmt;

BEGIN
   IF TMDA_mode=receiver_initiated THEN
{r_ini}BEGIN
      wait(job_departure_event);
      WHILE workload_indicator(q_length[local_id],local_id)
            < low_workload_threshold
         BEGIN
            system_maybe_lightly_loaded:= TRUE;
         FOR kk:=1 TO node_number DO
         BEGIN
            receive_job_request(kk,send_msg);  {send the workload req
                                                to all the other nodes}
            receive_any(return_msg);
            IF workload_indicator(return_msg.q_length,
                                  return_msg.source_id) >
                                  high_workload_threshold THEN
                    BEGIN
                       system_maybe_lightly_loaded:=FALSE;
                       IF correct_transfer_out(return_msg.source_id,
                         return_msg.q_length,q_length[local_id],
                         return_msg.job)  = TRUE   THEN
                         BEGIN
                             receive_job;
                             send_job_result(return_msg.source_id
```

```
                                              ,send_msg);
                              exit;
                           END;
                      END {of work_load_indicator}
          END; {of while not EOQ}
             IF system_maybe_lightly_loaded=true
             THEN
                TMDA_mode:=sender_initiated;
          END
       END; {of receiver_initiated}


   IF TMDA_mode=sender_initiated THEN
      BEGIN
{      IF (q_length[local_id]=0 and wakeup_timeout) or
         (q_length[local_id]=1 and job departs)
      THEN
         GOTO r_ini;
      ELSE
}
      wait(job_arrival_event);
      WHILE workload_indicator(q_length[local_id],local_id)
            > low_workload_threshold
         BEGIN
            send_job_request(0,send_msg);{broadcast the workload req
                                          to all the other nodes}
            system_maybe_heavily_loaded:=true;
            WHILE NOT end_of_probing_msg_q
               BEGIN
                  deque(return_msg);
                  IF workload_indicator(return_msg.q_length,
                                        return_msg.source_id) <
                                        low_workload_threshold THEN
                     BEGIN
                        system_maybe_lightly_loaded:=false;
                        deque(local_job); {deque job from the
                                           head of q}
                        IF correct_transfer_out(return_msg.source_id,
                          return_msg.q_length,q_length[local_id],
                          return_msg.job)     = TRUE
                           THEN
                              send_job(return_msg.source_id,send_msg);
                     END {of work_load_indicator}
               END; {of while not EOQ}
            IF system_maybe_heavily_loaded=true THEN
               TMDA_mode:=receiver_initiated;
         END
      END {of sender_initiated}
END;{of job_transfer}
```

```
{*********************************************************************
*                            TMDA main program                        *
*********************************************************************}
BEGIN
  CONCURRENT
  LOOP
      response_probing_request;
  FOREVER
  LOOP
      response_job_request;
  FOREVER
  LOOP
      receive_job_result;
  FOREVER
  LOOP
      job_transfer;
  FOREVER
  LOOP
     process normal process at low priority
  FOREVER
ENDCONCURRENT
END. {TMDA:Two-Mode Dynamic Algorithm}
```

# Appendix C

# Solve ERIDA with communication overhead by Gaussian elimination

```
Program VERIOH;

{* August 27, 1987 *}
{* ERIdA, with communication Overhead, homogeneous *}
{* threshold=1 *}

type
    d2 = array [0..30,0..30] of real;
    d1 = array [0..30] of real;

{*********************************************************************}
{*                      SOLVING LINEAR EQUATIONS                     *}
{*                        (courtesy to C. Yuan)                      *}
{*********************************************************************}
procedure leqn(n:integer;p: d2; var x: d1);

var a : d2;
    b : d1;

{*********************************************************************}
{*                           Rearrange                               *}
{*********************************************************************}
procedure rearr;
var i,j : integer;

begin
   b[0]:=1.;
   for i:=1 to n-1 do
```

```
      b[i]:=0.;
   for i:=0 to n-1 do
    begin
     for j:=0 to n-1 do
     begin
         a[i,j]:=p[i,j];
     end;
    end;
end;

{*********************************************************************}
{*                          Switch Row                               *}
{*********************************************************************}
procedure switch_row(row_no:integer);
var old_no,i : integer;
    dummy : real;

begin
   old_no:=row_no;
   repeat
      row_no:=row_no+1;
   until a[row_no,old_no]<>0;
   for i:=old_no to n-1 do
   begin
      dummy:=a[old_no,i];
      a[old_no,i]:=a[row_no,i];
      a[row_no,i]:=dummy;
   end;
   dummy:=b[old_no];
   b[old_no]:=b[row_no];
   b[row_no]:=dummy;
end;


{*********************************************************************}
{*                      Eliminate                                    *}
{*********************************************************************}
procedure eliminate;
var i,j,k : integer;
    factor : real;

begin
   for k:=0 to n-2 do
   begin
      if a[k,k]=0. then switch_row(k);
      for i:=k+1 to n-1 do
      begin
         factor:=a[i,k]/a[k,k];
         for j:=k+1 to n-1 do
```

```
            a[i,j]:=a[i,j]-a[k,j]*factor;
          b[i]:=b[i]-b[k]*factor;
       end;
   end;
end;

{*****************************************************************}
{*                          Get X                                *}
{*****************************************************************}
procedure get_x;
var i,j : integer;
    sum : real;

begin
   x[n-1]:=b[n-1]/a[n-1,n-1];
   for i:=2 to n do
   begin
      sum:=0.;
      for j:=1 to i-1 do
          sum:=sum+a[n-i,n-j]*x[n-j];
      x[n-i]:=(b[n-i]-sum)/a[n-i,n-i];
   end;
end;

begin
   rearr;
   eliminate;
   get_x;
end;
{*****************************************************************}
var
   pp                          :d2;      {Markov matrix}
   xx                          :d1;      {steady state prob}
   iteration,load_index        :integer; {iteration index}
   hh,ii,jj                    :integer; {iteration index}
   station_no                  :integer; {total number of stations}
   q_size                      :integer; {queue depth}
   rho                         :real;    {traffic intensity}
   Et_ERIDA                    :real;    {mean response time of
                                          ERIDA w/ OH}
   En_ERIDA                    :real;    {mean queue length of
                                          ERIDA w/ OH}
   inilam,lambda               :real;    {arrival rate}
   Eff_lam                     :real;    {effective arrival rate}
   Et_MM1                      :real;    {M/M/1 mean response time}
   deck,deck1                  :text;    {output files}
   lam,mu,lamw,mu_lan          :real;
   psi,alpha,beta              :real;
   gam1,gam2,gamma             :real;
```

```
begin
   q_size:=30;                          {initialized}
   station_no:=10;                      {initialized}
   lambda:=20.8333;
   inilam:=lambda/10.;
   lam:=inilam;
   lamw:=lambda/10;
   mu:=lambda;
   mu_lan:=(10/20)*mu;
   assign(deck,'a:verioh20.q');         {overhead=2.0}
   rewrite(deck);
   assign(deck1,'a:MM1');
   rewrite(deck1);

     for ii:= 0 to q_size do
       xx[ii]:=0.;

     xx[0]:=1-(lam/mu);
     xx[1]:=(1/2)*(1-(lam/mu))*(lam/mu);
     xx[2]:=(1/2)*(1-(lam/mu))*(lam/mu);

for load_index := 1 to 9 do
begin
     lam := load_index*inilam;

       for ii:=0 to q_size do
         begin
           for jj:=0 to q_size do
             pp[ii,jj]:=0.;
         end;

       for ii:= 0 to q_size do
           pp[0,ii]:=1.;

for iteration:=1 to 15 do               {iteration for loop}
begin psi:=1.;
     for jj:=1 to (station_no-1) do
          psi:=psi*(xx[0]+xx[1]+xx[2]);
     alpha:=mu*psi;
     beta:=lamw*(1-psi);
     gam1:=(station_no-1)*(xx[0]*lamw+xx[1]*mu);
     gam2:=(station_no-1)*(1-(xx[0]+xx[1]+xx[2]));
     gamma:=mu+gam1/(1+gam2);
pp[1,0]:=-1*(lam+beta);
pp[1,1]:=alpha;
pp[2,0]:=lam;
pp[2,1]:=-1*(mu+lam);
pp[2,2]:=mu_lan;
```

```
pp[2,3]:=gamma;
pp[3,0]:=beta;
pp[3,1]:=mu-alpha;
pp[3,2]:=-1*(mu_lan+lam);
pp[3,4]:=gamma;

for hh:=1 to ((q_size-4)div 2) do
begin
     pp[(hh-1)*2+4,(hh-1)*2+1]:=lam;
     pp[(hh-1)*2+4,(hh-1)*2+3]:=-1*(gamma+lam);
     pp[(hh-1)*2+4,(hh-1)*2+4]:=mu_lan;
     pp[(hh-1)*2+4,(hh-1)*2+5]:=gamma;
     pp[(hh-1)*2+5,(hh-1)*2+2]:=lam;
     pp[(hh-1)*2+5,(hh-1)*2+4]:=-1*(gamma+mu_lan+lam);
     pp[(hh-1)*2+5,(hh-1)*2+6]:=gamma;
end;
     pp[30,27]:=lam;
     pp[30,29]:=-1*(gamma+lam);
     pp[30,30]:=mu_lan;
     leqn(31,pp,xx);
 end;                                    {end iteration for loop}
En_ERIDA:=0;

for hh:=1 to (q_size div 2) do          {exepected # of customers}
   En_ERIDA:=En_ERIDA+hh*xx[2*(hh-1)+1]+hh*xx[2*(hh-1)+2];
   Eff_lam:=(1-xx[q_size])*lam;         {effective arrival rate}
   Et_ERIDA:=En_ERIDA/Eff_lam;          {ERIDA mean response T}
   rho:=lam/mu;                         {traffic intensity}
   Et_MM1:=(1/mu)/(1-rho);              {M/M/1 mean response T}
   writeln(rho:10:4,Et_ERIDA:10:3);
   writeln('    xx[q_size]=',xx[q_size]:10:3);
   writeln(deck,rho:10:4,Et_ERIDA:10:3);
   writeln(deck1,rho:10:4,Et_MM1:10:3);
end;                                    {of for}
close(deck);
close(deck1);
end.
```