Yale University
Department of Computer Science

Adaptive Load Sharing in Heterogeneous Systems

Ravi Mirchandaney, Don Towsley and John Stankovic

YALEU/DCS/TR-586
November 1987

This work has been supported in part by the National Science Foundation under grant ECS-8406402 and by RADC under contract RI-44896X.

Adaptive Load Sharing in Heterogeneous Systems*

Ravi Mirchandaney[†]
Dept. of Computer Science
Yale University
New Haven, CT 06520-2158

Don Towsley
Dept. of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

John A. Stankovic
Dept. of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

November 1987

Abstract

In this paper, we study the performance characteristics of simple load sharing algorithms for heterogeneous distributed systems. We assume that non-negligible delays are encountered in transferring jobs from one node to another and in gathering remote state information. We analyze the effects of these delays on the performance of two algorithms called Forward and Reverse. We formulate queueing theoretic models for each of the algorithms operating in heterogeneous systems under the assumption that the job arrival process at each node is Poisson and the service times and job transfer times are exponentially distributed. The models are solved using the Matrix-Geometric solution technique. Many interesting tests are conducted on the models: e.g., the effects of varying thresholds, the impact of changing the probe limit, the impact of biased probing, and determining the optimal response times over a large range of loads and delays. Wherever relevant, the

*This work was supported, in part, by the National Science Foundation under grant ECS-8406402 and by RADC under contract RI-44896X.

[†]This work was performed while the author was at the University of Massachusetts.

results of the models are compared with the $M/M/1$, Random Assignment, and the $M/M/K$ models.

1 Introduction

Load sharing (balancing) for distributed systems has been an active research area for several years. Many of these efforts have demonstrated that simple load sharing algorithms produce considerable performance improvement [4], [9], [3], [1], [12], [11], [2], [10]. The reader is referred to [5] for a comprehensive survey of load sharing research.

While the previous research provides considerable insight into various aspects of load sharing, the problem of communication delays and its effects on load sharing had not been investigated in great detail until [6]. In that paper, we examined the problem of communication delays in load sharing for homogeneous systems. In this paper, we extend our analysis to include heterogeneous distributed systems. Heterogeneity in distributed systems can arise primarily in the following two ways: In the first instance, the nodes in the system may be identical with regard to their processing capabilities and speeds, but, the rate at which external jobs arrive at nodes may be different. In the second instance, nodes although functionally identical, may process jobs at different rates (in such systems, the arrival rates of jobs at various nodes may also differ). We refer to these kinds of heterogeneous systems as **type-1** and **type-2** systems, respectively. Some of the observations we have made during the course of this study are as follows: In **type-1** systems, Forward probing becomes more sensitive to probe limit as the system becomes more heterogeneous while the opposite effect is seen for Reverse probing. For **type-1** systems, it was seen that Reverse probing could perform much worse than the Random scheduling algorithm under several conditions but Forward probing was always better. In the case of **type-2** systems, it was seen that selecting the appropriate operating threshold was very critical for the slow nodes but relatively less important for the fast nodes. Details of these and other results are presented in Section 4 of this paper.

The remainder of this paper is organized as follows: In Section 2, we provide a description of the system architecture and two simple load sharing algorithms, called Forward and Reverse. Section 3 contains the formulation of the Markov process corresponding to the Forward algorithm and its Matrix Geometric solution. The details of the analysis corresponding to the Reverse probing algorithm are not presented because of space limitations and the fact that the general solution technique is similar to the Forward case. In Section 4, we describe the important results of this research and summarize our work in Section 5.

2 System Architecture and Load Sharing Algorithms

We assume that the system under consideration is comprised of C distinct classes of nodes, based upon the external arrival rates of jobs and/or the processing speeds of nodes. All the nodes in a particular class are assumed to be identical. In addition, we assume that each node possesses a network controller which is responsible for most of the overhead associated with load sharing activities. Thus, although the CPU overhead due to load sharing is assumed to be negligible, job transfers and state update messages are delayed because of the network controller. It is the effects of these delays that we study in this paper.

2.1 Load Sharing Algorithms

The two algorithms that we have analyzed are called Forward and Reverse. They are described in the following paragraphs. Each algorithm is provided with thresholds T_c, $c = 1, \ldots, C$, one for each class of nodes.

- **Forward:** The algorithm executes at every node and is activated each time a local job arrives at a node. If the number of jobs at this node (including the job currently being executed) is greater than $T_c + 1$ (the node in question belongs to class-c), an attempt is made to transfer the newly arrived job to another node. A finite number, L_p, of nodes is probed at random and in parallel to determine a placement for the job. A probed node responds positively if the number of jobs it possesses is less than $T_d + 1$ (the probed node belongs to class-d) and it is not already waiting for some other remote job. If more than one node responds positively, the sender node transfers the job to one of these respondents, chosen at random. If none of the probed nodes responds positively, i.e., this probe was unsuccessful, the node waits for another local arrival before it can probe again.

- **Reverse:** This algorithm also executes at every node. It is activated every time a job completes at a node, and the total number of jobs at the node is less than $T_c + 1$, and the node is not already waiting for a remote job to arrive. If so, the node probes a subset of size L_p remote nodes at random to try and acquire a remote job. Only nodes that possess more than $T_d + 1$ jobs, (the probed node belongs to class-d) are allowed to respond positively. If more than one node can transfer a job, the probing node chooses one of these nodes at random from which it requests a job.

Implicit in the above algorithm descriptions is the assumption that a node randomly probes other nodes, i.e., a class-c node is as likely to be probed as a class-d node. In general, this assumption may be relaxed for potential performance benefits. For instance, if a node possessed information about the membership and load of each class, it can bias probing towards one or more classes. Thus, at each probe, class-c nodes may be selected with probability f_c (with nodes in a class being selected in an equiprobable manner). In the general case, $f_{c,d}$ is the probability that a probing node of class c will probe class d nodes, $\sum_{d=1}^{C} f_{c,d} = 1$, $\forall c$.

In the algorithms described above, it is assumed that probing takes zero time. This is based upon the assumption that probes are much smaller entities than are jobs. Thus, the overhead for processing a probe is much smaller than for jobs. Further, probes occupy much less of the communication bandwidth than jobs. Thus, the entire delay is assumed to occur during actual job transfer. This is a reasonable assumption since it has been demonstrated in [7] and [5], that as long as the ratio of job transfer times to probe transfer times is sufficiently large (≥ 20), the system essentially behaves as if probes take zero time.

3 Mathematical Analysis

It is assumed that the job arrival process at each class-c node is Poisson, with parameter λ_c. Also, the service times and job transfer times for class-c nodes are assumed to be exponentially distributed, with means $1/\mu_c$ and $1/\gamma_c$, respectively. In our study, we have assumed that γ_c is independent of the class of nodes under consideration. Thus, for the remainder of this paper, $\gamma_c = \gamma$, $\forall c$. The job transfer time includes the time between the initiation of a transfer from a node and the successful reception of the job at the destination node. Last, jobs are assumed to be executed on a First-Come-First-Served (FCFS) basis at each node.

Let $N_t^{(c,i)}$ be the number of jobs at node i in class-c at time t and $J_t^{(c,i)}$ be the probe state of the ith class-c node, at time t. The probe state indicates whether the node is probing or being probed, etc. For example, in a system which has M_c nodes in class-c, the instantaneous state of class-c nodes can be represented by the $2M_c$-tuple

$$(N^{(c)}, J^{(c)}) \overset{\text{def}}{=} (N_t^{(c,1)}, N_t^{(c,2)}, ..., N_t^{(c,M)}; J_t^{(c,1)}, J_t^{(c,2)}, ...J_t^{(c,M_c)})$$

The instantaneous state of the entire network (over all the node classes) may be represented by the following:

$$(N, J) \overset{\text{def}}{=} ((N^{(1)}, J^{(1)}), (N^{(2)}, J^{(2)}), ...(N^{(C)}, J^{(C)})).$$

The probe state $J_t^{(c,i)}$ can be defined in a way that the behavior of the system can be described by a Markov chain. However, this Markov chain has a very large state space and is extremely difficult to solve, even for moderately sized systems. Consequently, we decompose the model in the following way: Each node is modeled independently of the others. Because there are C different node classes, this results in C distinct single node queueing models. Further, we assume that each node can be modeled as a Markov chain. The interactions between the nodes which result in job transfers for the purpose of load sharing in the distributed system, are modeled by means of modifications to the arrival and/or departure process at each node. These interactions will be described in detail later in this section.

We conjecture that the method of decomposition is asymptotically exact as the number of nodes in each class tends to infinity. Actual simulation results indicate that there exists very good agreement between the model and simulations even when the systems are of relatively small size (10-15 nodes in each class). We expect the approximation to be even better for larger systems.

The analysis is performed using the Matrix-Geometric solution technique [8], which yields an exact solution of the model for each node. Due to space limitations, we will only briefly describe the analysis pertaining to the Forward algorithm. For complete details on the analysis, the reader is referred to [5].

Figure 1 represents the state diagram for the Forward probing algorithm operating at a single node of class-c using an arbitrary threshold T_c. The state of the node is represented by a tuple $(N_t^{(c)}, J_t^{(c)})$, where $N_t^{(c)}$ is the number of jobs at a node and $J_t^{(c)}$ indicates if the node is being probed or not. Specifically, $J_t^{(c)} = 0$ if the node is not being forward probed and $J_t^{(c)} = 1$ if it is being forward probed.

We define

$$y(n,j) = \lim_{t \to \infty} P(N_t^{(c)} = n,\, J_t^{(c)} = j),\ 0 \leq n,\ 0 \leq j \leq 1,$$

$$\mathbf{p}_n^{(c)} = (y(n,0), y(n,1)),\ 0 \leq n,$$

$$\vec{\mathbf{p}}^{(\mathbf{c})} = (\mathbf{p}_0^{(c)}, \mathbf{p}_1^{(c)}, \mathbf{p}_2^{(c)}, \ldots.. \mathbf{p}_i^{(c)}, \ldots.).$$

If the Markov process $(N^{(c)}, J^{(c)})$ is ergodic then $\vec{\mathbf{p}}^{(\mathbf{c})}$ is its steady state probability vector satisfying $\vec{\mathbf{p}} Q_F^{(c)} = 0$, where $Q_F^{(c)}$ is the infinitesimal generator of this Markov process. Here $Q_F^{(c)}$ has the following structure

$$Q_F^{(c)} = \begin{bmatrix} B_{00}^{(c)} & B_{01}^{(c)} & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots \\ A_2^{(c)} & B_{11}^{(c)} & B_{01}^{(c)} & \cdots & 0 & 0 & 0 & 0 & \cdots \\ 0 & A_2^{(c)} & B_{11}^{(c)} & \cdots & 0 & 0 & 0 & 0 & \cdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \\ 0 & 0 & 0 & \cdots & B_{11}^{(c)} & B_{01}^{(c)} & 0 & 0 & \cdots \\ 0 & 0 & 0 & \cdots & A_2^{(c)} & A_1^{(c)} & A_0^{(c)} & 0 & \cdots \\ 0 & 0 & 0 & \cdots & 0 & A_2^{(c)} & A_1^{(c)} & A_0^{(c)} & \cdots \\ 0 & 0 & 0 & \cdots & 0 & 0 & A_2^{(c)} & A_1^{(c)} & \cdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

with exactly $T_c - 1$ columns of $(B_{01}^{(c)}, B_{11}^{(c)}, A_2^{(c)})$. There is a unique generator associated with each class of nodes in the system.

In the subsequent discussion, h_c is the probability that a class-c node is unsuccessful in finding an assignment for a spare job in response to a set of forward probes. Thus, $\overline{h}_c = 1 - h_c$ is the probability that at least one of the probed nodes will accept a remote job. The effect of a node sending a set of unsuccessful forward probes when it surpasses $T_c + 1$ is represented by the transition $\lambda_c h_c$. When the node receives a forward probe and $0 \leq n \leq T_c$, it makes a transition from $(n, 0)$ to $(n, 1)$, i.e., it awaits the transfer of a job from another node. The rate at which a class-c node receives forward probes is denoted by α_c.

The behavior of a node under Reverse is described by a similar Markov chain. The details of this chain and its analysis can be found in [5].

3.1 Matrix-Geometric Solution

Neuts [8] examined Markov processes with such generators and determined the conditions for positive recurrence when the infinitesimal generator $A^{(c)} = A_0^{(c)} + A_1^{(c)} + A_2^{(c)}$, corresponding to the geometric part of the Markov Process, is irreducible. However, for our problem, A is lower triangular and reducible. In such cases, the stability criterion has to be determined explicitly.

Consider the non-linear matrix equation

$$A_0^{(c)} + R^{(c)} A_1^{(c)} + [R^{(c)}]^2 A_2^{(c)} = 0$$

such that $R^{(c)}$ is its minimal non-negative solution. It can be shown that $R^{(c)}$ is lower triangular, given the structure of $A_0^{(c)}, A_1^{(c)}$ and $A_2^{(c)}$ [8]. Also, $R^{(c)} = [r_{i,j}^{(c)}]$ for Forward probing can be written as follows:

$$\begin{aligned}
r_{1,1}^{(c)} &= \lambda_c h_c / \mu_c \\
r_{2,2}^{(c)} &= \frac{\theta_c + \gamma - ((\theta_c + \gamma)^2 - 4\mu_c \lambda_c h_c)^{1/2}}{2\mu_c} \\
r_{1,2}^{(c)} &= 0 \\
r_{2,1}^{(c)} &= \frac{\gamma}{\theta_c - (r_{1,1}^{(c)} + r_{2,2}^{(c)})\mu_c}
\end{aligned}$$

where $\theta_c = \lambda_c h_c + \mu_c$. It can be shown that the stability criterion for the Forward probing algorithm is

$$\lambda_c h_c < \mu_c.$$

Intuitively, the stability criterion implies that the rate of processing jobs (including the ones that are sent out of this node) be greater than the total arrival rate of jobs into this node. Thus, on the average, whenever there are more than $T_c + 1$ jobs at a node, the process drifts towards the boundary specified by the threshold T_c.

We know from Neuts [8] that

$$\mathbf{p}_i^{(c)} = \mathbf{p}_{T+1}^{(c)} R_{T_c+1-i}^{(c)}, \forall i \geq T_c + 1$$

Thus,

$$\sum_{i \geq T_c+1} \mathbf{p}_i^{(c)} = \mathbf{p}_{T_c+1}^{(c)} (I - R^{(c)})^{-1}$$

Also,

$$[\sum_{i=0}^{T_c} \mathbf{p}_i^{(c)} + \mathbf{p}_{T_c+1}^{(c)} (I - R^{(c)})^{-1}]e = 1$$

where I is the identity matrix, e is a column vector of all 1's, and $R^{(c)}$ is the minimal solution of

$$A_0^{(c)} + R^{(c)} A_1^{(c)} + (R^{(c)})^2 A_2^{(c)} = 0$$

with $R^{(c)} \geq 0$, and the spectral radius of $R^{(c)}$ is less than one.

The expected number of jobs at a node of class-c, $E[N_c]$, and the expected response time of a job of the same class, $E[D_c]$, are given by the following expressions:

$$\begin{aligned} E[N_c] &= \sum_{i \geq 1} i \, \mathbf{p}_i^{(c)} \, e \\ &= \mathbf{p}_{T_c+1}^{(c)} (I - R^{(c)})^{-2} e + T_c * [\mathbf{p}_{T_c+1}^{(c)} (I - R^{(c)})^{-1} e] + \sum_{i=1}^{T} i \mathbf{p}_i^{(c)} e \\ E[D_c] &= \frac{(E[N_c] + \frac{(Flow-In(c))}{\gamma})}{\phi_c} \end{aligned}$$

where ϕ_c is the throughput of class-c nodes and $Flow - In(c)$ is the flow into a node of remote jobs due to forward or reverse probes, depending upon the algorithm being used.

Further, the average response time of jobs over all classes in the system is given by:

$$E[D] = \frac{1}{\Phi} \sum_{c=1}^{C} \phi_c E[D_c]$$

where

$$\Phi = \sum_{c=1}^{C} \phi_c$$

is the throughput of the system over all classes.

3.2 Computational Procedure

We have parameterized the node models for each class in terms of two unknown parameters, h_c and α_c, $c = 1, \cdots, C$. In this section we derive a set of nonlinear equations that these parameters must satisfy. We then briefly describe the computational procedure used to determine these unknowns. A similar procedure is used for Reverse probing.

We define

- $FFRO_c$: Flow rate out of jobs from a class-c node, as a result of forward probes made by it.
- $FFRI_c$: Flow rate of jobs into a class-c node, as a result of forward probes made to it by other nodes in the network.
- $Probe - Out_c$: The rate of successful forward probes made by a class-c node.

These variables have the following expressions,

$$\begin{aligned} FFRI_c &= \overline{x}_c \sum_{d=1}^{C} FFRO_d \, f_{d,c} \, , \forall c, \\ FFRO_d &= Probe - Out_d / \overline{h}_d, \\ Probe - Out_d &= \lambda_d \overline{h}_d \sum_{i > T_d + 1} \mathbf{p}_i^{(d)} \, e. \end{aligned}$$

Here x_c is the probability that a class-c node is unable to accept a remote job and is given by

$$x_c = \sum_{i \leq T_c} \mathbf{p}_i^{(c)} \, [01]^T + \sum_{i > T_c} \mathbf{p}_i^{(c)} e.$$

Also, $\overline{x}_c = 1 - x_c$.

$h_d = (r_d)^{L_p}$ is the probability that L_p forward probes made by a class-d node are unsuccessful and $\overline{h}_d = 1 - h_d$ is the probability that at least one of the probes is successful, where $r_d = \sum_{c=1}^{C} f_{d,c} x_c$, is the probability that a forward probe made by a class-d node is unsuccessful and $\overline{r}_d = 1 - r_d$ is the probability of a successful probe.

$FFRI_c$ can also be expressed as follows:

$$FFRI_c = \alpha_c \overline{x}_c$$

In order to determine α_c, the the rate at which a class-c node receives forward probes, we can rewrite the above equation as:

$$\alpha_c = FFRI_c / \overline{x}_c.$$

We present a brief description of the solution algorithm. Details can be found in [6] Initially, it is assumed that the values for h_c and α_c are known for all classes and the model is solved using these values. In a typical step, a model solution is used to derive new values for h_c and α_c, for all classes, and a new solution is computed. This procedure is continued until the iterative procedure is determined to have converged. We have observed from experiments that the solution was insensitive to the initial values chosen for the unknown quantities. Consequently, we conjecture that there exists a unique solution to the models. Further, the number of iterations required for convergence was usually small, ranging between 20 and 40.

4 Performance Comparisons

In this section, we study the performance characteristics of the algorithms in the two main types of heterogeneous systems, i.e., **type-1** systems where the external job arrival rates at nodes may differ, and **type-2** systems, where the nodes themselves may have different processing rates. Even though the analysis presented in the previous section is valid for an arbitrary number of node classes, we restrict our study to systems with two distinct classes. The reason for this restriction is the fact that the computational effort required to study heterogeneous systems grows at an exponential rate. For example, a brute force search of the optimal thresholds for a set of parameters is easily seen to grow asymptotically as $O(T^C)$, where $0 \ldots T$ is the range of thresholds and C is the number of classes under consideration. For the subsequent discussion, when we refer to the quantities in both classes, we use the following system: For example, (T_1, T_2) refer to the operating thresholds at nodes of the two classes. All other parameters are represented in a similar fashion.

Wherever relevant, we compare the algorithm's performance to that of the Random assignment, the no-load-balancing or $M/M/1$ results, and the $M/M/K$ system or an appropriate lower bound for heterogeneous systems of **type-2** (called LB_2, which is described in [9]). In short, this algorithm ensures that the fast nodes in the system are utilized before the slower ones and if jobs need to be transferred from a slow node to a fast one, this transfer occurs with zero cost. The Random assignment algorithm has been described elsewhere [6] but, briefly, a class-c node that has a local arrival and its total queue length is $\geq T_c + 2$ transfers the new arrival to another node chosen at random. In this algorithm too, as in Forward and Reverse, it is possible to bias the selection of classes, with nodes in a class being selected randomly. In all the subsequent studies in this paper we use a probe limit, $L_p = 2$ unless mentioned otherwise[1]. Further, $S_c = 1/\mu_c$ is the mean service time for jobs executing at class-c nodes. For **type-1** systems, $S_1 = S_2 = S = 1$ unit. Also, ρ_1 and ρ_2 are the loads for nodes of class-1 and class-2, respectively.

[1] The next section substantiates our assumption that a probe limit of 2 is adequate.

4.1 Type-1 Heterogeneous Systems

In this study, $\rho_1 = 0.9$, is kept fixed for all the tests conducted. However, ρ_2 is varied between 0.1 and 0.7, in steps of 0.2. Thus, we examine systems in which the degree of heterogeneity (in this case the ratio of loads of the two classes) is varied over a large range. Although we have conducted tests for all four values of ρ_2, we will present only those results which are most interesting. In general, we emphasize those results with $\rho_2 = 0.1$ and 0.7, representing high and low degrees of heterogeneity, respectively. Wherever relevant, we also comment on the results of the other two loads of 0.3 and 0.5.

As might be imagined, the number of parameters that it is possible to vary in the study of these algorithms is very high. We will try to limit this study to what we believe are its most interesting aspects. These items are presented in the following list and they all consider varying the job transfer delay times.

- Validation of analytical models with simulations
- Effects of varying the probe limit L_p
- Effects of varying (T_1, T_2)
- Effects of biasing the probing probabilities, $(f_{c,d}\,,\ 1 \leq c, d \leq 2)$
- Performance of Biased Random
- Optimal response times under a large range of job transfer delays

Validation of models with simulations

As mentioned in Section 3, the solution technique we have used is only an approximation for finite sized multiple class systems, which is conjectured to be exact for infinitely large systems. Consequently, we have conducted simulation studies of systems executing the load sharing algorithms and compared these with the results of the analytical models. These comparisons allow us to determine the range of parameters for which our analysis is valid. It was seen that when the classes contained at least 15 nodes each, the simulation results were very close to the analytical models, with errors of about 3%. Further, we also computed the 95% confidence intervals which were within 2% (for futher details, please see [5]). We conjecture that the results become more accurate as the system size increases.

Effect of varying L_p, the probe limit

In this study, we consider the effect of varying the probe limit on the performance of the two algorithms. In general, probes can generate processing overheads which may degrade system performance and in real systems, this overhead would need to be traded off against the response time improvements that increased probing may offer. Figures 3 and 4 depict the results of varying L_p between 1 and 10 for two different sets of loads for the Forward probing algorithm. In Figure 3, (ρ_1, ρ_2) =(0.9,0.1) and in Figure 4, (0.9,0.7). The three curves in each graph correspond to different values of delays, i.e., $0.1S, S$ and $10S$. From Figures 3 and 4 we can make the following observations.

Increasing the probe limit beyond 3 or 4 appears to provide little or no improvement (3-4%) in response times. When the degree of heterogeneity is large, as in the case of Figure 3 the system response time is less sensitive to L_p than in a more balanced system. From Figure 3, it is easily seen that going beyond $L_p = 2$ does not provide significant performance benefit (at most 3-5%). On the other hand, in Figure 4, where at delays of $0.1S$ and perhaps even at S, $L_p = 3$ seems the proper limit (with about 10% gain over $L_p = 2$). The main reason for this behavior is the following: When systems are very highly unbalanced, a forward probe made by a node is very likely to find a recipient (even when the class selection is equiprobable). However, as the degree of heterogeneity decreases, a probing node needs to work harder to find a placement for a spare job. Further, at very high delays $\geq 10S$, the performance is less sensitive to L_p than at lower delays. This is because of reduced load sharing at high delays, caused by the high operating thresholds.

From our tests on systems with $(\rho_1, \rho_2) =$ (0.9,0.3) and (0.9,0.5) in connection with L_p, (results not presented here), we have seen that a gradual change occurs in the behavior. Thus, (0.9,0.3) behaves more like (0.9,0.1) than (0.9,0.7) while the behavior of (0.9,0.5) approaches (0.9,0.7), as might be expected. Further, we have seen that when ρ_1 is smaller, for example, 0.8 and less, there appears to be less sensitivity to probe limit. Again, this is because a probing node is more likely to find a recipient for a spare job, because the net load on the entire system is lower.

Figures 5 and 6 depict the results of identical tests performed on the Reverse probing algorithm. The loads indicated in Figures 5 and 6 are (0.9,0.1) and (0.9,0.7), respectively. From Figures 5 and 6, we can make the following observations:

Reverse probing appears more sensitive to probe limit, particularly when the degree of heterogeneity is large, as in the tests corresponding to Figure 5. This is because a probing node is less likely to find a spare job since class-2 nodes are very lightly loaded. Thus, a probing node needs to work harder to find a spare job. In general, it would appear that three or even four probes offer significant performance benefits. When delays are larger, the effect of increased probe limit is not as pronounced, except when ρ_2=0.1. This effect is due to the high thresholds which reduce load sharing in the first instance. Also, very few jobs execute at class-2 nodes. Thus, after an unsuccessful set of reverse probes, the node waits for a long time before another probe can be made.

Effect of varying thresholds (T_1, T_2)

The motivation behind these tests is to determine the sensitivity of thresholds in either class. This may provide some clues about how critical it is to choose the proper thresholds. Initially, the optimal threshold pair, (T_1, T_2) is determined for the loads and delays of interest. Then, while keeping T_2 fixed at its optimal value, we vary T_1 over a large range. Next, we keep T_1 fixed at its optimal value and vary T_2. Figures 7 and 8 show the effects of varying T_1 and T_2, respectively, for load (0.9,0.1), and delays $0.1S, S$, and $10S$. Also shown in the graphs are the results from the $M/M/1$ and the perfect load balancing at zero cost $M/M/20$ systems.

From Figures 7 and 8, we can see that the performance of the algorithm is very sensitive to T_1, particularly when the delays are $\leq S$. In these cases, $T_1 = 0$, for optimal performance. For a delay of $10S$, the optimal value for T_1 is 4. In comparison, the performance is insensitive to changes in T_2, showing an almost flat behavior in Figure 8. This is because the nodes of class-2 are so lightly loaded that increasing the threshold does not substantially increase load sharing. Even when T_2 is low (=1), there is high probability that a class-2 node is at or below its threshold (because $\rho_2 = 0.1$). Results of tests conducted when $\rho_2 = 0.3$, 0.5 and 0.7 (figures not presented here), show that the effects varying T_2 gradually approach that of varying T_1.

We have conducted similar tests for the Reverse probing algorithm and the results of these tests have shown us behavior similar to that of the Forward probing algorithm. Consequently, we will not present the results of these tests.

Effect of biased probing

In all the studies conducted thus far in this research, a probing node was equally likely to choose a class-1 or a class-2 node. Using our earlier notation, $f_{c,d} = 0.5, \forall c, d$ (this is asymptotically exact as the number of nodes in each class grows equally large). When $(\rho_1, \rho_2) = (0.9, 0.1)$, a node trying to make a forward probe is more likely to find a placement at a class-2 node, while a node making a reverse probe is more likely to find a spare job with a class-1 node. The question then is to determine if and when biased probing can make a substantial difference in performance and reduce the probe limit in the process. To conduct this part of our study, we have adopted the following procedure.

Initially, $f_{1,1} = f_{2,1} = f_1$, and $f_{1,2} = f_{2,2} = f_2$. Also, $f_1 + f_2 = 1$, at all times. Recall that f_c is the probability that a class-c node will be selected by a probing node of any class. Figures 9 and 10 depict the results of the experiments conducted on the effects of biasing the probes in the study of Forward probing. From Figure 9 ($(\rho_1, \rho_2) = $ (0.9,0.1)), we see that the equiprobable selection of classes is only marginally worse (at most 5%) than the optimal fractions. However, from Figure 10, it is clear that when the classes are more or less balanced, (0.9,0.7), it is best to have unbiased (or close to it) selection of classes for Forward probing in heterogeneous systems.

Finally, we study the effect of biased probing upon the Reverse probing algorithm. Here, we will only depict one set of curves, for load (0.9,0.1), because this is quite different from its Forward probing counterpart. Figure 11 shows the results from these tests. From the curves shown in the figure, we can see that there is between 30-40%

improvement by correctly biasing the probabilities, when the delays are $\leq S$. The effect is less pronounced for higher delays. This result is explained by the fact that Reverse probing is much more sensitive to probe limit when the imbalance in systems is large. Thus, each probe is very important and when $L_p = 2$, biasing appears to provide substantial benefits. In any event, as the degree of heterogeneity is decreased, the effect of biased probing becomes less prominent, as was seen in the case of Forward probing.

Thus, from our studies concerning biased probing, we are able to reach the following conclusions: Biasing appears to have a second or third order effect on response times, except in the case of Reverse probing in highly heterogeneous systems. We have seen from the graphs on biasing that choosing incorrectly biased probabilities has a much more detrimental effect on performance than equiprobable selection.

Performance of Biased Random

In the description provided for the Random assignment algorithm, it was stated that any node in the system was equally likely to be a recipient for a transferred job (Unbiased Random). Thus, in a system with two classes, a node executing the "Unbiased" Random algorithm will transfer approximately half its spare jobs to either class of nodes (in the limiting case as the number of nodes in each class become equally large). When the degree of heterogeneity is high, there is an impact of biasing the probabilities when the job transfer delays are high ($10S$). The improvement in that case over the unbiased Random is about 20%. However, for moderate delays, the difference is almost nil (delay=S) and the unbiased version actually performs better by about 7% at delay = $0.1S$. When the systems become more balanced, unbiased Random is clearly superior to any biased Random (except at delay = $10S$, where it is worse by about 4%). Because the gain due to biasing does not appear very significant, all further references to Random will imply an "Unbiased" version of that algorithm.

Optimal Normalized Response Times

In this study, we vary the job transfer delay from $0.01S$ to $100S$ and determine the optimal performance achieved by the two probing algorithms as well as the Random assignment algorithm. The results of these experiments are shown in Figures 12 and 13. The Y-coordinate depicts the optimal response time normalized by the corresponding $M/M/1$ response time. Thus, the smaller the value on a curve, the greater is the gain over the corresponding $M/M/1$ value. We only depict two of the four pairs of curves because of size limitations of the graphs. For more detailed figures, the reader is referred to [5]. The reason we are interested in comparing the probing algorithms against the Random algorithm is because it provides information on the range of parameters over which probing is useful.

Figure 12 depicts the results for the Forward probing and Random assignment algorithms. From the figure, we are able to infer the following facts: There is performance benefit from probing when delays are less than about two to three times the average service time. For higher values of delays, the curves for the probing and random algorithms become almost indistinguishable. This means that the delays are so large that there is almost no correlation between the state of a remote node when a probe is made and the state of that node after a job is transferred to it. While probing may cease to be useful at high delays, load sharing provides considerable benefits over the $M/M/1$ performance. This is very easily seen from the curves, which show that even at delays up to $100S$, there is about 7-10% performance gain. When $(\rho_1, \rho_2) = (0.9,0.1)$, the net load over all the nodes in the system is 0.5. The greater the imbalance in the systems, the larger is the gain over the corresponding $M/M/1$ value. The greater the imbalance in the systems, the smaller is the difference between the random and probing algorithms. This effect is more pronounced for lower delays as can be seen from Figure 12, where the pairs of curves appear to be more separated at higher values of ρ_2. This may be explained as follows: When both classes of nodes are busy, there is a greater likelihood that a random assignment will result in a transfer to an already busy node.

Figure 13 depicts the results of similar tests conducted on the Reverse probing algorithm. From Figure 13, we are able to see that the behavior for Reverse probing is quite different from that of Forward probing (Figure 12). The following points are worth noting.

When the imbalance between the classes is high, Random assignment is significantly better than Reverse probing. For example, the curves corresponding to loads (0.9,0.1) show that the Reverse curve approaches the Random curve only asymptotically. Recall that Reverse probes are sent by a node only when it is below the threshold when a job completes. Now, in very lightly loaded nodes, very few jobs actually execute. Thus, there are few opportunities to balance the load. On the other hand, the Random assignment strategy is quite effective in highly unbalanced systems, with a high probability of transferring jobs to idle nodes in class-2. As the degree of heterogeneity decreases (i.e., the systems become more alike), the gap between Reverse and Random decreases. From the curves, it is seen that Reverse actually performs better than Random when the loads are (0.9,0.7). At this point, class-2 nodes are able to perform reasonably effective load balancing and the higher loads make the Random assignment less suitable.

In the previous two graphs, we have determined the optimal response times of the algorithms under different values of total system load. In Figure 14, we depict results of tests where the total system load is kept fixed at 0.5, while changing the distribution of the load between the classes. Thus, the three sets of curves depict results for $(\rho_1, \rho_2) = (0.5,0.5)$, $(0.7,0.3)$ and $(0.9,0.1)$, i.e., from a homogeneous system to a very heterogeneous one.

From Figure 14, we are able to make the following observations: As might be expected, the benefits of load sharing over the M/M/1 response time is greater as the degree of heterogeneity is increased. Thus, the curves for (0.5,0.5) lie at the top with (0.7,0.3) next and the curves for (0.9,0.1) are the lowest in the graph. In systems with greater heterogeneity, the performance of Random is less distant from the probing curves. This is because when the classes are highly unbalanced, a Random job transfer is likely to reach an idle node, particularly when the average system load is low. The separation between the Random and probing curves is most pronounced when (ρ_1, ρ_2) is (0.5,0.5). At delay = $100S$, (0.9,0.1) generates about 10% improvement over the corresponding $M/M/1$ response time. However, for (0.5,0.5) and (0.7,0.3), delays of about $5S$ and $20S$ produce the $M/M/1$ response time, respectively.

4.2 Type-2 Heterogeneous Systems

In the set of experiments described in this section, we study systems in which the nodes in the two classes process jobs at different rates. Thus, transferring a job from a longer queue to one that is shorter may not necessarily result in lower response times (even after accounting for delays). Nodes of class-1 have the same service rate as in **type-1** systems, i.e., $\mu_1 = 1$. The service rate for nodes of class-2, on the other hand, is varied from 0.2 to 0.8. Thus, we study systems where the nodes possess different degrees of heterogeneity. The load at nodes of both classes is kept fixed at 0.8. This is for ease of comparison and we believe that this does not limit the validity of our experiments. It is relatively easy to conjecture the behavior of systems in which the above restriction may not apply.

Because of space considerations, many of the results will not be presented and others will be briefly described. Before we conducted any substantial tests for **type-2** systems, we compared the results of the analytical models with simulations. Exactly the same kind of tests were conducted as for **type-1** systems and the results were very similar. It was seen that between 10-15 nodes were needed in each class for the simulation results to be very close ($2-5\%$) to the analytical models. The 95% confidence intervals for the simulation results were computed using the Student-t tests and were seen to be within $\pm5\%$ of the sample mean.

While we have conducted all the subsequent tests on all four values of μ_2, we only present curves for $\mu_2 = 0.2$ and 0.8. The results corresponding to the other two values of μ_2 will only be described in brief unless they happen to be counter-intuitive. The reasons for adopting this approach are identical to those given in the case of **type-1** systems. Furthermore, we assume that $S_1 = S$. Further, we concluded that $L_p = 2$ is adequate for the subsequent tests.

Effect of varying (T_1, T_2)

For the loads and delays tested, (T_1, T_2) corresponding to the optimal response time is determined. Then, keeping T_2 fixed at its optimal value, we vary T_1 over a large range. Next, T_2 is varied, keeping T_1 fixed at its optimal value. In each of the figures, we present the results of the $M/M/1$ and LB_2 response times corresponding to the parameters in question. LB_2 is a lower bound for **type-2** systems that was described in [9].

The results of these tests for $\mu_2 = 0.2$ are depicted in Figures 15 and 16. From Figure 15, the behavior of the system in response to varying T_1 can be explained by the following arguments: For the most part, the optimal value of T_2 is 0. This implies that very few jobs are transferred to class-2 nodes although they are more likely to transfer jobs out. This is facilitated by the fact that the optimal value of T_1 is 2 or higher, depending upon the delay. If T_1 were lower, say 0 or 1, fewer jobs could be transferred between class-1 nodes, resulting in poor performance, as may be seen from the curves in Figure 15 (class-1 nodes are likely to be higher than T_1 for low values of T_1). However, if T_1 was set higher than a certain value (2 or 3 in the case of delays $\leq S$), this would mean that virtually no jobs may be transferred out of class-1 nodes to balance the instantaneous loads among class-1 nodes.

The results obtained by varying T_2 are presented in Figure 16. From this figure, it is observed that the performance is much more sensitive to changes in T_2 and actually gets much worse than the $M/M/1$ value for delays $\leq S$. The reasons for this behavior are as follows: As mentioned earlier, the optimal value for T_2 is 0 for delays $\leq S$ and 1 for delay $= 10S$. If T_2 is increased beyond the optimal value, there is greater likelihood that jobs will be transferred to class-2 nodes. Because the service rate of class-2 nodes is low, long queues build up and the response times get worse. In fact, there is the possibility that the class-2 queues can approach saturation, particularly in the case of delays $= 0.1S$ and S. This behavior appears to occur when T_2 is around 3 or 4 and the rate of increase of response time appears to be very high.

Figures 17 and 18 depict similar tests conducted when μ_2 was set to 0.8. Thus, jobs execute 25% longer on class-2 nodes as compared to class-1 nodes and the degree of heterogeneity between the classes is not very high. From these figures, we can see the effect of varying the thresholds is very similar to the earlier set of tests when μ_2 was equal to 0.2. The only effect is that the curves are shifted along the X-axis. The optimal value of T_1 is lower than in the previous case. For delay $= 0.1S$, $T_1 = 1$ is optimal, while for delay $= S$, it is 2. This implies that class-1 nodes are likely to be more active in load balancing. Because class-2 nodes are not much slower than class-1 nodes, they are able to accept more jobs than in the case of the earlier set of tests. The phenomenon of saturation in class-2 nodes is also observed in Figure 18. The explanation is the same as given for $\mu_2 = 0.2$. High thresholds in class-2 nodes result in many job transfers to these nodes from class-1 nodes. Because μ_2 is only 25% less than μ_1, many more jobs need to be transferred to class-2 nodes before this unstable behavior occurs. Thus, the thresholds are higher here than the case when μ_2 was 0.2.

Similar tests concerning the effects of varying T_1 and T_2 were conducted for the Reverse probing algorithm. It was seen that when T_2 is increased (particularly when the imbalance is high), the performance gets worse. This is because high values of T_2 result in more jobs transferred to class-2 nodes as well as fewer jobs transferred out of class-2 nodes to class-1 nodes (which should be the ideal strategy). Thus, class-2 nodes have a tendency become saturated if T_2 is higher than its optimal value. This effect, although less dramatic, is also seen when μ_2 is 0.8, i.e., the imbalance is not very high.

The effect of varying T_1 is seen to be much less pronounced than T_2, especially when the degree of heterogeneity is high. For delays of $\leq S$, the optimal value of T_1 is 1. Increasing T_1 beyond that results in worse performance because higher thresholds result in much less active balancing among class-1 nodes, even though they further restrict jobs being transferred to class-2 nodes (particularly since the optimal value of T_2 is 0 or 1, for the range of delays tested).

Optimal Normalized Response Times

Figure 19 depicts the results of subjecting the Forward probing algorithm to a large range of delays. The optimal response times are plotted for each value of delay along with the optimal response time generated by the Random assignment algorithm. The tests are conducted for all four values of μ_2, as indicated earlier, but results only for $\mu_2 = 0.2$ and 0.8 are depicted in the figures. From Figure 19, we are able to infer the following facts.

The benefits from load sharing are more pronounced for systems in which the degree of heterogeneity is large. For example, the curve for $\mu_2 = 0.2$ is the lowest of all curves, for most of the range of delays tested (except for very low delays around $0.05S$). The fact that the curve is the lowest signifies that the response time improvement over the corresponding $M/M/1$ value is the greatest. The curves for the Random assignment algorithm are seen to be located much higher than their respective probing curves. This means that Random performs much worse than Forward probing for certain values of parameters (at delay $= 0.1S$ and $\mu_2 = 0.2$, Random is about 55% worse). Further, the greater the degree of heterogeneity, the more the displacement. This is quite easily understood by the following argument: Because class-2 nodes are slow, the job transfers that occur to these nodes by random assignment slow down the system by a large amount. For large degrees of heterogeneity, the curves corresponding to Random approach their respective probing curves after fairly high values for delays, as compared with **type-1** heterogeneous systems. For example, when $\mu_2 = 0.2$, there appears to be about $5 - 6\%$ benefit by probing, even with delays as high as $5S$. Thus, the value of remote state information is more pronounced and its effect is significant for higher delays in highly unbalanced **type-2** systems as compared with **type-1** systems.

The results in the case of Reverse probing (Figure 20) are very similar to that of the Forward probing example. In general, Random assignment performs quite poorly in comparison to probing, be it forward or reverse. Also, the greater the degree of heterogeneity, the more advantageous it is to probe. Here too, as in Forward probing, there is a more significant effect of probing at higher delays than for **type-1** systems.

5 Conclusions

In this paper, we have studied the problem of load sharing in the presence of delays when the underlying system is heterogeneous. We have studied two main types of heterogeneous systems, i.e., **type-1** systems where all the nodes have the same processing rates and capabilities but the arrival rate of local jobs at nodes may not be the same, and **type-2** systems, where different nodes may process jobs at different rates (however, the nodes are assumed to be functionally identical). The two algorithms which were studied in this context were called Forward and Reverse. The resulting Markov processes from these algorithms were solved using the Matrix Geometric solution technique. The analytical models were subjected to many interesting tests and the results of these tests and their implications were described in the previous section. We now summarize the most interesting results obtained.

- The solution technique of studying the nodes independently is an approximation of the Markov process for the system. We compared the analytical results with simulations of finite sized systems and determined that the results were very accurate, with the variation between the analytical and simulation results being about 2-4%.

- For **type-1** systems, the sensitivity to probe limit was observed to depend upon the degree of heterogeneity. In the case of Forward probing, the sensitivity was smaller with greater imbalance and the opposite was observed to be true for Reverse probing.

- In **type-1** systems, the effect of varying (T_1, T_2) showed that when the imbalance in systems was large, the performance was relatively insensitive to changes in T_2. However, choosing the appropriate value for T_1 was quite critical. The above observations were also hold for Forward as well as Reverse probing. Furthermore, as the imbalance between the classes decreased, the effect of changes in T_1 and T_2 became more alike.

- We determined the optimal performance of the two algorithms operating in **type-1** systems and compared these results against the optimal performance of the Random assignment, for a large range of delays. It was seen that when the systems were highly heterogenous, Random was much better than Reverse. However, Forward always performed better than Random, although the advantage of probing was more pronounced when the systems were more balanced. Further, the advantages of probing seemed to disappear when the delays were greater than three times the processing times.

- In the case of **type-2** systems it was seen that when T_2 is increased beyond a certain small number and the nodes are executing Forward probing, there is the potential for class-2 nodes to get saturated. The effect of varying T_1 is much less dramatic, however. The above effects are much more pronounced when class-2 nodes are much slower than class-1 nodes. Thus, selection of appropriate thresholds is much more critical for **type-2** heterogeneous systems than it is for homogeneous as well as **type-1** heterogeneous systems.

- In the case of **type-2** systems, we determined the optimal performance for the two algorithms over a large range of delays and compared these results against the Random assignment algorithm. It was seen from these tests that probing does significantly better than Random for **type-2** systems where very few jobs should be executed on class-2 nodes (depending upon how slow they are in comparison to class-1 nodes and some other factors). In **type-2** systems, the advantages of probing are seen to exist for greater values of delays ($\leq 4S - 5S$) than in **type-1** systems.

References

[1] R. Agrawal and A. Ezzat. Processor sharing in nest: a network of computer workstations. *Proc. 1st Int'l Conf. on Computer Workstations*, Nov. 1985.

[2] Y. C. Chow and W. H. Kohler. Models for dynamic load balancing in a heterogeneous multiple processor system. *IEEE Trans. Computers*, C-28:354–361, May 1979.

[3] D. L. Eager, E. D. Lazowska, and J. Zahorjan. Adaptive load sharing in homogeneous distributed systems. *IEEE Trans. Soft. Engg.*, SE-12(5):662–675, May 1986.

[4] M. Livny and M. Melman. Load balancing in homogeneous broadcast distributed systems. *Performance Evaluation Review*, 11(1):47–55, 1982.

[5] R. Mirchandaney. *Adaptive Load Sharing in the Presence of Delays.* PhD thesis, ECE Dept., University of Massachusetts, August 1987.

[6] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Analysis of the effects of delays on load sharing. *IEEE Trans. Computers*, 1987. To Appear.

[7] R. Mirchandaney, D. Towsley, and J. A. Stankovic. Effects of delays on receiver-initiated load sharing policies. *Journal of Parallel and Distributed Computing*, 1987. submitted for review.

[8] M. F. Neuts. *Matrix-Geometric solutions in Stochastic Models: An Algorithmic Approach. Mathematical Sciences*, Johns Hopkins University Press, 1981.

[9] J. A. Stankovic. Simulations of three adaptive decentralized controlled, job scheduling algorithms. *Computer Networks*, 8(3):199–217, June 1984.

[10] H. Stone. Critical load factors in two processor distributed systems. *IEEE Trans. Soft. Engg.*, SE-4(3), May 1978.

[11] A. Tantawi and D. Towsley. Optimal static load balancing in distributed computer systems. *J. ACM*, 32:445–465, Apr. 1985.

[12] M. Theimer, K. Lantz, and D. Cheriton. Preemptable remote execution facilities for the v-system. *Proceedings of the 10th Symposium on Operating System Principles*, December 1985.

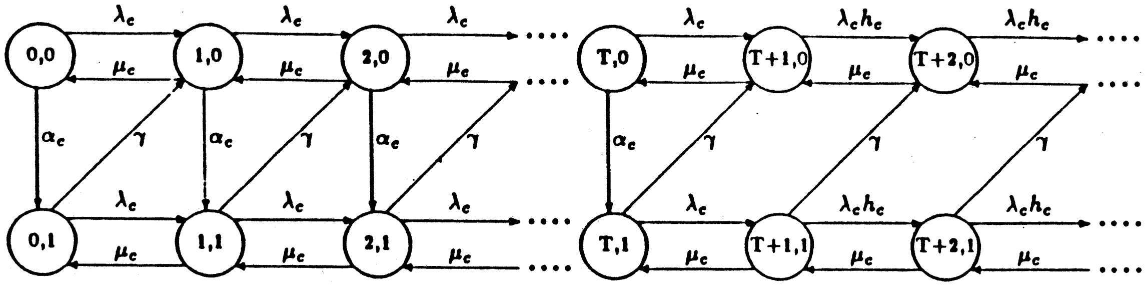


Figure 1: State Diagram of Forward Probing Algorithm

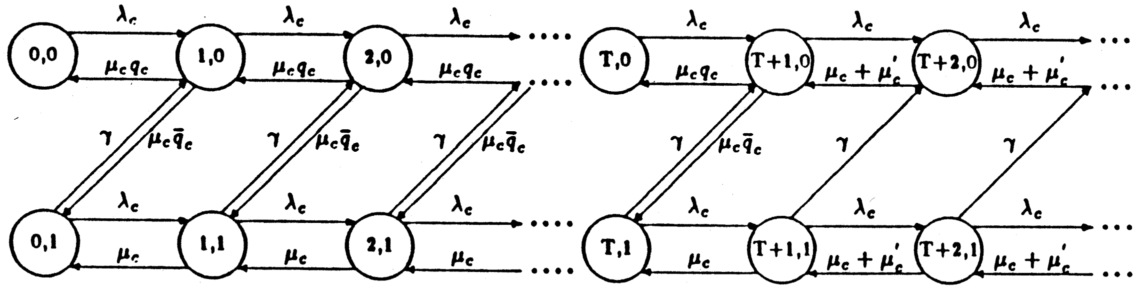


Figure 2: Staie Diagram of Reverse Probing Algorithm

Fig. 3. Probe Limit (FP, 0.1)

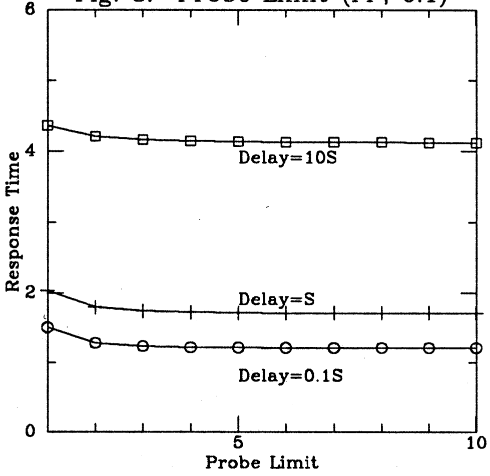


Fig. 4. Probe Limit (FP, 0.7)

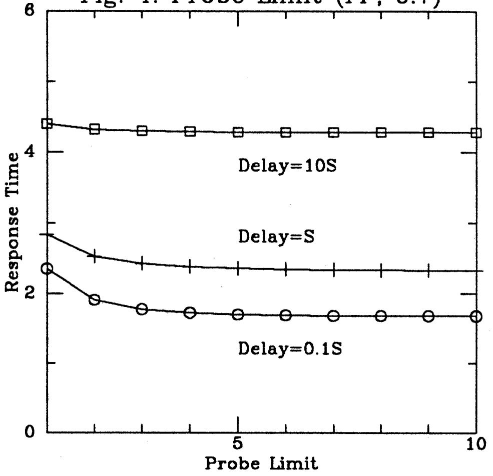


Fig. 5. Probe Limit (RP, 0.1)

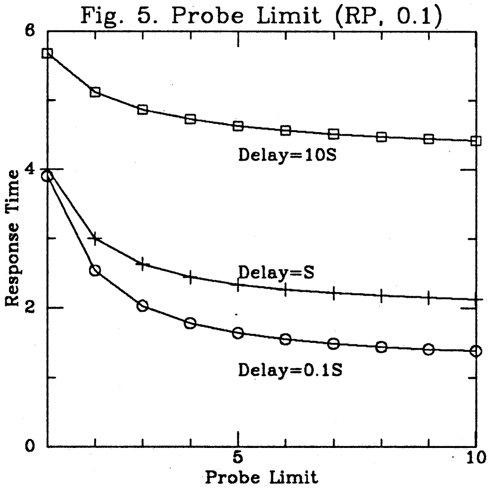


Fig. 6. Probe Limit (RP, 0.7)

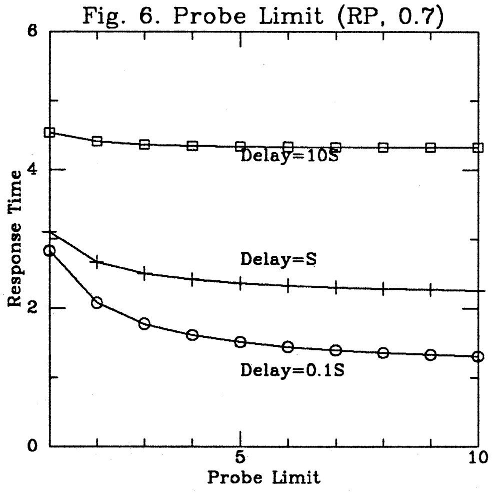


Fig. 7. T_1 (FP, 0.1)

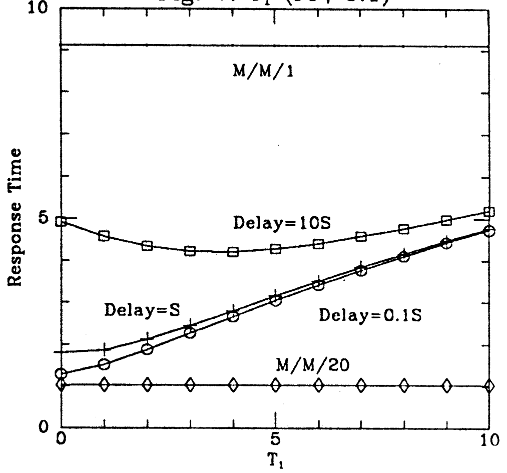


Fig. 8. T_2 (FP, 0.1)

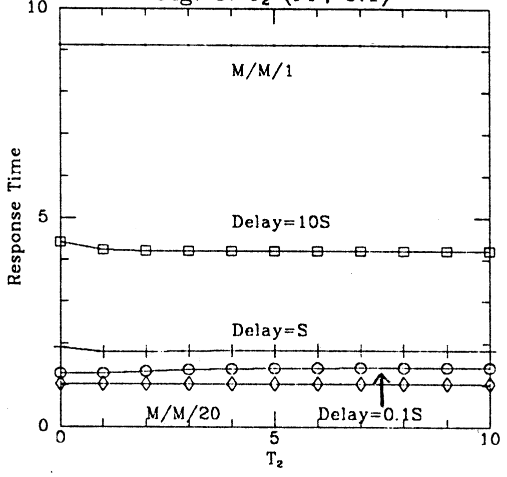


Fig. 9. Biased Probing (FP, 0.1)

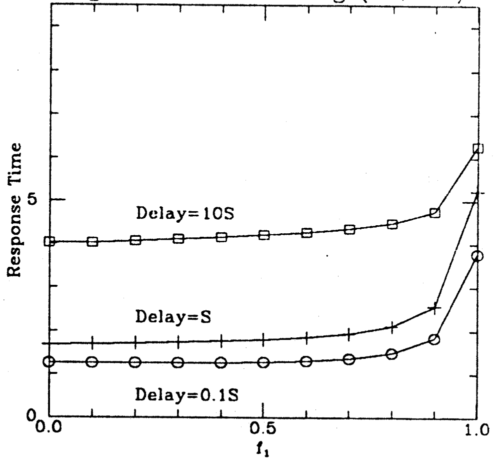


Fig. 10. Biased Probing (FP, 0.7)

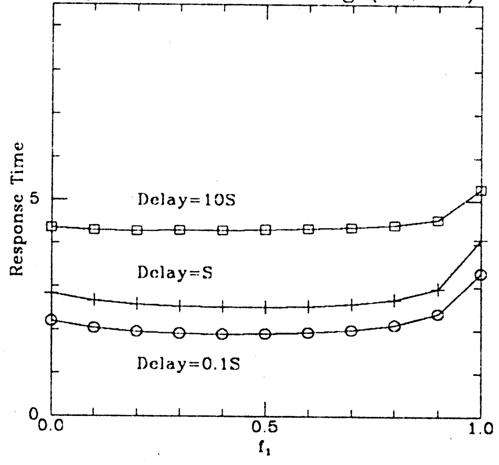


Fig. 11. Biased Probing (RP, 0.1)

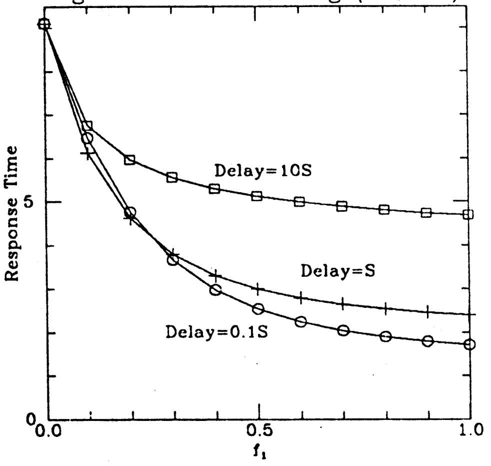


Fig. 12. Opt. R.T.(FP)

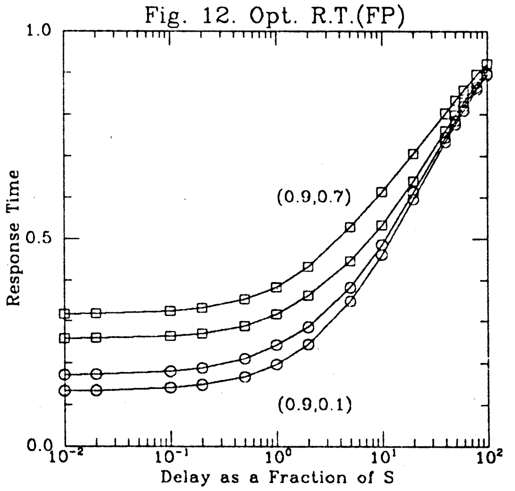


Fig. 13. Opt. R.T.(RP)

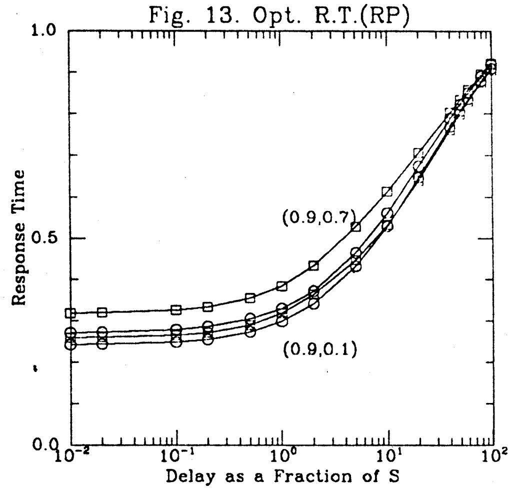


Fig. 14. Opt. R.T. Fixed Load

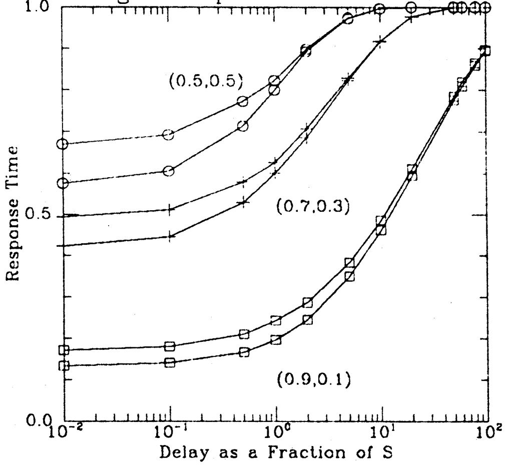


Fig. 15. T_1 (FP, 0.2)

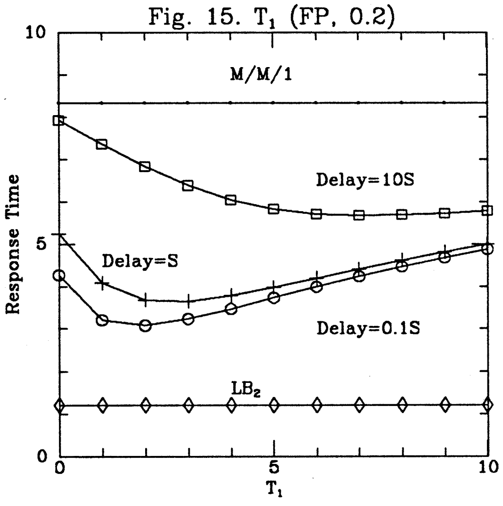


Fig. 16. T_2 (FP, 0.2)

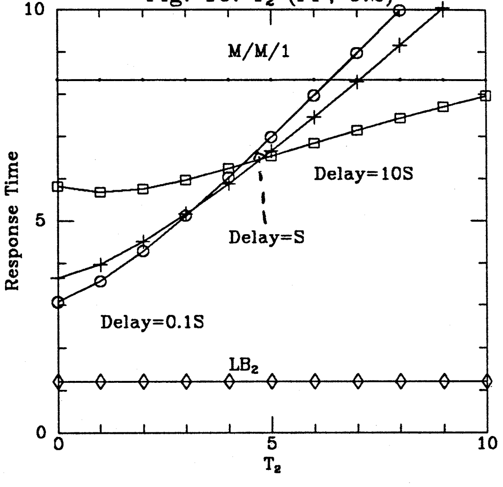


Fig. 17. T_1 (FP,0.8)

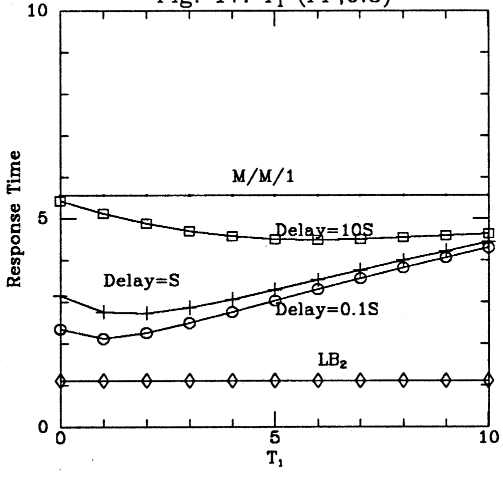
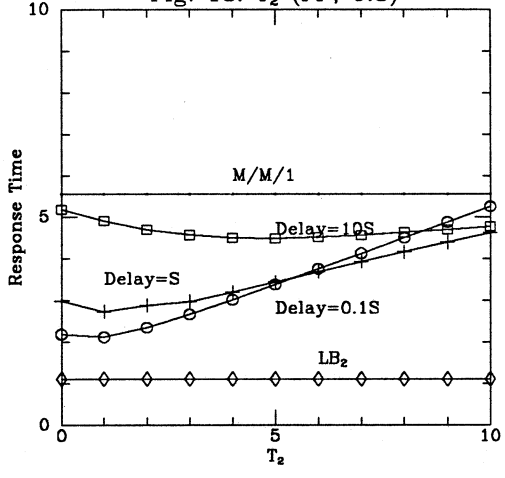


Fig. 18. T_2 (FP, 0.8)



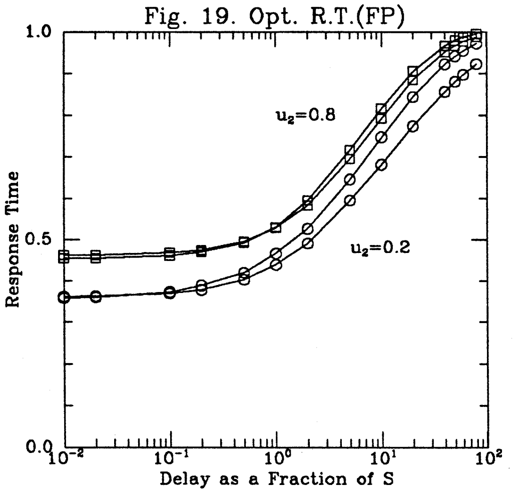


Fig. 19. Opt. R.T.(FP)

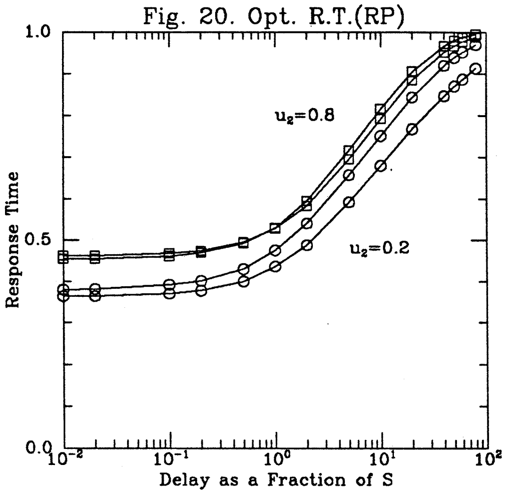


Fig. 20. Opt. R.T.(RP)