Adaptive Load Sharing in Homogeneous Distributed Systems

DEREK L. EAGER, EDWARD D. LAZOWSKA, AND JOHN ZAHORJAN

Abstract—**In most current locally distributed systems, the work generated at a node is processed there; little sharing of computational resources is provided. In such systems it is possible for some nodes to be heavily loaded while others are lightly loaded, resulting in poor overall system performance. The purpose of *load sharing* is to improve performance by redistributing the workload among the nodes.**

The load sharing policies with the greatest potential benefit are *adaptive* in the sense that they react to changes in the system state. Adaptive policies can range from simple to complex in their acquisition and use of system state information. The potential advantage of a complex policy is the possibility that such a scheme can take full advantage of the processing power of the system. The potential disadvantages are the overhead cost, and the possibility that a highly tuned policy will behave in an unpredictable manner in the face of the inaccurate information with which it inevitably will be confronted.

The goal of this paper is not to propose a specific load sharing policy for implementation, but rather to address the more fundamental question of the appropriate level of complexity for load sharing policies. We show that extremely simple adaptive load sharing policies, which collect very small amounts of system state information and which use this information in very simple ways, yield dramatic performance improvements. These policies in fact yield performance close to that expected from more complex policies whose viability is questionable. We conclude that simple policies offer the greatest promise in practice, because of their combination of nearly optimal performance and inherent stability.

Index Terms—**Design, load sharing, local area networks, performance, queueing models, threshold policies.**

I. Introduction

LOAD SHARING attempts to improve the performance of a distributed system by using the processing power of the entire system to "smooth out" periods of high congestion at individual nodes. This is done by transferring some of the workload of a congested node to other nodes for processing. The potential attractiveness of load sharing is enhanced by factors such as the increasing size of locally distributed systems, the use of shared file servers, the presence of pools of computation servers, and the development of streamlined communication protocols.

Manuscript received October 31, 1984; revised June 28, 1985. This work was supported by the National Science Foundation under Grants MCS-8302383 and DCR-8352098, and by the Natural Sciences and Engineering Research Council of Canada. Part of this work was conducted while E. D. Lazowska was on leave at Digital Equipment Corporation's Systems Research Center.

D. L. Eager is with the Department of Computational Science, University of Saskatchewan, Sask. S7N 0W0, Canada.

E. D. Lazowska and J. Zahorjan are with the Department of Computer Science, University of Washington, Seattle, WA 98195.

IEEE Log Number 8607941.

Two important components of a load sharing policy are the *transfer* policy, which determines whether to process a task locally or remotely, and the *location* policy, which determines to which node a task selected for transfer should be sent. Policies that use only information about the average behavior of the system, ignoring the current state, are termed *static* policies. Static policies may be either *deterministic* or *probabilistic*. Policies that react to the system state are termed *adaptive* policies.

Numerous static load sharing policies have been proposed. In the earliest formulations of the problem it was assumed that information about the average execution times and intercommunication requirements of all tasks were known. Typically the goal was to find a technique to deterministically allocate tasks to nodes so that the total time to process all tasks was minimized; for example [2], [13], [14]. More recently, Tantawi and Towsley [15] developed a technique to find the optimal probabilistic assignment.

Adaptive load sharing policies have received less attention. Livny and Melman [11] showed that in a network of autonomous nodes there is a large probability that at least one node is idle while tasks are queued at some other node, over a wide range of network sizes and average node utilizations. This is a key result because it clearly indicates the potential benefit of adaptive load sharing. Livny and Melman also developed a taxonomy of load sharing policies, and used simulation to evaluate a number of them. Bryant and Finkel [3] proposed a specific adaptive load sharing policy, and analyzed its performance using simulation. They also explored techniques for estimating the remaining service time of a task already being processed, a quantity of interest in deciding which task to transfer from a congested node. Krueger and Finkel [7] also used simulation to evaluate the performance of a specific policy. Barak and Shiloh [1] used limited experimentation with synthetic workloads to investigate a policy distinguished by the technique used to maintain system state information. They showed that if the workload remained constant, their policy converged to a load distribution that was near optimal.

Static load sharing policies are attractive because of their simplicity: "transfer all compilations originating at node X to computation server Y" or " $\cdots$ to computation servers Y and Z with probabilities 0.8 and 0.2, respectively." It is clear, though, that the potential of static policies is limited by the fact that they do not react to the

current system state: at the time a particular compilation originates at node X, computation server Y may be so heavily loaded that Z is a much superior choice, or both Y and Z may be so congested that processing the task locally is preferable, even considering the impact of this decision on other tasks originating at node X. The attraction of adaptive policies is that they do respond to system state, and so are better able to avoid those states with unnecessarily poor performance. However, since adaptive policies must collect and react to system state information, they are necessarily more complex than static policies. The adaptive policies that have been examined in the literature collect considerable state information and attempt to make the "best" choice possible based on that information. For example, the policy proposed by Krueger and Finkel [7] attempts to keep the queue length at each node near the system average queue length.[1]

From a practical point of view, such complexity raises a number of concerns. The first concern is the effect of overhead. The value of a policy depends critically on the overhead required to administer it, which may vary considerably depending on system characteristics. Excessive overhead may negate the benefits of an improved workload distribution.

The second concern is the effect of the occasional poor decisions that inevitably will be made. Complex policies rely on detailed information about the system state and the behavior of the workload. Not only is this information expensive to gather, but some quantities, such as the expected congestion at nodes in the near future or the amount of processing that a particular task requires to complete, cannot be known precisely regardless of the effort expended. Because of this, a decision that a complex load sharing policy expects to be near optimal may in fact be quite poor.

The final concern is the potential for instability. In attempting to fully exploit system processing power, a complex load sharing policy must make decisions based on subtle apparent misallocations of load. This requirement to react to small distinctions means that the inherent inaccuracy and rapidly changing nature of system state information may cause the policy to react in an unstable manner [6]. At the extreme, a form of *processor thrashing* can occur, in which all of the nodes are spending all of their time transferring tasks. Less complex policies, because they tend to react more slowly to changes in the system state, are inherently less susceptible to such instability.

Motivated by these concerns, in this paper we ask a fundamental question concerning adaptive load sharing policies in general: what is an appropriate level of complexity for such policies? We show that:

- Extremely simple adaptive load sharing policies—policies that collect a very small amount of state information and that use this information in very simple ways—yield dramatic performance improvements relative to the no load sharing case.
- These extremely simple policies in fact yield performance close to that which can be expected from complex policies that collect large amounts of information and that attempt to make the "best" choice given this information—policies whose viability is questionable.
- These results are valid over a wide range of system parameters.

We conclude that simple adaptive load sharing is of considerable practical value, and that there is no firm evidence that the potential costs of collecting and using extensive state information are justified by the potential benefits.

II. Policies and Models

In studying the appropriate level of complexity for adaptive load sharing policies, we consider a set of abstract policies that represent only the essential aspects of load sharing, and we investigate these policies using simple analytic models. Our objective is not to determine the absolute performance of particular load sharing policies, but rather to assess the relative advantages of varying degrees of sophistication. By representing only the essential aspects of load sharing and eliminating secondary details, we are better able to interpret the results of our comparative analysis and so build our intuition.

An obvious concern is that this approach may ignore "details" with significant practical implications—the issues noted in Section I, such as the actual cost of collecting and reacting to state information, the behavior of policies when this information is unavailable or out-of-date, etc. If the conclusion of our study were that increasing sophistication yielded substantial benefit, then these concerns would have to be addressed, because failure to properly account for these characteristics will tend to overstate the performance of complex policies relative to the performance of simple ones. However, the conclusion of our study is quite the opposite, despite giving the "benefit of the doubt" to complex policies.

A. System Model

We represent distributed systems as collections of identical nodes, each consisting of a single processor. The nodes are connected by a local area broadcast channel (e.g., an Ethernet). All nodes are subjected to the same average arrival rate of tasks, which are of a single type.

In contrast to previous papers on load sharing, we represent the cost of task transfer as a processor cost rather than as a communication network cost. It is clear from measurement and analysis [9] that the processor costs of packaging data for transmission and unpackaging it upon reception far outweigh the communication network costs of transmitting the data. Further, network delays are small, and are almost entirely overlapped with processing

[1] An implicit assumption of most proposed schemes is that it is desirable to attempt to balance the queue lengths at the processors. In fact, such balancing is not required. All that is necessary for optimal performance (in the standard homogeneous model) is that all processors be busy if any task is waiting. Thus in this paper we purposefully adopt the terminology "load *sharing*" rather than "load *balancing*."

related to use of the network. Representing the network cost in addition to the processor cost would not affect the tractability of our models, but nor would it affect our results. Thus, for simplicity, it is omitted. (In Section III-E we will show that, under reasonable assumptions, the total communication network load imposed by adaptive load sharing is negligible.)

Our homogeneity assumptions—that nodes are identical and are subjected to the same average arrival rate of tasks—also are made principally to simplify the presentation, and do not undermine the applicability of the results. Node homogeneity is a reasonable assumption when considering load sharing among clusters of workstations or clusters of computation servers. Arrival homogeneity merely implies that *over the long term* the external load imposed on each node is the same. Over the short term, these loads may vary considerably. The entire objective of adaptive load sharing is to respond to such variations. Even if homogeneity does not hold (the system consists of a mix of nodes of different types, or there are differences in external loads), models that consider this case (but that are not considered here) indicate the suitability of simple policies. These simple policies are similar to those for homogeneous systems, but they additionally utilize the relatively static information specifying the system inhomogeneities.

B. Load Sharing Policies

We will study three abstract load sharing policies, comparing their performance to each other and to two "bounding" cases: no load sharing, and perfect load sharing at zero cost. As noted in Section I, a load sharing policy has two components: a *transfer* policy that determines whether to process a task locally or remotely, and a *location* policy that determines to which node a task selected for transfer should be sent. Each of these subpolicies might be expected to employ system state information. The three load sharing policies that we consider have identical transfer policies, but differ in their location policies.

The transfer policy that we have selected is a *threshold* policy: a distributed, adaptive policy in which each node uses only local state information. *No exchange of state information among the nodes is required in deciding whether to transfer a task.* A task originating at a node is accepted for processing there if and only if the number of tasks already in service or waiting for service (the node *queue length*) is less than some threshold T. Otherwise, an attempt is made to transfer that task to another node. Note that only newly received tasks are eligible for transfer. Transferring an executing task poses considerable difficulties in most systems [12].

The three location policies that we examine for use in conjunction with this extremely simple transfer policy are referred to as *Random*, *Threshold*, and *Shortest*. They are discussed in the subsections that follow.

1) Random: The simplest location policy is one that uses no information at all. With the Random policy a destination node is selected at random and the task is transferred to that node. *No exchange of state information among the nodes is required in deciding where to transfer a task.*

A question that arises in considering the behavior of the random policy is how the destination node should treat an arriving transferred task. The obvious answer is that it should treat it just as a task originating at the node: if the local queue length is below threshold the task is accepted for processing; otherwise it is transferred to some other node selected at random. As shown in Appendix A, this choice has the unfortunate property of causing instability: no matter what the average load, it is guaranteed that eventually the system will enter a state in which the nodes are devoting all of their time to transferring tasks and none of their time to processing them. This instability is analogous to that arising in the infinite population ALOHA system [5]; repeated task transfers in load sharing systems play a similar role with respect to stability as do message collisions in ALOHA.

Instability can be overcome by the use of an appropriate control policy. Such control policies have been developed for a number of multiple access systems [8], [16]. The simple control policy that we adopt here is to restrict the number of times that a task can be transferred using a static *transfer limit*, L_t. The destination node of the L_tth transfer of a task must process that task regardless of its state.

A key result of this paper is that, in many situations, this extremely simple combination of a threshold transfer policy and a random location policy with a static transfer limit dramatically improves system response time relative to no load sharing. Since this policy uses no system state information at all, this is an indication that very simple schemes can yield significant benefits.

2) Threshold: Threshold is a location policy that acquires and uses a small amount of information about potential destination nodes. Under this policy a node is selected at random and *probed* to determine whether the transfer of a task to that node would place it above threshold. If not, then the task is transferred; the destination node must process the task regardless of its state when the task actually arrives. If so, then another node is selected at random and probed in the same manner. This continues until either a suitable destination node is found, or the number of probes exceeds a static *probe limit*, L_p. In the latter case, the originating node must process the task.

The objective of the Threshold policy is to avoid "useless" task transfers (those to nodes already at or above their threshold), although, like Random, it makes no attempt to choose the "best" destination node for a task. The use of probing with a fixed limit, rather than broadcast, ensures that the cost of executing the load sharing policy will not be prohibitive even in large networks. As will be discussed in Section III-D, the performance of this policy is surprisingly insensitive to the choice of probe limit. In other words, the performance with a small (and

economical) probe limit, e.g., 3 or 5, is almost as good as the performance with a large probe limit, e.g., 20.

A key result of this paper is that the Threshold policy provides substantial performance improvement relative to the Random policy for a wide range of system parameters. This indicates that the use of a small amount of state information in a simple (and computationally inexpensive) way is likely to more than compensate for the additional cost.

3) Shortest: This location policy acquires additional system state information and attempts to make the "best" choice given this information. L_p distinct nodes are chosen at random, and each is polled in turn to determine its queue length. The task is transferred to a node with the shortest queue length, unless that queue length is greater than or equal to the threshold, in which case the originating node must process the task. The destination node must process the task regardless of its state at the time the task actually arrives. (A simple improvement to Shortest is to discontinue probing whenever a node with queue length of zero is encountered, since that node is guaranteed to be an acceptable destination.)

The Shortest policy uses more state information, in a more complex manner, than does the Threshold policy. A key result of this paper is that the performance of Shortest is not significantly better than that of the simpler Threshold policy. This suggests that state information beyond that used by Threshold, or a more complex usage of state information, is of little benefit.

C. Analytic Model Structure and Solution

The three policies introduced in the previous section have similar analytic models.

Each node is modeled as a queueing center [4]. New tasks arrive at each node at average rate λ. The average task service time (processing cost) is S. We define the *load factor* ρ of each node to be the ratio of offered load to service capacity (i.e., $\rho = \lambda S$). Because of the cost of task transfer, the average utilization of the nodes may be significantly greater than ρ.

The cost of transferring a task from one node to another is represented by a processing cost at the sending node whose average value is denoted by C. This cost is a key parameter. (The processing cost of receiving a task is included in the service time of the task, S.) As discussed earlier, communication network costs are assumed to be negligible (relative to other costs). In addition, the cost of probing a node is assumed to be negligible. These assumptions are examined in Section III-E.

At each node, the transferring of tasks is given preemptive priority over the processing of tasks. In the processing of tasks, any service discipline that selects tasks in a way that is independent of their actual service time (e.g., First-Come-First-Served, Processor Sharing) is allowed. All of the performance measures that will be considered here are independent of the actual discipline used.

Under the assumptions stated above, a Markov model of a distributed system under each of the load sharing policies can be constructed. The model has a very large state space, with complex structure. To simplify the analysis we decompose the model, by assuming that the state of each node is stochastically independent of the state of any other node. Each node can then be analyzed in isolation. The effect of the remainder of the system on an individual node is represented by an arrival process of transferred tasks. Because the network is homogeneous, system performance measures can be obtained by analyzing a model of any individual node.

This decomposition approach is asymptotically exact as the number of nodes in the system increases, since the queue lengths of the nodes are asymptotically independent. For systems of finite size the analysis is an approximation. Results obtained from simulation indicate that this approximation, which also has been used in modeling multiple access protocols such as ALOHA, introduces negligible errors even for relatively small numbers of nodes. In particular, the major numerical results used in our study have been validated through simulation for networks of 20 nodes (and thus certainly for greater numbers of nodes, although not necessarily for smaller numbers). A sample of our simulation results is contained in Appendix C.

All of the quantities needed to determine the state transition rates of the model of an individual node are input parameters, with the exception of a description of the arrival process of transferred tasks. The nature of this arrival process depends on the load sharing policy. For the Random policy, the arrival rate of transferred tasks is independent of the current queue length (i.e., state) of the node, since Random utilizes no information about the state of potential destination nodes. For the Threshold policy, arrivals of transferred tasks are constrained to those states in which the node is below its threshold. For the Shortest policy, the arrival rate of transferred tasks decreases as the queue length at the node increases.

The assumption of homogeneous nodes makes it possible to determine the arrival rate of transferred tasks: the overall arrival rate must equal the overall rate at which the node transfers tasks to other nodes, and the equilibrium state probabilities of potential destination nodes, as "observed" when probing, for example, are identical to those of the node itself. These quantities are model outputs. For the Random and Threshold policies this dependence of model inputs on outputs yields a single equation in a single unknown, which is solved numerically. For the Shortest policy, the dependence is sufficiently complex that an iterative numerical technique is required.

Equations relating the variables of the model are developed by considering the node to be in one of two phases: "processing" (when the node queue length is less than or equal to the threshold value), or "transferring" (when the node queue length is greater than the threshold value). During a processing phase the node is either idle or is processing tasks. During a transferring phase the node is busy, either transferring tasks or processing tasks

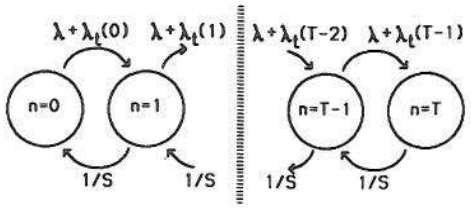


Fig. 1. Processing phase birth-death model.

that could not be transferred because of a restriction imposed by the location policy.

Fig. 1 shows the birth-death model corresponding to the processing phase. In each state the arrival rate of tasks is the sum of the rate of arrival of new tasks (λ) and the rate of arrival of tasks transferred to this node by the remainder of the system ($\lambda_t(n)$). This latter term is in general dependent on the queue length n at the node and the load sharing policy being modelled. This submodel can be analyzed using standard methods.

A transferring phase is identical in behavior to a busy period of a two class, preemptive priority HOL M/M/1 queue [4], where the classes are tasks that are processed and tasks that are transferred. The total arrival rate at the node is $(\lambda + \lambda_t(T^+))$, where $\lambda_t(T^+)$ denotes the arrival rate of tasks transferred to the node conditioned on the node being in a transferring phase. The proportion of this total arrival rate consisting of tasks that will be processed and the proportion consisting of tasks that will be transferred depends on the probability of a task not being transferred because of a location policy restriction.

The analyses of the birth-death model corresponding to the processing phase and of the HOL priority model corresponding to the transferring phase yield conditional state probabilities and performance measures. These are combined using weights representing the proportion of time the system spends in each phase to determine overall performance. The performance measures that can be obtained include average response times, utilizations, queue lengths, transfer rates, and probe rates. Details on the analyses of the two phases and the calculation of performance measures are given in Appendix B.

III. Performance Comparisons

Our objective is to compare the performance of three abstract load sharing policies—Random, Threshold, and Shortest—to each other and to two "bounding" cases: no load sharing (represented by K independent M/M/1 queues, where K is the number of nodes), and perfect load sharing at zero cost (represented by an M/M/K queue). Our measure of performance is mean response time as a function of system load.

This comparison is potentially difficult because of the large number of parameters involved: the average task service time S, the average cost of task transfer C, the threshold T, the probe limit for the Threshold and Shortest policies L_p, the transfer limit for the Random policy L_t, and the number of nodes K. Fortunately, the results are robust in the sense that the intuition gained from studying

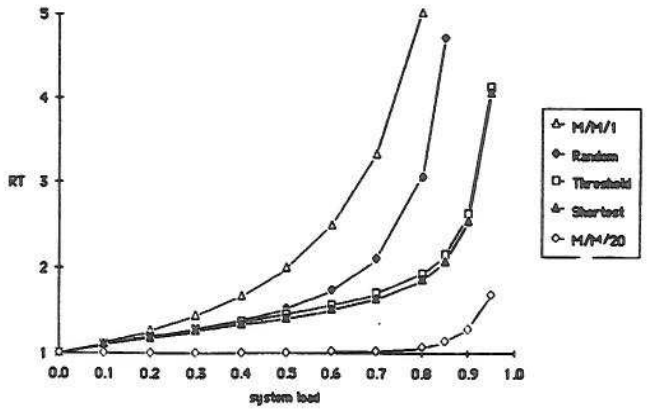


Fig. 2. Principal performance comparison: response time versus load ρ. S(task service time) = 1. C(cost of task transfer) = 0.1. T(threshold) = 2. L_p(probe limit for Threshold and Shortest) = 3. L_t(transfer limit for Random) = 1.

performance for a "representative" set of parameter values is valid over a wide range of parameter values. The structure of our presentation exploits this fact: Section III-A contains a thorough discussion of response time versus system load for a particular choice of parameter values, while Sections III-B–III-F explore the sensitivity of these results to the various parameters.

A. Principal Performance Comparison

Fig. 2 is a graph of average response time versus load for each of the five policies under consideration. For convenience, S is fixed at 1 throughout our analysis so that response times may be considered to be reported in units of the task service time.

We will first discuss the figure, and then the choice of parameter values indicated in the text accompanying the figure. The key observations concerning the figure are as follows.

- The Random policy yields substantial performance improvement over no load sharing. The degree of the improvement is surprising since the Random policy is so simple.
- The Threshold policy yields substantial further performance improvement for system loads greater than 0.5. This shows the value of the small amount of additional information utilized by Threshold.
- The Shortest policy yields negligible further performance improvement over the Threshold policy. Again this is somewhat surprising, since Shortest acquires considerably more information than Threshold, and attempts to make the "best" decision based on that information.

If factors such as the actual cost of collecting and reacting to state information, the behavior of policies when this information is unavailable or out-of-date, etc., are ignored, then intuitively Shortest should have the best performance among all load sharing policies that employ threshold transfer policies. Thus, based on the comparison of Threshold and Shortest, we can conclude (subject to verification that our results are robust with respect to the choice of parameter values) that relatively simple information concerning potential destination nodes is suffi-

cient to obtain essentially all of the benefit available through this class of policies. This conclusion is reinforced by the fact that our analysis indeed gives the "benefit of the doubt" to complex policies by ignoring the issues just noted, which clearly are more significant for complex policies such as Shortest than for simpler policies such as Threshold.

In Fig. 2 there is significant room for improvement between the performance of the Shortest policy and the bound established by the M/M/K analysis. This might suggest that our conclusion with respect to the information required by location policies does not hold for transfer policies: perhaps significantly improved performance can be obtained by using a transfer policy that employs more than local threshold information. However, there are reasons (in addition to the obvious pragmatic ones) to believe that simple transfer policies are as relatively advantageous as simple location policies. The M/M/K analysis does not provide a tight bound: it assumes perfect load sharing at zero cost, when in fact an "optimal" policy would require a significant rate of task transfers, each of which has a nonnegligible cost. Further, the parameter values used in Fig. 2 are conservative, rather than being advantageous to the policies under consideration. We will discuss these parameter values now, and return to the question of an appropriate optimistic bound on achievable performance in Section III-F.

In Fig. 2, the average cost of task transfer C was 0.1, that is, 10 percent of the average task service time. We believe this to be a conservative (overly high) choice; our reasoning, as well as the sensitivity of the results to the cost of task transfer, is explored in Section III-B.

The threshold T was 2. That is, a node would attempt to transfer a task that arrived when two (or more) tasks already were present. The sensitivity of the results to the choice of threshold is explored in Section III-C.

The probe limit for the Threshold and Shortest policies L_p was 3. The sensitivity of the results to the choice of probe limit is explored in Section III-D. The rates of probing in the Threshold and Shortest policies are compared in Section III-E.

The transfer limit for the Random policy L_t is set to 1. The implications of this will be discussed in Section III-B. The rate of task transfers for all policies, and its impact on network congestion, is discussed in Section III-E.

As noted in Section II, the number of nodes K is not a parameter of our analysis of Random, Threshold, and Shortest. The analysis is asymptotically exact as the number of nodes increases. Our major results have been validated through simulation for networks of 20 nodes, implying that the performance of the policies quickly becomes insensitive to the number of nodes as the number of nodes increases.

B. Sensitivity to Transfer Cost

We believe that the average cost of task transfer C, although nonnegligible, can be expected to be quite low relative to the average cost of task processing S; the range

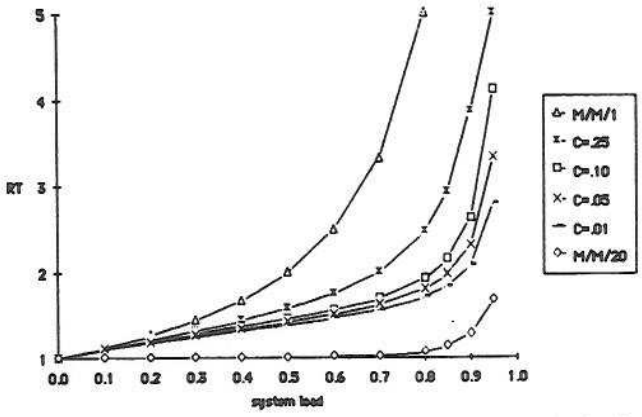


Fig. 3. Response time versus load ρ for various transfer costs C(Threshold policy). S(task service time) = 1. T(threshold) = 2. L_p(probe limit) = 3.

1–10 percent seems to include the cases of greatest interest. (We mean this to be interpreted as an average across many tasks; we are not asserting a relationship between processing cost and transfer cost.)

Transfer costs higher than 10 percent, although certainly possible, would likely be infrequent. On current systems not designed to facilitate load sharing (e.g., 4.2 BSD Unix running on Vaxes connected by Ethernet, using FTP on top of TCP/IP for task transfer), the transfer costs for relatively small compilations and formatting runs are a few percent of the processing costs. We would expect that any practical implementation of load sharing would attempt to select tasks such as these for migration—tasks with a relatively high ratio of processing cost to transfer cost. One also can easily imagine more efficient protocols. The advent of systems based on file servers and database servers will further decrease the cost of task transfer: only a descriptor will be shipped.

At the other end of the spectrum, performance is insensitive to transfer cost for costs of 1 percent or less.

Fig. 3 shows average response time versus system load for the Threshold policy for four different average transfer costs C: 0.01 (1 percent of the processing cost), 0.05 (5 percent), 0.10 (10 percent as shown in Fig. 2) and 0.25 (25 percent). The other parameters (e.g., threshold, probe limit) are fixed as in Fig. 2. Note that in practice the average transfer cost would be a factor considered in selecting the value of the threshold, whereas a fixed threshold of 2 was used for each transfer cost in Fig. 3.

Fig. 4 shows average response time versus average transfer cost C for all policies, for a fixed system load of 0.7. (Note that a log scale is used for the transfer cost axis.) Again, the other parameters (e.g., threshold, probe limit) are fixed as in Fig. 2. The performance of Threshold and Shortest relative to one another is insensitive to transfer cost. Their performance relative to the extremes of the M/M/1 and M/M/K analyses is insensitive to transfer cost for values below 0.05 (5 percent of processing cost), but degrades rapidly as transfer costs exceed 0.25 (25 percent). The Random policy performs relatively better at low transfer costs than at high ones. In fact, our

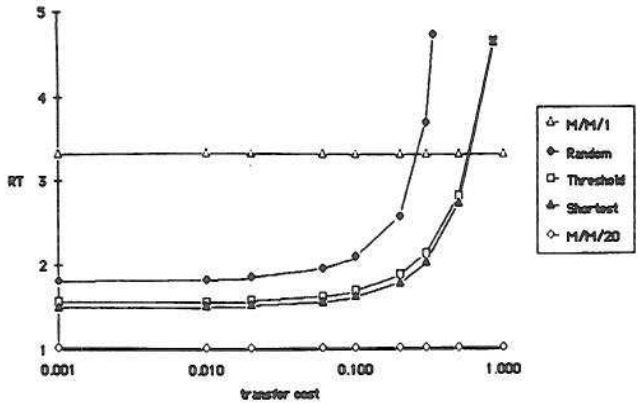


Fig. 4. Response time versus transfer cost C at fixed load ρ. S(task service time) = 1. ρ(system load, λS) = 0.7. T(threshold) = 2. L_p(probe limit for Threshold and Shortest) = 3. L_t(transfer limit for Random) = 1.

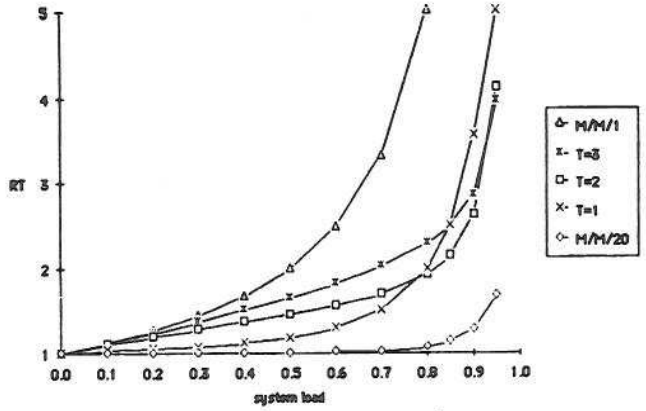


Fig. 5. Response time versus load ρ for three thresholds T (Threshold policy). S(task service time) = 1. C(cost of task transfer) = 0.1. L_p(probe limit) = 3.

analysis does not do justice to Random at low transfer costs. Reasonable performance at relatively high transfer costs requires a transmission limit L_t of 1, the value used throughout our analysis. However, at relatively low transfer costs a higher transmission limit yields substantially better performance, since tasks can be transferred multiple times (at low cost) in search of a suitable node. This yields behavior similar to that of the Threshold policy, except that the task itself is sent, rather than a probe.

C. Choice of Threshold

The threshold T is a fundamental parameter: for each of the three load sharing policies it determines when a task transfer will be attempted (through the transfer policy); for the Threshold and Shortest policies it determines whether the transfer will be allowed (through the location policy).

Clearly the "best" threshold depends on the system load and the transfer cost. At low loads a low threshold is appropriate because many nodes are idle, whereas at high loads a high threshold is appropriate because most nodes have significant queue lengths. Low thresholds are appropriate for low transfer costs, since smaller differences in node queue lengths can be exploited; high costs demand higher thresholds.

One might imagine that a complex adaptive threshold selection strategy would be required to obtain reasonable performance. Figs. 2–4, which used a fixed threshold of 2, indicate that this is not the case. To explore this point further, Fig. 5 shows average response time versus system load for the Threshold policy for three thresholds: 1, 2 (as shown in Fig. 2) and 3. (The corresponding graph for Shortest is essentially indistinguishable.) The other parameters are fixed as in Fig. 2. We see that 1 is the optimal threshold for system loads below 0.8, 2 is the optimal threshold for loads between 0.8 and 0.9, and thresholds greater than 2 are advantageous at (unreasonably high) system loads above 0.95. (The Random policy exhibits greater sensitivity to choice of threshold, but the optimal threshold still is 1 over a wide range of system load.)

These results suggest that the optimal threshold is not very sensitive to system load. Thus, a simple adaptive policy that selects among two or three threshold values, perhaps based on information acquired while probing, offers potential benefit at low cost and risk. Such policies are an area of current research.

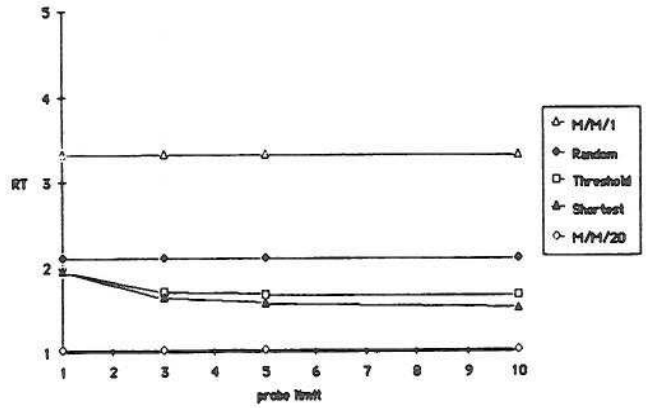


Fig. 6. Response time versus probe limit L_p at fixed load ρ. S(task service time) = 1. C(cost of task transfer) = 0.1. ρ(system load, λS) = 0.7. T(threshold) = 2. L_t(transfer limit for Random) = 1.

D. Choice of Probe Limit

Fig. 6 shows average response time versus probe limit for all policies, for a fixed system load of 0.7. (Random has no probe limit; it is included along with M/M/1 and M/M/K for comparison purposes.)

In the case of Threshold, the rapid decrease in the marginal benefit of increasing the probe limit is easy to explain. The purpose of probing in this policy is to locate a node that is below threshold. If p is the probability that a particular node is below threshold, then (because the nodes are assumed to be independent) the probability that a node below threshold is first encountered on the ith probe is $p(1-p)^{i-1}$. For large p, this quantity decreases rapidly: the probability of succeeding on the first few probes is high. For small p, the quantity decreases more slowly. However, since most nodes are busy, the improvement in system-wide response time that will result from locating a node below threshold is small, so abandoning the search after the first few probes does not carry a substantial penalty. It is clear that small probe limits are appropriate.

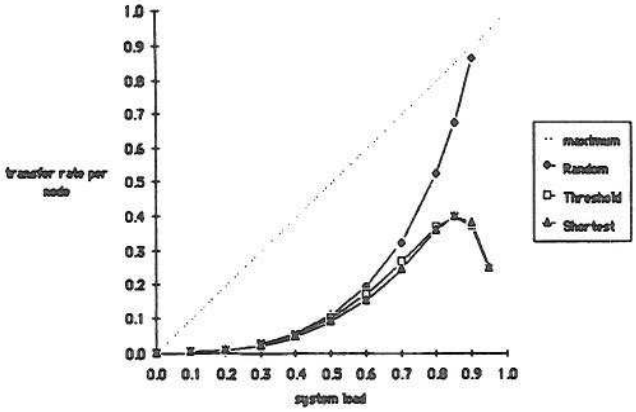


Fig. 7. Task transfer rate per node versus load ρ. S(task service time) = 1. C(cost of task transfer) = 0.1. T(threshold) = 2. L_p(probe limit for Threshold and Shortest) = 3. L_t(transfer limit for Random) = 1.

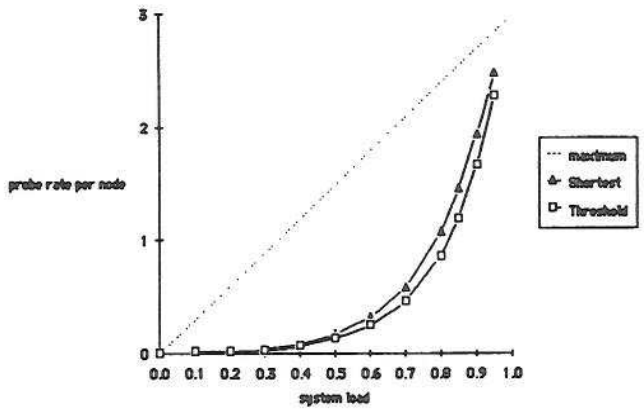


Fig. 8. Rate of probes per node versus load ρ. S(task service time) = 1. C(cost of task transfer) = 0.1. T(threshold) = 2. L_p(probe limit) = 3.

In the case of Shortest, the situation is somewhat more complex. There may be some marginal benefit even to very large probe limits. Fig. 6 shows this, but understates the effect for two reasons. First, this benefit is greatest at high system loads; the load of 0.7, selected for consistency with other figures and a "reasonable" high load for illustrative purposes, is not high enough to fully display the effect. Second, at a threshold value of 2, Shortest cannot find a node with a queue length more than one task shorter than that of the first acceptable destination found, since the maximum acceptable destination queue length is 1. If the threshold were higher (because of a higher system load, for example), there would be more room for improvement. However, it still is the case that relatively small probe limits are appropriate for Shortest. The marginal benefit of increasing the probe limit does decrease (although this decrease is not as rapid as for Threshold), and, as the probe limit increases, the rate and hence the cost of probing increases (this increase is actually greater for Shortest than for Threshold). (The latter effect is not shown in the figures, since the cost of probing is omitted from our analysis.)

E. Transfer and Probing Traffic

Here we consider the network traffic due to task transfers (for all three policies) and probes (for Threshold and Shortest).

Threshold and Shortest each will transfer an individual task at most one time. This also is true of Random with the transfer limit of 1 that we have been using in our examples. This implies that the transfer rate *per node* can be no greater than λ, and that the task transfer rate over the entire system can be no greater than $K\lambda$.

As illustrated in Fig. 7, the actual task transfer rates for Threshold and Shortest are extremely similar and are considerably less than this maximum value, while the rate for Random approaches this maximum only for relatively high system loads. (The unit of time in the figure is the task processing time S, which is equal to 1.) It is impossible to translate these results into network utilization without making rather arbitrary assumptions, but for the sake of illustration, suppose that the processing cost of tasks is related to their size in the ratio of 1 second per 1 kbytes (e.g., a 100K task would process for 100 seconds), and that we are considering a 10 Mbit network (i.e., each 1K transferred requires 0.0008 seconds of network time). Then, an upper bound network utilization due to task transfers under the Threshold or Shortest policies would be $0.4 \times 0.0008 \times K$ (since these policies never exceed a transfer rate of 0.4), yielding a network utilization of 3 percent in a system of 100 nodes. If we envision a system based on file servers, then the network cost of load sharing over and above the inherent cost of remote file access is insignificant regardless of the particular assumptions that are made.

Note, incidentally, that the decrease in transfer rate at high loads for Threshold and Shortest is exactly what one should expect: in this situation many nodes are over threshold, so there is an increasing probability that a transfer attempt will fail, i.e., that no suitable destination node will be found during the probe phase.

Fig. 8 shows the rate of probes per node for Threshold and Shortest. Because the probe limit is 3, the maximum probe rate per node can be no greater than 3λ. The figure shows that the two policies behave similarly, with Threshold requiring marginally fewer probes than Shortest. The difference is maximized at approximately the point corresponding to the maximum transfer rate. (The difference between the two policies can be much larger for larger probe limits and/or thresholds.) As the system load increases beyond 0.7, the probe rate for each policy begins to increase substantially. Still, given the maximum rate per node of $L_p\lambda$, the network load and processor load due to probing will be negligible. (Note that at most L_p probes can be performed per processed task.) Probing could be implemented, for example, using a single remote procedure cell with a return value that is binary (in the case of Threshold) or integer (in the case of Shortest).

F. An Optimistic Evaluation

We noted in Section III-A that our evaluation was conservative in two respects: the M/M/K analysis does not provide a tight bound, and the choice of parameter values is not advantageous to the policies under consideration.

In this section we briefly consider the Threshold scheme from a more optimistic point of view:

- The threshold T, rather than being fixed, is set to either 1 or 2 depending on the system load. Fig. 5 suggests the improvement that this offers.
- The probe limit is increased from 3 to 5. Fig. 6 suggests the improvement that this offers.
- We consider an average transfer cost of 0.01, in addition to 0.10. (The average task processing time remains fixed at 1.) Fig. 3 suggests the improvement that this offers.
- We compare performance to a plausible lower bound that includes the average transfer cost, obtained as follows:

An M/M/K queueing system can be viewed as a model of perfect load sharing among K nodes: no node will ever be idle when more than a single task is present at some other node. The results of an M/M/K analysis are "too optimistic," though, because the cost of the task transfers required to achieve this perfect load sharing is ignored.

Livny and Melman [11] calculate a lower bound on the number of task transfers required to ensure that no node will ever be idle when another node has a queue length greater than 1. They note that a task must be transferred when one of the following events occurs: an arrival occurs at a busy node when there are less than K tasks in the system, or a completion occurs at a node with only one task present when there are more than K tasks in the system. Thus, the minimum rate of task transfers $\tilde{\lambda}_T$ can be expressed as

$$\tilde{\lambda}_T = \sum_{i=1}^{K-1} \left\{ \lambda i P[i] + \frac{1}{S}(K - i)\, P[K + i] \right\}$$

where $P[j]$ is the probability that there are j tasks in an M/M/K queueing system with arrival rate $K\lambda$ and service rate per server $1/S$.

This expression can be used to increase the average task service time S by the transfer cost C multiplied by the probability that a task requires a transfer, $\tilde{\lambda}_T/\lambda$. Since the use of these increased service times in the M/M/K analysis results in a new set of state probabilities (implying a new rate of task transfers), an iteration is used.

Fig. 9 shows a comparison of M/M/1, Threshold with transfer costs of 0.1 and 0.01, the modified M/M/K analysis just described (labeled "Mod. M/M/20") with a transfer cost of 0.1, and the traditional M/M/20 analysis. The important observations are:

- With a variable threshold and a probe limit of 5, the performance of Threshold with a transfer cost of 0.1 is noticeably improved over that shown in Fig. 2, i.e., noticeably further from the performance of the M/M/1 system, and noticeably closer to the performance of the M/M/20 system.
- Viewing the modified M/M/K analysis as a plausible lower bound, there is little room for improvement beyond the performance of Threshold.

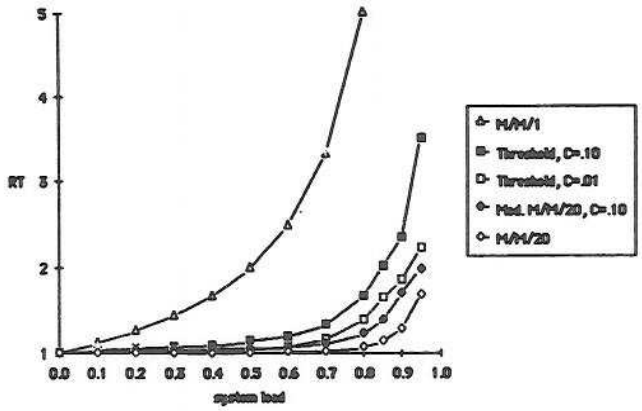


Fig. 9. Optimistic evaluation of the Threshold policy. S(task service time) = 1. T(threshold) = *variable*. L_p(probe limit) = 5.

- When the transfer cost drops to 0.01, the performance of Threshold is such that there is very little room for improvement relative to the absolute bound established by the M/M/20 analysis. (The performance of the modified M/M/20 system with a transfer cost of 0.01 is indistinguishable from the performance of the traditional M/M/20 system.)

IV. Summary

We have explored the use of system state information in adaptive load sharing policies for locally distributed systems, with the goal of determining an appropriate level of policy complexity. Our investigations have been based on the use of simple analytic models of load sharing policies. Simulation results have indicated the validity of these models.

Our results suggest that extremely simple load sharing policies using small amounts of information perform quite well—dramatically better than when no load sharing is performed, and nearly as well as more complex policies that utilize more information. This provides convincing evidence that the potential benefits of adaptive load sharing can in fact be realized in practice.

Our original intent in considering the class of threshold policies in general, and the Threshold policy in particular, was to establish a plausible bound on the performance of realistic load sharing schemes by considering a policy so simple that one expects to be able to do better in practice. However, the results of our analysis indicate that fairly direct derivatives of Threshold are plausible candidates for implementation. In particular, our results have shown the benefit of "threshold-type" information, as opposed to no information at one extreme or to "complete" information at the other.

Appendix A
Instability of Random with No Transmission Limit

This Appendix considers a variation of the Random policy in which there is no transfer limit ($L_t = \infty$): transferred tasks are treated exactly as new tasks when applying the transfer policy. We refer to this policy as *Uncontrolled Random*. Uncontrolled Random is *unstable* for a

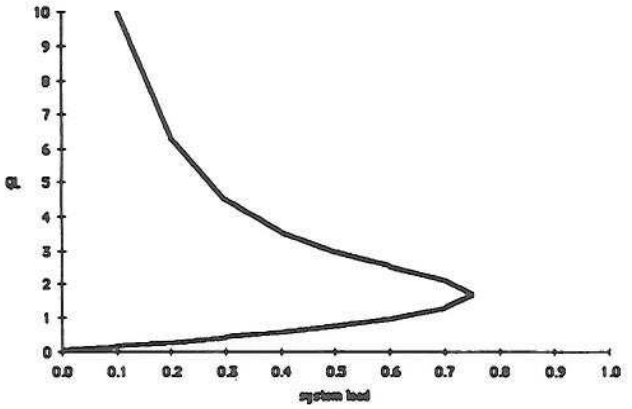


Fig. 10. Uncontrolled Random: queue length equilibrium contour. S(task service time) = 1. C(cost of task transfer) = 0.1. T(threshold) = 2.

nonzero transfer cost. No matter what the average load ρ, the system does not approach an equilibrium behavior: the expected backlog of work increases monotonically with time. Intuitively, this behavior occurs since there is a positive probability that all nodes will be in a transferring phase simultaneously. In Uncontrolled Random, each node would then begin to receive transferred tasks at rate $1/C$, and would try to retransfer these tasks at the same rate. Since no useful work is being done, the queue size at each node would increase at rate λ.

The instability of Uncontrolled Random is analogous to that of the infinite population ALOHA system [5]. Much of the terminology regarding stability that has been developed for use in the ALOHA context will be used here.

Fig. 10 is a result of the analysis that follows in this Appendix. The figure shows the mean node queue length *equilibrium contour* as a function of the system load, for Uncontrolled Random with a threshold of 2 and a transfer cost of 0.1. The average task service time is fixed at one. The equilibrium contour is composed of *system equilibrium points*, at which the output traffic intensity of the system (defined as the throughput multiplied by the average task service time) equals the input traffic intensity (or loading factor). A *load line* is defined as a vertical line corresponding to a particular value of input traffic intensity. There may be none, one, or two points of intersection of an equilibrium contour with a particular load line. At the input traffic intensity $\rho_{\max}$ equal to the maximum possible output traffic intensity, there is only one intersection point. For values of ρ greater than $\rho_{\max}$, the system is overloaded and queue lengths grow without bound; in this case there are no intersection points. For values of ρ less then $\rho_{\max}$, there are two intersection points; the lower is termed the *system operating point* and the upper the *system saturation point*. Below the system saturation point, the tendency of the system is to return to the system operating point. However, once the system saturation point is exceeded (which occurs eventually due to random fluctuations), the performance of the system degrades rapidly.

The remainder of this Appendix consists of the analysis of the Uncontrolled Random policy for system equilibrium points. Here $p_n (0 \le n \le T)$ and $\bar{n}_p$ denote the conditional probability of queue length n, and the mean queue length, respectively, given that the node is in a processing phase. λ_t and $\lambda_t(n)$ denote unconditioned and conditioned arrival rates of transferred tasks, respectively.

In Uncontrolled Random, the arrival rate of transferred tasks is independent of the node state. Therefore, for all n:

$$\lambda_t(n) = \lambda_t. \tag{UR1}$$

The arrival rate of transferred tasks must equal the rate of task transfers. Also, since no tasks are processed in the transferring phase, the probability of being in this phase is equal to that portion of the node utilization that is due to transferring tasks, or $C\lambda_t$. Since only those tasks that arrive while the queue length is greater than or equal to T are transferred,

$$\lambda_t = [p_T(1 - C\lambda_t) + C\lambda_t](\lambda + \lambda_t).$$

Solving for p_T yields

$$p_T = \frac{\dfrac{\lambda_t}{\lambda + \lambda_t} - C\lambda_t}{1 - C\lambda_t}. \tag{UR2}$$

Consider now the birth–death model of Fig. 1. The conditional probability p_0 is given by

$$p_0 = \frac{1 - \rho - C\lambda_t}{1 - C\lambda_t}.$$

Using this expression, (UR1), and the formula for the solution of a birth–death model [4] yields, for $0 \le n \le T$,

$$p_n = \frac{1 - \rho - C\lambda_t}{1 - C\lambda_t}[S(\lambda + \lambda_t)]^n. \tag{UR3}$$

Equating the right-hand side of (UR3) for $n = T$ with the right-hand side of (UR2) gives

$$\frac{\lambda_t}{\lambda + \lambda_t} - C\lambda_t = (1 - \rho - C\lambda_t)[S(\lambda + \lambda_t)]^T. \tag{UR4}$$

Equation (UR4) has the solution $\lambda_t = (1/S) - \lambda$ for all $C \ge 0$. However, for each S in the region of interest ($C < S < (1/\lambda)$), this solution does not result in conditional state probabilities that sum to one, except for a special case value of λ that depends on S. This special value can be found by substituting this solution for λ_t into equation (UR3). Noting that the right-hand side is then independent of n, and that all of the conditional probabilities must therefore be equal, the conditional probabilities sum to one if and only if $p_n = 1/(T + 1)$. Making this substitution and solving for λ yields as the special case value:

$$\lambda = \frac{1 - \dfrac{C}{S}}{\dfrac{T + 1}{T} S - C}.$$

If $C = 0$ (and $\rho < 1$), (UR4) has exactly one valid so-

lution. Therefore, there is only one equilibrium point, and the system is stable. If $C > 0$, (UR4) has exactly two valid solutions for all positive values of ρ less than some value $\rho_{max} < 1$, one valid solution for $\rho = \rho_{max}$, and no valid solutions for $\rho > \rho_{max}$. The value ρ_{max} defines the maximum possible throughput of the system. For $\rho \geq \rho_{max}$ the system is overloaded. For $0 < \rho < \rho_{max}$ the system is unstable.

The solutions of equation (UR4) may be found numerically by any appropriate method. (The method used to obtain the numerical results of Fig. 10 is based on the bisection method of finding roots.) For each valid solution, the conditional state probabilities p_n are given by (UR3). The conditional mean queue length $\bar{n}_p$ is given by

$$\bar{n}_p = S(\lambda + \lambda_t)\left[\frac{1 - \rho - C\lambda_t}{1 - C\lambda_t}\right]\left[\frac{1 - [S(\lambda + \lambda_t)]^T}{[1 - S(\lambda + \lambda_t)]^2} - \frac{T[S(\lambda + \lambda_t)]^T}{1 - S(\lambda + \lambda_t)}\right]. \quad \text{(UR5)}$$

From this solution all of the performance measures of interest can be derived. For example, the mean response time R of tasks is given by

$$R = \frac{\bar{n}_p(1 - C\lambda_t) + TC\lambda_t}{\lambda} + \frac{\lambda_t}{\lambda}\left[\frac{C}{1 - C(\lambda + \lambda_t)}\right]. \quad \text{(UR6)}$$

The first term in (UR6) is derived by applying Little's equation [10], using expressions for the throughput and mean queue length of locally processed tasks. The throughput of processed tasks is given by λ. The mean queue length of tasks to be processed at the node is given by the mean queue length during a processing phase, multiplied by the probability of being in a processing phase, plus the queue length during a transferring phase, multiplied by the probability of being in a transferring phase. The second term in (UR6) is just the mean number of times a task must be transferred, multiplied by the mean delay experienced each time.

Appendix B
Solution of Models

This Appendix completes the analysis of the load sharing models introduced in Section II. As in Appendix A, $p_n(0 \leq n \leq T)$ and $\bar{n}_p$ denote the conditional probability of queue length n and the mean queue length, respectively, given that the node is in a processing phase; λ_t and $\lambda_t(n)$ denote unconditioned and conditioned arrival rates of transferred tasks, respectively. In addition, p^a_{T+} denotes the absolute probability of being in a transferring phase.

A. Random Policy

In the Random policy, the arrival rate of transferred tasks is independent of the node state. Therefore, for all n,

$$\lambda_t(n) = \lambda_t. \quad \text{(R1)}$$

The arrival rate of transferred taks must equal the rate of task transfers. Since only those tasks that arrive while the queue length is greater than or equal to T are transferred, under the constraint of the transfer limit L_t,

$$\lambda_t = \lambda\left[\sum_{l=1}^{L_t}[p_T(1 - p^a_{T+}) + p^a_{T+}]^l\right].$$

Note, in the above equation, that λ multiplied by the lth term in the summation gives the rate at which tasks that have already been transferred $l - 1$ times are transferred once more. Performing the summation yields

$$\lambda_t = [p_T(1 - p^a_{T+}) + p^a_{T+}]\,\lambda\left[\frac{1 - [p_T(1 - p^a_{T+}) + p^a_{T+}]^{L_t}}{1 - [p_T(1 - p^a_{T+}) + p^a_{T+}]}\right]. \quad \text{(R2)}$$

The probability of being in a transferring phase is just that portion of the node utilization due to performing task transfers and processing tasks that could not be transferred because of the transfer limit. Therefore,

$$p^a_{T+} = C\lambda_t + S\lambda[p_T(1 - p^a_{T+}) + p^a_{T+}]^{L_t+1}.$$

Using (R2) to substitute for the exponentiated term and solving for p^a_{T+} yields

$$p^a_{T+} = \frac{p_T S(\lambda + \lambda_t) - (S - C)\lambda_t}{1 - (1 - p_T)S(\lambda + \lambda_t)}. \quad \text{(R3)}$$

Consider now the birth–death model of Fig. 1. Using the formula for the solution of a birth–death model along with equation (R1) yields

$$p_T = \frac{[S(\lambda + \lambda_t)]^T[1 - S(\lambda + \lambda_t)]}{1 - [S(\lambda + \lambda_t)]^{T+1}}. \quad \text{(R4)}$$

Equation (R4) can be used to substitute for p_T in (R2) and (R3). Equation (R3) can then be used to substitute for p^a_{T+} in (R2), yielding a nonlinear equation in the single unknown λ_t. The solution of this equation may be found numerically by any appropriate method. (The method used to obtain numerical results is based on the bisection method of finding roots.) Once λ_t has been found, p_T is given by (R4), and p^a_{T+} is then given by (R3). From the formula for the solution of a birth–death model, the conditional mean queue length $\bar{n}_p$ is given by

$$\bar{n}_p = S(\lambda + \lambda_t)\left[\frac{1 - \rho - C\lambda_t}{1 - p^a_{T+}}\right]\left[\frac{1 - [S(\lambda + \lambda_t)]^T}{[1 - S(\lambda + \lambda_t)]^2} - \frac{T[S(\lambda + \lambda_t)]^T}{1 - S(\lambda + \lambda_t)}\right]. \quad \text{(R5)}$$

All of the performance measures of interest can now be computed. In particular, the mean response time R of tasks is given by:

$$R = \frac{\bar{n}_p(1 - p^a_{T^+}) + Tp^a_{T^+} + \lambda[p_T(1 - p^a_{T^+}) + p^a_{T^+}]^{L_t+1} R_{T^+_p}}{\lambda} + \frac{\lambda_t}{\lambda} R_{T^+_t} \tag{R6}$$

where $R_{T^+_t}$ denotes the mean delay experienced in being transferred, and $R_{T^+_p}$ denotes the processing delay for a task that arrives at the node when the node is at or over threshold, and yet is not transferred due to the transfer limit. The first term in (R6) is derived by applying Little's equation, using expressions for the throughput and mean queue length of locally processed tasks. The throughput of processed tasks is given by λ. The mean queue length of tasks to be processed at the node is given by the mean queue length during a processing phase, multiplied by the probability of being in a processing phase, plus the threshold multiplied by the probability of being in a transferring phase, plus the mean queue length of tasks that are processed locally only because of the transfer limit (which is given by the arrival rate of such tasks multiplied by their mean delay). The second term in (R6) is just the mean number of times a task must be transferred, multiplied by the mean delay experienced each time.

Expressions for $R_{T^+_t}$ and $R_{T^+_p}$ are derived from the solution of a preemptive priority HOL M/M/1 queue [4]. $R_{T^+_t}$ is given by

$$R_{T^+_t} = \frac{C}{1 - C\dfrac{\lambda_t}{p_T(1 - p^a_{T^+}) + p^a_{T^+}}}. \tag{R7}$$

$R_{T^+_p}$ is given by

$$R_{T^+_p} = \frac{S - (S - C)\, C\dfrac{\lambda_t}{p_T(1 - p^a_{T^+}) + p^a_{T^+}}}{\left[1 - C\dfrac{\lambda_t}{p_T(1 - p^a_{T^+}) + p^a_{T^+}}\right]\left[1 - C\dfrac{\lambda_t}{p_T(1 - p^a_{T^+}) + p^a_{T^+}} - S\lambda[p_T(1 - p^a_{T^+}) + p^a_{T^+}]^{L_t}\right]}. \tag{R8}$$

B. Threshold Policy

In the model of the Threshold policy, all transferred tasks arrive when the node queue length is less than the threshold T. Therefore, $\lambda_t(T)$ and $\lambda_t(T^+)$ are both zero. When the node queue length is less than T, the arrival rate of transferred tasks is independent of the node state. Since the probability that the node queue length is less than T is $(1 - p_T)(1 - p^a_{T^+})$, it must be the case that, for $0 \le n \le T - 1$,

$$\lambda_t(n) = \lambda_t^* \tag{T1}$$

where λ_t^* is defined by

$$\lambda_t^* = \frac{\lambda_t}{(1 - p_T)(1 - p^a_{T^+})}. \tag{T2}$$

The arrival rate of transferred tasks must equal the rate of task transfers. Since only those tasks that arrive while the queue length is greater than or equal to the threshold T are transferred, under the constraint of a probe limit of L_p,

$$\lambda_t = [p_T(1 - p^a_{T^+}) + p^a_{T^+}]\,\lambda \cdot [1 - [p_T(1 - p^a_{T^+}) + p^a_{T^+}]^{L_p}].$$

This gives

$$\lambda_t^* = \frac{[p_T(1 - p^a_{T^+}) + p^a_{T^+}]}{(1 - p_T)(1 - p^a_{T^+})}\lambda \cdot [1 - [p_T(1 - p^a_{T^+}) + p^a_{T^+}]^{L_p}]. \tag{T3}$$

The probability of being in a transferring phase is just that portion of the node utilization due to performing task transfers and processing tasks that could not be transferred because of a failure to find a suitable destination. Therefore,

$$p^a_{T^+} = C\lambda_t + S\lambda[p_T(1 - p^a_{T^+}) + p^a_{T^+}]^{L_p+1}.$$

Using (T3) to substitute for the exponentiated term and solving for $p^a_{T^+}$ yields

$$p^a_{T^+} = \frac{p_T S\lambda - (1 - p_T)(S - C)\,\lambda_t^*}{1 - (1 - p_T)(S\lambda + (S - C)\,\lambda_t^*)}. \tag{T4}$$

Consider now the birth–death model of Fig. 1. Using the formula for the solution of a birth–death model along with (T1) yields

$$p_T = \frac{[S(\lambda + \lambda_t^*)]^T (1 - S(\lambda + \lambda_t^*))}{1 - [S(\lambda + \lambda_t^*)]^{T+1}}. \tag{T5}$$

Equation (T5) can be used to substitute for p_T in (T3) and (T4). Equation (T4) then can be used to substitute for $p^a_{T^+}$ in (T3), yielding a nonlinear equation in the single unknown λ_t^*. The solution of this equation may be found numerically by any appropriate method. (The method used to obtain numerical results is based on the bisection method of finding roots.) Once λ_t^* has been found, p_T is given by (T5), $p^a_{T^+}$ is then given by (T4), and λ_t is then given by (T2).

From the formula for the solution of a birth–death model, the conditional mean queue length $\bar{n}_p$ is given by

$$\bar{n}_p = S(\lambda + \lambda_t^*)\left[\frac{1 - \rho - C\lambda_t}{1 - p^a_{T^+}}\right]\left[\frac{1 - [S(\lambda + \lambda_t^*)]^T}{[1 - S(\lambda + \lambda_t^*)]^2} - \frac{T[S(\lambda + \lambda_t^*)]^T}{1 - S(\lambda + \lambda_t^*)}\right]. \tag{T6}$$

The mean response time R of tasks is given by

$$R = \frac{\bar{n}_p(1 - p^a_{T+}) + Tp^a_{T+} + \lambda[p_T(1 - p^a_{T+}) + p^a_{T+}]^{L_p+1} R_{T+p}}{\lambda} + \frac{\lambda_t}{\lambda} R_{T+t} \quad \text{(T7)}$$

where R_{T+t} denotes the mean delay experienced in being transferred, and R_{T+p} denotes the processing delay for a task that arrives at the node when the node is at or over threshold, and yet is not transferred since a suitable destination has not been found after the maximum number of probes. This equation is quite similar to that for the Random policy, and is derived in a similar manner.

Expressions for R_{T+t} and R_{T+p} are derived from the solution of a preemptive priority HOL M/M/1 queue. R_{T+t} is given by exactly the same equation as for the Random policy

$$R_{T+t} = \frac{C}{1 - C\dfrac{\lambda_t}{p_T(1 - p^a_{T+}) + p^a_{T+}}}. \quad \text{(T8)}$$

R_{T+p} is given by

$$R_{T+p} = \frac{S - (S - C)\, C\dfrac{\lambda_t}{p_T(1 - p^a_{T+}) + p^a_{T+}}}{\left[1 - C\dfrac{\lambda_t}{p_T(1 - p^a_{T+}) + p^a_{T+}}\right]\left[1 - C\dfrac{\lambda_t}{p_T(1 - p^a_{T+}) + p^a_{T+}} - S\lambda[p_T(1 - p^a_{T+}) + p^a_{T+}]^{L_p}\right]}. \quad \text{(T9)}$$

C. Shortest Policy

Iteration is used to evaluate the model of the Shortest policy. In a typical step, a model solution is used to derive new values for the arrival rates of transferred tasks, and a new solution is computed. In the following description of the iteration equations, $pshort_n$ denotes the probability that a node with queue length n is selected when attempting to find a suitable node to which to transfer a task. The following equation gives $pshort_n$, $0 \le n \le T - 1$, in terms of the probe limit L_p, the conditional state probabilities p_n, and the probability of being in a transferring phase p^a_{T+}:

$$pshort_n = \left[1 - \left(\sum_{m=0}^{n-1} p_m\right)(1 - p^a_{T+})\right]^{L_p} - \left[1 - \left(\sum_{m=0}^{n} p_m\right)(1 - p^a_{T+})\right]^{L_p}. \quad \text{(S1)}$$

The first term in (S1) gives the probability that all of the L_p randomly chosen nodes have queue length greater than or equal to n; the second term gives the probability that all of the L_p randomly chosen nodes have queue length greater than or equal to $n + 1$. The difference of the two terms provides the required probability. Noting that only those tasks that arrive when a node is at or over threshold can be transferred, under the constraint of a probe limit of L_p, and that the arrival rate of transferred tasks must equal the rate of task transfers,

$$\lambda_t = [p_T(1 - p^a_{T+}) + p^a_{T+}]\,\lambda \cdot [1 - [p_T(1 - p^a_{T+}) + p^a_{T+}]^{L_p}]. \quad \text{(S2)}$$

The probability of being in a transferring phase is just that portion of the node utilization due to performing task transfers and processing tasks that could not be transferred because of a failure to find a suitable destination. Therefore:

$$p^a_{T+} = C\lambda_t + S\lambda[p_T(1 - p^a_{T+}) + p^a_{T+}]^{L_p+1}. \quad \text{(S3)}$$

The arrival rates $\lambda_t(n)$, for $0 \le n \le T - 1$, are then given by

$$\lambda_t(n) = \frac{\lambda_t}{p_n(1 - p^a_{T+})} \frac{pshort_n}{1 - [p_T(1 - p^a_{T+}) + p^a_{T+}]^{L_p+1}}. \quad \text{(S4)}$$

The formula for the solution of a birth–death model yields, for $1 \le n \le T$,

$$p_n = p_{n-1}\left[1 + \frac{\lambda_t(n - 1)}{\lambda}\right]\rho. \quad \text{(S5)}$$

Finally, p_0 is given by

$$p_0 = \frac{1 - \rho - C\lambda_t}{1 - p^a_{T+}}. \quad \text{(S6)}$$

In one iteration of the method used to compute numerical results, (S1)–(S6) are applied to a model solution in order, yielding a new model solution. The conditional state probabilities of the new solution then are normalized to sum to one by scaling the probabilities p_k for $k > 0$. The iteration stopping criterion is based on comparing old and new conditional mean queue length values, as obtained from the conditional state probabilities. Empirically, the iteration is insensitive to the initializations used. Once the solutions of (S1)–(S6) have been obtained, the mean response time R of tasks is given by the same equations as in the analysis of the Threshold policy.

Appendix C
Simulation Results

Experimentation with an event-driven simulation program has provided validation of the decomposition approximation utilized in our analytic models. The simulation program uses the same system model as do the analytic models, but does not make the decomposition approximation.

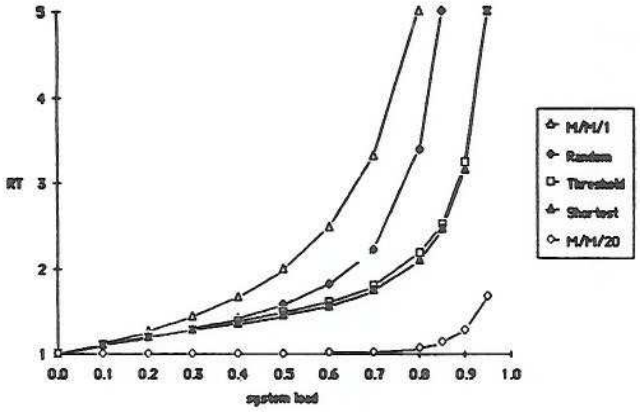


Fig. 11. Simulation results: response time versus load ρ. S(task service time) = 1. C(cost of task transfer) = 0.1. T(threshold) = 2. L_p(probe limit for Threshold and Shortest) = 3. L_t(transfer limit for Random) = 1.

In Fig. 11 we present a small sample of the results of our simulation experiments. A system with 20 nodes has been simulated. Fig. 11 should be compared to Fig. 2; for the Random, Threshold, and Shortest policies the former figure shows the simulation results that correspond to the analytic results of the latter figure. (Fig. 11 also includes the analytic results for M/M/1 and M/M/20 for comparison purposes.) Note the close correspondence between the two figures, both with respect to the absolute values of the performance measures (particularly at low to moderate loadings), and with respect to the indicated relative performance of the three load sharing policies.

Acknowledgment

D. Towsley of the University of Massachusetts discussed these issues extensively with us. K. Sevcik of the University of Toronto provided comments on an earlier draft.

References

[1] A. Barak and A. Shiloh, "A distributed load balancing policy for a multicomputer," Dep. Comput. Sci., Hebrew Univ. of Jerusalem, Jerusalem, Israel, 1984.

[2] S. H. Bokhari, "Dual processor scheduling with dynamic reassignment," *IEEE Trans. Software Eng.*, vol. SE-5, pp. 341–349, July 1979.

[3] R. Bryant and R. A. Finkel, "A stable distributed scheduling algorithm," in *Proc. 2nd Int. Conf. Distributed Comput. Syst.*, 1981, pp. 314–323.

[4] L. Kleinrock, *Queueing Systems: Volume I—Theory*. New York: Wiley, 1976.

[5] L. Kleinrock and S. S. Lam, "Packet switching in a multiaccess broadcast channel: Performance evaluation," *IEEE Trans. Commun.*, vol. COM-23, pp. 410–423, Apr. 1975.

[6] A. Kratzer and D. Hammerstrom, "A study of load levelling," in *Proc. IEEE Fall COMPCON*, 1980, pp. 647–654.

[7] P. Krueger and R. A. Finkel, "An adaptive load balancing algorithm for a multicomputer," Dep. Comput. Sci., Univ. Wisconsin, Madison, Tech. Rep. 539, Apr. 1984.

[8] S. S. Lam and L. Kleinrock, "Packet switching in a multiaccess broadcast channel: Dynamic control procedures," *IEEE Trans. Commun.*, vol. COM-23, pp. 891–904, Sept. 1975.

[9] E. D. Lazowska, J. Zahorjan, D. R. Cheriton, and W. Zwaenepoel, "File access performance of diskless workstations," Dep. Comput. Sci., Univ. Washington, Seattle, Tech. Rep. 84-06-01, June 1984.

[10] J. D. C. Little, "A proof of the queueing formula $L = \lambda W$," *Oper. Res.*, vol. 9, pp. 383–387, May 1961.

[11] M. Livny and M. Melman, "Load balancing in homogeneous broadcast distributed systems," in *Proc. ACM Comput. Network Performance Symp.*, 1982, pp. 47–55.

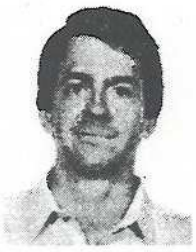
[12] M. L. Powell and B. P. Miller, "Process migration in DEMOS/MP," in *Proc. 9th ACM Symp. Operat. Syst. Principles*, 1983, pp. 110–119.

[13] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Software Eng.*, vol. SE-3, pp, 85–93, Jan. 1977.

[14] ——, "Critical load factors in two processor distributed systems," *IEEE Trans. Software Eng.*, vol. SE-4, pp. 254–258, May 1978.

[15] A. N. Tantawi and D. Towsley, "Optimal static load balancing in distributed computer systems," *J. ACM*, vol. 32, pp. 445–465, Apr. 1985.

[16] F. A. Tobagi and L. Kleinrock, "Packet switching in radio channels: Part IV—Stability considerations and dynamic control in carrier sense multiple access," *IEEE Trans. Commun.*, vol. COM-25, pp. 1103–1119, Oct. 1977.



Derek L. Eager received the B.Sc. degree in computer science from the University of Regina, Regina, Sask., Canada, in 1979, and the M.Sc. and Ph.D. degrees in computer science from the University of Toronto, Toronto, Ont., Canada, in 1981 and 1984, respectively.

He is currently an Assistant Professor in the Department of Computational Science at the University of Saskatchewan, Saskatoon. His research interests are in the areas of performance modeling and distributed systems.

Edward D. Lazowska received the A.B. degree from Brown University, Providence, RI, in 1972 and the Ph.D. degree in computer science from the University of Toronto, Toronto, Ont., Canada, in 1977.

Since 1977 he has been on the faculty of the Department of Computer Science at the University of Washington, Seattle, where he recently returned after a sabbatical leave at Digital Equipment Corporation's Systems Research Center. His research interests fall within the general area of computer systems: modeling and analysis, design and implementation, and distributed systems.

Dr. Lazowska is the Chairman of SIGMETRICS, the Association for Computing Machinery's Special Interest Group concerned with computer system performance.

John Zahorjan received the Sc.B. degree in applied mathematics from Brown University, Providence, RI, in 1975, and the M.Sc. and Ph.D degrees in computer science from the University of Toronto, Toronto, Ont., Canada, in 1976 and 1980, respectively.

Presently, he is an Associate Professor of Computer Science at the University of Washington, Seattle. His active research interests include performance modeling of computer systems, load sharing policies in distributed systems, and issues in naming in distributed systems. He is an author of papers in these areas, and is coauthor of a recent book on performance modeling of computer systems using queueing network models.