Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts

KANG G. SHIN, SENIOR MEMBER, IEEE, AND YI-CHIEH CHANG, STUDENT MEMBER, IEEE

Abstract—**If task arrivals are not uniformly distributed over the nodes in a distributed real-time system, some nodes may become overloaded while others are underloaded. Consequently, some tasks cannot be completed before their deadlines, even if the overall system has the capacity to meet all deadlines. Load sharing (LS) is one way to alleviate this problem.**

In this paper, we propose a decentralized, dynamic LS method for a distributed real-time system. Whenever the state of a node changes from underloaded to fully-loaded and vice versa, the node broadcasts this change to a set of nodes, called a *buddy set*, in the system. An overloaded node can select, without probing other nodes, the first available node from its *preferred list*, an ordered set of nodes in its buddy set. Preferred lists are so constructed that the probability of more than one overloaded node "dumping" their loads on a single underloaded node may be made very small.

Performance of the proposed LS method is evaluated with both analytic modeling and simulation. The LS method is modeled first by an embedded Markov chain to which numerical solutions are derived. The model solutions are then used to calculate the distribution of queue length at each node and the probability of meeting task deadlines. Our analytic results show that buddy sets of ten nodes outperform those of less than ten nodes, and the incremental benefit gained from increasing the buddy set size beyond 15 nodes is insignificant. These and other analytic results are verified with simulation. The proposed LS method is shown to meet task deadlines with a very high probability.

Index Terms—**Buddy set, deadlines, distributed real-time systems, load sharing, missing probability, preferred list, state-change broadcasts.**

I. INTRODUCTION

FAILURE to complete a real-time task before its deadline could cause a disaster [1], [2]. Due to their potential for high performance and reliability with the multiplicity of processors, distributed systems are natural candidates to implement real-time applications. However, if task arrivals are unevenly distributed over the nodes in a distributed real-time system, some nodes may become overloaded, and thus, unable to complete all their tasks in time, while other nodes are underloaded. In such a case, even if the total processing power of the system is sufficient to complete all incoming tasks in time, some tasks arriving at overloaded nodes may not complete in time. One way to alleviate this problem is load sharing (LS); some of those tasks arriving at overloaded nodes are transferred to underload nodes for execution.

LS in general-purpose distributed systems has been studied extensively by numerous researchers [3]–[8]. Decisions on how to share loads among the nodes are either *static* or *dynamic*. A static decision is independent of the current system state, whereas a dynamic decision depends on the system state at the time of decision. Static LS can also be viewed as nondeterministic allocation of tasks in a system [8]–[10], where an overloaded node N_i will transfer some of its tasks to node N_j with probability P_{ij}, which is independent of the current system state. Although static LS is simple and easy to analyze with queueing models, its potential benefit is limited since it does not adapt itself to the time-varying system state [4]. For example, even when N_i is overloaded, it still has to accept tasks from other nodes with the same probability as if it were underloaded. On the other hand, when dynamic LS is used, an overloaded node can transfer its task(s) to other node(s) using the information on the current system state [4], [6], [7], [11]–[13]. Since any dynamic policy requires each node to know states of the other nodes, it is inherently more complex than any static policy. The advantage of a dynamic policy is that it adapts itself to the time-varying system state, and thus, can ease the difficulty associated with static LS.

LS algorithms can be source-initiated or server-initiated, depending on which node initiates task transfer. The node at which external tasks arrive is the source (sender) node, and the node that processes these tasks is the server (receiver) [3]. In the source-initiated approach, an overloaded source node initiates the transfer of a newly arriving external task based on some strategies, while in the server-initiated approach, an underloaded server will probe each of the potential source nodes to share its load with. LS algorithms are further divided into several levels according to the amount of information required for them. After analyzing and comparing the performance of these algorithms, Wang and Morris [3] concluded that an algorithm that collects more information will generally produce better results and that the server-initiated approach will usually outperform the source-initiated approach if the task transfer cost is not significant.

As was discussed by Eager *et al.* [4], LS is composed of a *transfer policy* and a *location policy*. The transfer policy determines when a node should transfer its task(s), i.e., when a node becomes overloaded. The location policy determines

Manuscript received September 25, 1988; revised May 17, 1989. This work was supported in part by the Office of Naval Research under Contract N00014-85-K-0122, by NASA under Grant NAG-1-887, and by the NSF under Grant DMC-8721492. Any opinions, findings, and recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the funding agencies.

The authors are with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI48108.

IEEE Log Number 8928530.

where to send task(s) to. Their objective was to minimize the average system response time by moving tasks from overloaded nodes to underloaded ones. A simple threshold was used in the transfer policy; that is, whenever the queue length of a node exceeds this threshold, it will attempt to transfer its incoming task to another node. Three different location policies were simulated and compared: random, threshold, and shortest. Their simulation results indicate that the random policy improves system performance significantly, as compared to the system without LS. The threshold policy can further improve system performance, as compared to the random policy. The improvement of the shortest policy is about the same as that of the threshold policy, although it requires more system state information than that of the threshold policy.

LS in distributed real-time systems is addressed far less in literature than that in general-purpose distributed systems. Kurose and Chipalkatti proposed a quasi-dynamic LS algorithm in a soft real-time system [5]. A job is considered *lost* if it is not completed before its deadline. Their primary objective was to reduce the probability of losing a job with LS. A node will transfer some of its jobs to another node if the unfinished workload exceeds its time constraints. The server was selected on a probabilistic basis which is independent of the current system state.

The location policy in most early work can be viewed as sender-initiated, since an overloaded node (sender) selects another node (a candidate receiver) and checks whether or not this node can share its load. If it can, the sender will transfer some of its tasks to that node; otherwise, the sender will probe another node. This process will repeat until a receiver is found or a prespecified limit is reached. There are two major drawbacks associated with this approach. First, the sender needs to probe other nodes before transferring any of its tasks. This will introduce an additional delay in completing the tasks to be transferred. Second, if only a few nodes in the system are underloaded, the sender may not be able to locate a receiver by probing only a limited number of nodes. In such a case, overloaded nodes must execute all their tasks locally, missing the deadlines of some of these tasks.

In a real-time system, the probability of missing a task must be kept as low as possible because the loss of a task may lead to a disastrous circumstance [1]. Note that almost all LS methods known to date are concerned only with the *average* system performance, rather than the performance of each individual task. To alleviate this weakness, we propose a new LS method in which each node needs to maintain state information of only a small set of nodes, called a *buddy set*. Three thresholds, denoted by TH_u, TH_f, and TH_v, are used to define the (loading) state of a node. A node is said to be *underloaded* if its queue length (QL) is less than or equal to TH_u, *medium-loaded* if $TH_u < QL \leq TH_f$, *fully-loaded* if $TH_f < QL \leq TH_v$, and *overloaded* if $QL > TH_v$. Whenever a node becomes fully-loaded (underloaded) due to the arrival and/or transfer (completion) of tasks, it will broadcast its change of state to all the other nodes in its buddy set. Every node that receives this information will update its state information by eliminating the fully-loaded node from, or adding the underloaded node to, its ordered list (called a *preferred list*) of available receivers. An overloaded node can select the first node in its preferred list and transfer a task to that node. Notice that our LS method is completely different from conventional receiver-initiated LS methods that are characterized in [3]. An underloaded server in the conventional receiver-initiated approach probes other nodes to share their work with. By contrast, our method transfers tasks from overloaded nodes to underloaded nodes using state-change broadcasts within their respective buddy sets.

The proposed LS method is modeled with an embedded Markov chain to which numerical solutions are obtained by a two-step approximation approach. The numerical solutions are then used to compute the distribution of QL at each node and the probability of meeting deadlines. Use of a preferred list results in transferring "surplus" tasks[1] only to the first few nodes in the list. More importantly, different buddy sets are designed to overlap one another so that tasks may be distributed evenly over the entire system, rather than within one buddy set. Thus, buddy sets each with 10–15 nodes are usually sufficient for load sharing regardless of system size.

The rest of this paper is organized as follows. Section II discusses the problem of implementing the proposed LS method. Collection of state information and construction of preferred lists are detailed in Section III. Section IV presents one exact model, one approximate model, and an approximate solution to the exact model for the proposed LS method. In Section V, the performance of the proposed LS method is evaluated with the models derived in Section IV and is also simulated to verify the analytic results. Finally, the paper concludes with Section VI.

II. Problem Statement

In the proposed LS method, each node must maintain and update the state information of other nodes. An overloaded node can transfer a task to another node based on state information without any probing delay. To implement this method, one must develop:

- Efficient means of collecting and updating state information; collection of state information must not hamper normal communications, such as intertask communications and task transfers.
- An automatic method for selecting a server node in case there is more than one underloaded node, and minimizing the probability of more than one overloaded node simultaneously transferring their surplus tasks to the same underloaded node.

These issues are addressed below in some detail.

A. State Information

To collect state information, one must decide from which nodes the information should be collected and how often this information should be updated. One straightforward method is for each node to collect and update state information from all other nodes in the system at a fixed time interval. However, it is very difficult to determine an appropriate collection and update interval which ensures the accuracy of state informa-

[1] Those tasks arriving at a node that cannot be completed by the node before their deadlines.

tion while keeping below an acceptable level the additional network traffic resulting from the collection of state information. Although a short interval (i.e., frequent state update) ensures the accuracy of state information, this will introduce an $O(n^2)$ traffic overhead each time to collect state information, where n is the number of nodes in the system. This may, in turn, severely delay normal intertask communications and task transfers, thus degrading (rather than improving) system performance. On the other hand, the traffic overhead decreases as the frequency of state collection and update decreases. But, this may cause the state information recorded in a node to be obsolete. For example, if the state of a node has changed from underloaded to fully-loaded before the next update, other nodes may transfer their tasks to this already fully-loaded node based on the obsolete state information.

Ideally, each node should keep state information as accurate and as up-to-date as possible while keeping the traffic overhead as low as possible. To achieve this goal, we propose *state-change broadcasts* within the buddy set of each node to collect and update the state information at the node. Thus, each node needs to maintain only the state information of a small set of nodes in the system, e.g., neighbors of the node. Each node will broadcast the change of state to all the other nodes in its buddy set only if it switched from underloaded to fully-loaded, and vice versa. Since a node will receive new information only when the state of a node in its buddy set has been changed, each node will have the exact state information of all the other nodes in its buddy set, as long as there is no significant delay in broadcasting state changes. Since the buddy set of each node is formed by those nodes in its physical proximity, the delay in broadcasting a state-change and/or transferring a task should not, in general, be too long. (See Section V-A7 for more on the broadcasting delay.) Moreover, as we shall see, the additional network traffic resulting from state-change broadcasts can be controlled by adjusting the two thresholds, TH_u and TH_f.

B. Preferred List

In the sender-initiated location policy, an overloaded node will transfer a task to the underloaded node found first during the probing [3]–[5]. The problem in our proposed location policy is that an overloaded node may find more than one underloaded node in its buddy set and/or more than one overloaded node could select the same node to transfer their tasks to. One must therefore establish a rule for selecting a receiver among possible multiply underloaded nodes while minimizing the probability of more than one overloaded node simultaneously transferring tasks to the same underloaded node. A *preferred list* is proposed to counter the above problem. Since each node maintains the state information of the nodes in its buddy set, one can order these nodes in each node's preferred list. The first node in this list is the *most preferred* and the second the *second most preferred*, and so on. Note that order of preference changes with time, e.g., if the most preferred node becomes overloaded, then the second most preferred node, if not overloaded, becomes the most preferred. An overloaded node will transfer its task(s) to its most preferred node available in the list.

Based on the system topology, the "static" order of nodes in each node's preferred list is so permuted that a node is the most preferred of one and only one other node in the corresponding buddy set. (This order does not change with time, although some of nodes will drop out of the list of available receivers when they become fully-loaded, and regain their spots when they become underloaded again.) Since each overloaded node is most likely to select the first node in its preferred list, the problem of more than one overloaded node "dumping" their loads on one node is unlikely to occur. Nevertheless, dumping could occur since, for example, the third most preferred node, say N_x, of an overloaded node N_o can become its most preferred while N_x is also the most preferred of another overloaded node. But the probability of this happening is small, since it will occur only after overloading all the nodes ahead of N_x in N_o's preferred list.

III. State Information and Preferred List

The nodes in the system are connected by an arbitrary network, and each node is equipped with a network processor which handles the usual communications and task transfers between nodes without burdening the node processor. A node has two sources of task arrivals, external tasks and transferred-in tasks, and one server (single node processor). The tasks arriving at each node may be executed locally, or remotely, at any other nodes in the system.

Every node is assumed to be *stable*, or its load density (i.e., the ratio of average external arrival rate to average service rate) is less than one. Thus, the need of load sharing arises when there are bursty arrivals of external tasks at one or more nodes in the system.

A. Collection of State Information

As mentioned earlier, three thresholds, TH_u, TH_f, and TH_v, are used to determine the state of a node. These thresholds can be QL or cumulative execution time (CET), depending on task characteristics. For example, if every task has the same (identically distributed) execution time, one can use the (average) QL to measure the workload of each node. However, if task execution times are neither identical nor identically distributed, one must use the CET to measure the workload of each node. By comparing each node's current workload to these thresholds, the node is determined to be in one of four possible states: under (U), medium (M), full (F), and over (V). QL will be used to measure a node's workload throughout this paper. (Use of CET is much more involved, as pointed out in Section VI, and will be treated in a forthcoming paper.) A node is in U state if $\mathrm{QL} \leq \mathrm{TH}_u$, M state if $\mathrm{TH}_u < \mathrm{QL} \leq \mathrm{TH}_f$, F state if $\mathrm{TH}_f < \mathrm{QL} \leq \mathrm{TH}_v$, and V state if $\mathrm{QL} > \mathrm{TH}_v$.

A U-state node can accept one or more tasks from other nodes and complete them before their deadlines. A node in F state cannot accept tasks from any other nodes but can complete all of its own tasks in time. A node in V state cannot complete all of its own tasks in time, and thus, must transfer, if possible, some of its tasks to other node(s). Since a node in U state can share other nodes' work, it is said to be in *share mode*. A node in F state will neither accept tasks from other

nodes nor transfer its own tasks to others, and is said to be in *independent mode*. A node in V state must transfer some of its tasks, and is said to be in *transfer mode*. Note that V is usually a transient state because, if arrival of a new task at a node switches the node to V state, the node will transfer this task to another underloaded node and then switch back to F state. However, if a V-state node cannot find a U-state node from its buddy set, it will be forced to remain in V state, missing some deadlines.

According to our state-change broadcasts, each node will broadcast the change of state to all the other nodes in its buddy set only when it switches from U to F and/or U to F. Upon receiving a state-change broadcast, every node in the corresponding buddy set will update its state information accordingly.

Two different thresholds, TH_u and TH_f, are used in the proposed LS method for the following reason. If only one threshold were used instead, a node would be in U (F) state when its QL is less (greater) than this threshold. In such a case, a U-state node may switch to F state after receiving a task from another node, and an F-state may switch back to U state after completing a task. Since a U-state node will accept tasks from other overloaded nodes, it is likely to switch to F state. On the other hand, an F-state node only accepts its own external tasks and is likely to switch back to U state, since every node is assumed to be stable. Thus, every node in the system will frequently switch between U and F, thereby increasing the network traffic to broadcast these state changes. Change of state occurs infrequently (frequently) when the difference between TH_u and TH_f is large (small).

The third threshold, TH_v, is used to avoid unnecessary task transfers. If we combine TH_f and TH_v into one threshold, then acceptance of one transferred task may make a node fully- as well as overloaded. In this case, the fully- and overloaded node must transfer its own newly arriving task to another node. Had it not accepted the transferred task, the node would not have to transfer the newly arriving task, and thus, one of the two task transfers would not have been needed. By introducing another threshold TH_v, each node will broadcast the change of state when it switches to F state, preventing other nodes from transferring tasks to that node. Since $TH_v - TH_f \neq 0$ and every node is assumed to be stable, a node is unlikely to become overloaded with its own arriving tasks. Thus, this difference can be used to control unnecessary task transfers.

The above three thresholds greatly influence system performance, such as the average task execution time, the probability of missing deadlines, and the traffic overhead of broadcasting state changes. These thresholds must therefore be determined to meet the system performance requirement. For example, in a real-time system, TH_v is a critical point below which a node processor can complete all queued tasks before their deadlines with a probability higher than required. The difference between TH_u and TH_f must be chosen to keep the traffic overhead induced by state-change broadcasts below a specified value. These thresholds are also sensitive to system load and have to be adjusted as system load varies. (More on this will be discussed in Section V.)

B. List of Preferred Nodes

As mentioned in Section II, the purpose of constructing a preferred list for each node is to avoid the probing delay and the dumping problem. The cost of task transfer is an increasing function of the physical distance between the sender and receiver nodes. To reduce this cost, the receiver node should be located as closely to the source node as possible. The preferred list of each node is thus structured based on the number of hops between the source and receiver nodes. The first entry of a node's preferred list consists of its immediate neighbors, and the second entry consists of those nodes two hops away from the node, and so on. When there is more than one node in each entry, these nodes must be ordered to minimize the dumping problem.

To demonstrate how to order the nodes in each buddy set based on system topology, consider a regular[2] system with n nodes, $N_1, N_2, \cdots, N_n$, where the degree of N_i is k, $\forall i$. Link j of N_i is assigned a direction d_j, $0 \leq j \leq k-1$. N_i's "static" preferred list[3] is then constructed as follows. The set of N_i's immediate neighbors, denoted by P_1^i, is placed in the first entry of N_i's preferred list. The N_i's second entry, denoted by P_2^i, consists of the nodes in the first entry of every node in P_1^i, excluding the duplicated nodes. Generally, P_l^i is the set of nodes which are listed in the first entry of every node in P_{l-1}^i, excluding the duplicated nodes.

Among the nodes in P_1^i, the node in direction d_0 is chosen to be the N_i's most preferred node in this entry, denoted by N_1^{i1}, and the node in direction d_1 is the N_i's second most preferred node in this entry, denoted by N_2^{i1}, and so on. The nodes in P_2^i are ordered as follows. The nodes in the N_1^{i1}'s first entry are checked according to their order in the entry. If a node in the N_1^{i1}'s first entry did not appear at any N_i's previous entry, it will be copied into the second entry of N_i in the same order as in the N_1^{i1}'s first entry. After all nodes in the first entry N_1^{i1} are checked and copied, the nodes in the first entry of node N_2^{i1} will be checked and copied by the same procedure. This procedure will repeat until P_2^i is completed. The ordering of nodes in P_l^i, $\forall l > 2$, can be determined similarly.

As an example, consider how the preferred list of each node in a four-cube system (Fig.1) is actually constructed. The identity (ID) of each node is coded with a 4-bit number, $b_3b_2b_1b_0$. The direction d_i of N_k is the link that connects N_k to a node whose ID differs from N_k's ID in bit position i, where $0 \leq i \leq 3$. One can now apply the above procedure to construct the preferred list for each node in the four-cube system as shown in Fig. 2.

Once each node's preferred list is constructed, an overloaded node N_i can select an underloaded node as follows. Check N_1^{i1} first; if it is underloaded, N_i will transfer a task to N_1^{i1}; otherwise, N_2^{i1} is checked, and so on. (This checking can easily be implemented with a pointer which is made to point to the first available node in the list.) If all the nodes in P_1^i are overloaded, N_i will sequentially check the nodes in P_2^i. If, albeit rare, an overloaded node cannot find any underloaded

[2] A system is said to be *regular* if all node degrees are identical.

[3] This list is determined by the system topology and remains unchanged, but the availability of each node in this list changes with time .

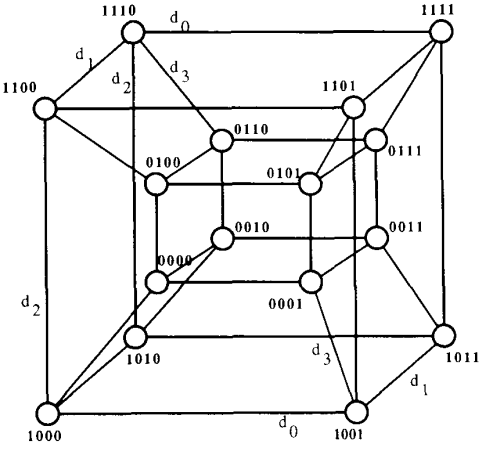


Fig. 1. A four-cube system.

Order of preference	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
node 0	1	2	4	8	6	10	12	3	5	9	14	13	11	7	15
node 1	0	3	5	9	7	11	13	2	4	8	15	12	10	6	14
node 2	3	0	6	10	4	8	14	1	7	11	12	15	9	5	13
node 3	2	1	7	11	5	9	15	0	6	10	13	14	8	4	12
node 4	5	6	0	12	2	14	8	7	1	13	10	9	15	3	11
node 5	4	7	1	13	3	15	9	6	0	12	11	8	14	2	10
node 6	7	4	2	14	0	12	10	5	3	15	8	11	13	1	9
node 7	6	5	3	15	1	13	11	4	2	14	9	10	12	0	8
node 8	9	10	12	0	14	2	4	11	13	1	6	5	3	15	7
node 9	8	11	13	1	15	3	5	10	12	0	7	4	2	14	6
node 10	11	8	14	2	12	0	6	9	15	3	4	7	1	13	5
node 11	10	9	15	3	13	1	7	8	14	2	5	6	0	12	4
node 12	13	14	8	4	10	6	0	15	9	5	2	1	7	11	3
node 13	12	15	9	5	11	7	1	14	8	4	3	0	6	10	2
node 14	15	12	10	6	8	4	2	13	11	7	0	3	5	9	1
node 15	14	13	11	7	9	5	3	12	10	6	1	2	4	8	0

Fig. 2. Preferred lists of a four-cube system.

node from its preferred list, all of its tasks will be forced to execute locally.

The preferred list constructed above has the following advantages. First, since each node is the most preferred node of one and only one node in the "static" list, the probability of an underloaded node being selected by more than one overloaded node is very small. Second, the cost of task transfer is minimal, since a receiver node is selected, with a high probability, from the physical proximity of the source node. Moreover, the time overhead for selecting an underloaded node is negligibly small, because the time-consuming probing procedure used in most known methods [4], [5], [7] is not needed.

Since the size of preferred list or buddy set will affect the probability of a task missing its deadline, it must be chosen to ensure this probability is lower than the specified limit. However, a buddy set must not be too large because the larger the size of buddy set, the higher traffic overhead for state-change broadcasts will result. Thus, there is a tradeoff between the capability of meeting deadlines and the traffic overhead caused by state-change broadcasts. More on this will be discussed in Section V.

IV. Models for the Proposed LS Method

An embedded Markov chain is used to model the performance of the proposed LS method. We begin with the development of an exact model from which an approximate solution and an approximate model, called the *upper bound model*, will be derived. The exact solution will be shown to be 1) always upper bounded by the solution to the upper bound model, called the *upper bound solution*, and 2) very close to the approximate solution. Note that an embedded Markov chain is commonly used to analyze arbitrary task arrivals. Since (average) QL is used to measure workloads, without loss of generality, one can assume (average) task execution time to be one unit of time. Let k_t and α_{k_t} be the number of task

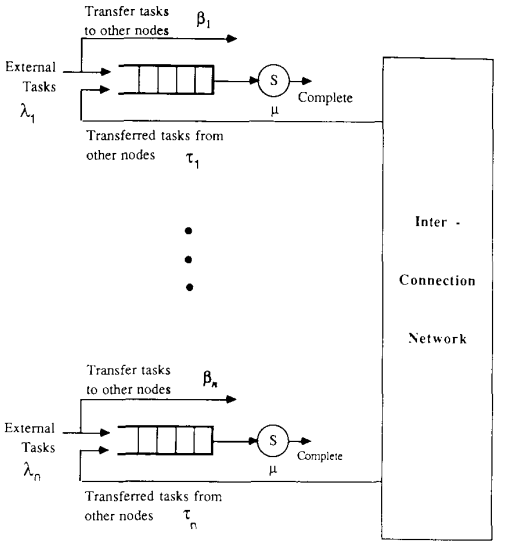


Fig. 3. System model.

arrivals and the probability of having k_t arrivals during the interval $[t, t+1)$, respectively. For example, when the interarrival time of external tasks is exponentially distributed with rate λ, α_{k_t} can be calculated as shown in [14] by

$$\alpha_{k_t} = \frac{\lambda^{k_t}}{k_t!} e^{-\lambda}. \tag{4.1}$$

Let x_t and x_{t+1} denote the QL at time t and $t+1$, respectively. Then,

$$x_{t+1} = \begin{cases} k_t & \text{if } x_t = 0 \text{ and } k_t \le \mathrm{TH}_v \\ x_t + k_t - 1 & \text{if } x_t > 0 \text{ and } x_t + k_t \le \mathrm{TH}_v + 1 \\ \mathrm{TH}_v & \text{if } x_t + k_t > \mathrm{TH}_v. \end{cases} \tag{4.2}$$

The above relation represents the case of ideal LS, since overloaded nodes are assumed to always find underloaded nodes to transfer their surplus tasks to.

Using (4.2), one can derive the probability distribution of QL. Two modifications must be made to include the effects of transferring and accepting tasks among the nodes in a buddy set. The first modification is to adjust the task arrival rate to include transferred-in tasks when a node is in U state. As shown in Fig. 3, the total arrival rate becomes $\omega = \lambda + \tau$, where λ and τ are the arrival rates of external and transferred-in tasks, respectively. A node's state transition probability depends on ω when the node is in U state, and thus, α's must be recalculated accordingly. Let α^*'s represent the transition probability corresponding to ω, whereas α's represent that corresponding to λ only. The second modification is made to the maximum QL. Since a node will always transfer tasks to other nodes when QL $>$ TH_v, the QL of a node with ideal LS is bounded by TH_v.

To illustrate these modifications, consider the threshold pattern "1 2 3" (i.e., $\mathrm{TH}_u = 1$, $\mathrm{TH}_f = 2$, $\mathrm{TH}_v = 3$) as an example, and let $q_i = P(\mathrm{QL} = i)$, $\forall i$. Then,

$$q_0 = \alpha_0^* q_0 + \alpha_0^* q_1$$

$$q_1 = \alpha_1^* q_0 + \alpha_1^* q_1 + \alpha_0 q_2$$

$$q_2 = \alpha_2^* q_0 + \alpha_2^* q_1 + \alpha_1 q_2 + \alpha_0 q_3$$

$$q_3 = (1 - \alpha_0^* - \alpha_1^* - \alpha_2^*) q_0 + (1 - \alpha_0^* - \alpha_1^* - \alpha_2^*) q_1 + (1 - \alpha_0 - \alpha_1) q_2 + (1 - \alpha_0) q_3$$

$$q_k = 0 \quad \text{for all } k > 3. \tag{4.3}$$

Note that the assumption that a task takes one unit of time to complete is used in the above equation.

As mentioned earlier, (4.3) represents ideal LS, i.e., overloaded nodes can always locate underloaded nodes to which their surplus tasks are transferred. In reality, however, an overloaded node may not always be able to find an underloaded node from its buddy set. An embedded Markov chain is developed below to handle this realistic case. In our LS method, the tasks in a node will be transferred to other nodes if its QL exceeds $\mathrm{TH}_v + 1$ (TH_v) upon (before) completion of a task. A node can accept tasks from other nodes only when QL $<$ TH_f. To transfer surplus tasks, the sharing capacity of each buddy set must be greater than or equal to the total number of surplus tasks in that buddy set. If this condition does not hold, an overloaded node's QL could grow larger than TH_v. To calculate the probability of a node's QL growing larger than TH_v, the following parameters are introduced. Let ϵ_i (θ_i) be the probability of having exactly (at least) i nodes available to share the surplus tasks within a buddy set. So, $\theta_i = 1 - \sum_{k=0}^{i-1} \epsilon_k$ for $1 \le i \le n$, and $\theta_i = \epsilon_i = 0$ for $i > n$, where n is the size of buddy set.

Assuming $x_t > 0$ for the previous example with threshold pattern "1 2 3," the number of surplus tasks in a node is $k_{ov} \equiv x_t + k_t - \mathrm{TH}_v$. When $k_{ov} = 1$, $x_{t+1} = \mathrm{TH}_v$ and the node will not transfer any task. When $k_{ov} = 2$, $x_{t+1} = \mathrm{TH}_v$ if there is at lease one node available for LS in its buddy set, and $x_{t+1} = \mathrm{TH}_v + 1$ if none of the nodes in the buddy set are available for LS. Similarly, when $k_{ov} = l > 2$, $x_{t+1} = \mathrm{TH}_v$ if there are at least $l-1$ nodes available in its buddy set, or $x_{t+1} = \mathrm{TH}_v + 1$ if there are exactly $l-2$ nodes available, or, in general, $x_{t+1} = \mathrm{TH}_v + j$ when there are exactly $l - (j+1)$ nodes available. Then, the state transition relation can be rewritten as

$$x_{t+1} = \begin{cases} k_t & \text{if } x_t = 0 \text{ and } k_t \le \mathrm{TH}_v \\ \mathrm{TH}_v \text{ with prob. } \theta_{k_t - \mathrm{TH}_v}, \text{ or } (\mathrm{TH}_v + 1) \text{ with prob.} & \\ \quad \epsilon_{k_t - \mathrm{TH}_v - 1}, \cdots, \text{ or } k_t \text{ with prob. } \epsilon_0 & \text{if } x_t = 0 \text{ and } k_t > \mathrm{TH}_v \\ x_t + k_t - 1 & \text{if } x_t > 0 \text{ and } x_t + k_t - 1 \le \mathrm{TH}_v \\ \mathrm{TH}_v \text{ with prob. } \theta_{x_t + k_t - \mathrm{TH}_v - 1}, \text{ or } (\mathrm{TH}_v + 1) \text{ with prob.} & \\ \quad \epsilon_{x_t + k_t - \mathrm{TH}_v - 2}, \cdots, \text{ or } (x_t + k_t - 1) \text{ with prob. } \epsilon_0 & \text{if } x_t > 0 \text{ and } x_t + k_t - 1 > \mathrm{TH}_v. \end{cases} \tag{4.4}$$

From the above relation, q_k's can be derived. For example, when the threshold pattern "1 2 3" is used, one can derive

$$q_3 = \left(\alpha_3^* + \sum_{i=1}^{\infty} \theta_i \alpha_{i+3}^*\right)(q_0 + q_1) + \sum_{i=2}^{4} \left(\alpha_{4-i} + \sum_{j=1}^{\infty} \theta_j \alpha_{j+4-i}\right) q_i + \sum_{i=5}^{\infty} \left(\sum_{j=i-4}^{\infty} \theta_j \alpha_{j-i+4}\right) q_i$$

available for LS,[4] the exact probability of an overloaded node being unable to transfer a task is always less than that derived from this model. The beauty of this model is the extreme simplicity in describing the state transition relation, as compared to the exact model. If $k_{ov} = 1$, then $x_{t+1} = \mathrm{TH}_v$ and no task will be transferred. If $k_{ov} = 2$, then $x_{t+1} = \mathrm{TH}_v$ with probability $\theta_1 = 1 - \epsilon_0$, or $x_{t+1} = \mathrm{TH}_v + 1$ with probability ϵ_0. When $k_{ov} = l$, $x_{t+1} = \mathrm{TH}_v + l - 2$ with probability θ_1, or $x_{t+1} = \mathrm{TH}_v + l - 1$ with probability ϵ_0. Summarizing the above leads to

$$x_{t+1} = \begin{cases} k_t & \text{if } x_t = 0 \text{ and } k_t \le \mathrm{TH}_v \\ (k_t - 1) \text{ with prob. } \theta_1, \text{ or } k_t \text{ with prob. } \epsilon_0 & \text{if } x_t = 0 \text{ and } k_t > \mathrm{TH}_v \\ x_t + k_t - 1 & \text{if } x_t > 0 \text{ and } x_t + k_t - 1 \le \mathrm{TH}_v \\ (x_t + k_t - 2) \text{ with prob. } \theta_1, \text{ or } (x_t + k_t - 1) \text{ with prob. } \epsilon_0 & \text{if } x_t > 0 \text{ and } x_t + k_t - 1 > \mathrm{TH}_v. \end{cases} \tag{4.6}$$

$$q_k = \sum_{i=0}^{n} \epsilon_i \alpha_{i+4}^* (q_0 + q_1) + \sum_{i=2}^{k+1} \left(\sum_{j=0}^{\infty} \epsilon_j \alpha_{j+k-i+1}\right) q_i + \sum_{i=k+2}^{\infty} \left(\sum_{j=0}^{\infty} \epsilon_{j+i-k-1} \alpha_j\right) q_i \qquad \text{for } k = 4, \cdots \infty. \tag{4.5}$$

Note that the q_k's for $k < \mathrm{TH}_v = 3$ are the same as shown in (4.3), and q_k's for other threshold patterns can be derived similarly.

Although the above equations can be used to calculate the distribution of QL, ϵ_k's and θ_k's are in practice too complex to compute. For example, ϵ_k is the probability of having k nodes available for LS in an n-node buddy set, the calculation of which requires us to consider $n!/(n-k)!k!$ different possibilities. The total number of possibilities that need to be considered for the calculation of ϵ_k for $k = 1, \cdots, n$ is 2^n. Our analysis shows this number to be over 1000 patterns when each buddy set contains 10–15 nodes. Furthermore, each of these patterns needs to be analyzed separately, since the probability of a node being in U state depends on the state of other nodes in the buddy set. Thus, it is extremely tedious to compute these parameters. To alleviate this difficulty, we develop the upper bound model and an approximate solution to (4.4). The former is used to derive ϵ_k's and a rough idea on the performance of our LS method, while the latter is used to obtain approximate q_k's from (4.4) using the parameters derived from the upper bound model.

A. Upper Bound Model and Solution

1) Upper Bound Model: This model is derived under the assumption that every node can always transfer only one surplus task to another node and the rest of its surplus tasks are forced to queue at that node. Since on the average 50 percent of the computation capacity in each buddy set has to be

The distribution of QL can now be derived from this equation as follows:

$$\begin{aligned} q_0 &= \alpha_0^* q_0 + \alpha_0^* q_1 \\ q_1 &= \alpha_1^* q_0 + \alpha_1^* q_1 + \alpha_0 q_2 \\ q_2 &= \alpha_2^* q_0 + \alpha_2^* q_1 + \alpha_1 q_2 + \alpha_0 q_3 \\ q_3 &= [\alpha_3^* + (1-\epsilon)\alpha_4^*] q_0 + [\alpha_3^* + (1-\epsilon)\alpha_4^*] q_1 \\ &\quad + [\alpha_2 + (1-\epsilon)\alpha_3] q_2 + [\alpha_1 + (1-\epsilon)\alpha_2] q_3 \\ &\quad + [\alpha_0 + (1-\epsilon)\alpha_1] q_4 + (1-\epsilon)\alpha_0 q_5 \\ q_4 &= [\epsilon\alpha_4^* + (1-\epsilon)\alpha_5^*](q_0 + q_1) + [\epsilon\alpha_3 + (1-\epsilon)\alpha_4] q_2 \\ &\quad + [\epsilon\alpha_2 + (1-\epsilon)\alpha_3] q_3 + [\epsilon\alpha_1 + (1-\epsilon)\alpha_2] q_4 \\ &\quad + [\epsilon\alpha_0 + (1-\epsilon)\alpha_1] q_5 + (1-\epsilon)\alpha_0 q_6 \\ q_k &= [\epsilon\alpha_k^* + (1-\epsilon)\alpha_{k+1}^*](q_0 + q_1) \\ &\quad + \sum_{j=2}^{k+1} [\epsilon\alpha_{k-j+1} + (1-\epsilon)\alpha_{k-j+2}] q_j \\ &\quad + (1-\epsilon)\alpha_0 q_{k+2} \qquad \text{for all } k \ge 4, \end{aligned} \tag{4.7}$$

where $\epsilon = \epsilon_0$. Equation (4.7) can be rewritten in vector form: $Q = AQ$, where $Q = [q_0, \cdots, q_n]^T$, A is an $n \times n$ coefficient matrix, and n is the size of buddy set. Using (4.7) and $\sum_{i=0}^{n} q_i = 1$, one can solve for Q, the upper bound solution.

The upper bound solution bounds the exact solution for the following reason. The only difference between the exact model (4.4) and the upper bound model (4.6) is that transitions to queue lengths TH_v, $\mathrm{TH}_v + 1, \cdots, (x_t + k_t - 2)$ in (4.4) are combined into a single transition to QL $= x_t + k_t - 2$ in

[4] Otherwise, load sharing is usually infeasible, and thus, should not be considered.

(4.6). Since $x_t + k_t - 2 \geq \mathrm{TH}_v$, the transition to a QL $> \mathrm{TH}_v$ is exaggerated in (4.6). Thus, the solution to the exact model will be bounded by the upper bound solution when $k > \mathrm{TH}_v$. Note that the upper bound model is identical to the exact model when $k \leq \mathrm{TH}_v$.

2) Solving the Upper Bound Model: The upper bound model is analyzed first to get a rough idea of the performance of our LS method. ω's and ϵ must be known before solving the upper bound model for q_k's. On the other hand, these parameters depend on q_k's, and thus, the model cannot be solved for q_k's without knowing ω's and ϵ. A two-step approximation approach is taken to handle the difficulty associated with this recursion problem. In the first step, the model is solved for τ and q_k's with $\epsilon = 0$. The resulting q_k's are still an upper bound for the exact solution. The second step is to compute ϵ based on the q_k's obtained in the first step.

By setting $\epsilon := 0$, q_k's for $k \geq 3$ in the upper bound model become

$$q_3 = (\alpha_3^* + \alpha_4^*)(q_0 + q_1) + (\alpha_2 + \alpha_3)q_2 + (\alpha_1 + \alpha_2)q_3 + (\alpha_0 + \alpha_1)q_4 + \alpha_0 q_5$$

$$q_4 = \alpha_5^*(q_0 + q_1) + \alpha_4 q_2 + \alpha_3 q_3 + \alpha_2 q_4 + \alpha_1 q_5 + \alpha_0 q_6$$

$$q_k = \alpha_{k+1}^*(q_0 + q_1) + \sum_{j=2}^{k+1} \alpha_{k-j+2} q_j + \alpha_0 q_{k+2} \qquad \text{for all } k > 4. \tag{4.8}$$

The above equation can be solved by using an iterative method. Initially, τ is set to 0. One can compute q_k's and then τ from

$$\beta \equiv \left[\sum_{k=4}^{\infty} (k-3)\alpha_k^*\right](q_0 + q_1) + \sum_{i=2}^{4}\left[\sum_{k=5-i}^{\infty} (k-4+i)\alpha_k\right] q_i + \sum_{i=5}^{\infty}\left[\sum_{k=0}^{\infty} k\alpha_k\right] q_i. \tag{4.9}$$

Note that β is the rate of task transfer out of a node. If all nodes' external task arrival rates are identical, then $\tau = \beta$. Otherwise, τ must be calculated by (4.11). After calculating τ, ω is obtained by adding λ to τ, and then q_k's are recalculated with the new ω, which will, in turn, change τ. This procedure will repeat until q_k's and τ converge to fixed values. (The convergence will be proved later in Theorem 1.)

Lemma 1: $dq_k/d\omega$ satisfies the following properties:

1) $\dfrac{dq_0}{d\omega} < 0$

2) $\displaystyle\sum_{k=0}^{\infty} \frac{dq_k}{d\omega} = 0$

3) $\left|\dfrac{dq_k}{d\omega}\right| < 1, \qquad \forall k.$

Proof: Since the probability of a system being idle (q_0) will decrease as the task arrival rate increases, the first property holds. The second property holds because the sum of all q_k's is equal to 1, and thus, the sum of the variations of all q_k's must be equal to 0. The last property can be proved by contradiction. Suppose $|dq_k/d\omega| \geq 1$. Then q_k may become negative or greater than 1 if the variation of ω exceeds 1, a possible event when a U-state node is surrounded by more than one V-state node. However, q_k can be neither negative nor greater than 1. Contradiction. □

Lemma 2: $0 \leq d\tau/d\omega < 1$.

Proof:

$$\begin{aligned}\frac{d\tau}{d\omega} &= \left[\sum_{k=4}^{\infty}(k-3)\frac{d\alpha_k^*}{d\omega}\right](q_0+q_1) + \left[\sum_{k=4}^{\infty}(k-3)\alpha_k^*\right]\left(\frac{dq_0}{d\omega}+\frac{dq_1}{d\omega}\right) \\ &\quad + \sum_{i=2}^{4}\left[\sum_{k=5-i}^{\infty}(k-4+i)\alpha_k\right]\frac{dq_i}{d\omega} + \sum_{i=5}^{\infty}\left[\sum_{k=0}^{\infty}k\alpha_k\right]\sum_{d\omega}^{dq_i} = (1-\alpha_0^*-\alpha_1^*-\alpha_2^*)(q_0+q_1) \\ &\quad + \left[\sum_{k=4}^{\infty}(k-3)\alpha_k^*\right]\left(\frac{dq_0}{d\omega}+\frac{dq_1}{d\omega}\right) + \sum_{i=2}^{4}\left[\sum_{k=5-i}^{\infty}(k-4+i)\alpha_k\right]\frac{dq_i}{d\omega} + \sum_{i=5}^{\infty}\left[\sum_{k=0}^{\infty}k\alpha_k\right].\end{aligned}$$

According to the definitions of α_k and α_k^*, each summation in the above equation is equal to, or less than, the average task arrival rate which is less than 1 in a stable system. (Recall that each node's service rate is assumed to be unity.) By the second and third properties of Lemma 1, the sum of the last three terms will be less than one. Furthermore, the first term will be much less than one, because the first three α^*'s usually dominate the determination of transition probabilities. Thus, the lemma follows. □

Theorem 1: q_k's and τ derived from the above iterative method converge to fixed values in a finite number of steps.

Proof: Let $d\omega^{(i)}$ and $d\tau^{(i)}$ be the variations of ω and τ at the ith iteration, respectively. These parameters at the $(i+1)$th iteration are related to those at the ith iteration by

$$d\tau^{(i+1)} = \frac{d\tau}{d\omega}\, d\omega^{(i)}$$

$$d\omega^{(i+1)} = \tau^{(i+1)} - \tau^{(i)} = d\tau^{(i+1)}.$$

Since $|d\tau/d\omega| < 1$ by Lemma 4, $d\tau$ at the $(i+1)$th iteration will be smaller than $d\omega$ at the ith iteration. Since the variation of ω at the $(i+1)$th iteration is equal to that of τ at the $(i+1)$th iteration, we get $d\tau^{(i+1)} < d\tau^{(i)}$ and $d\omega^{(i+1)} < d\omega^{(i)}$. Thus, the variation of τ will decrease to zero after a finite number of iterations, and so is ω. Substituting the convergent τ

and ω into (4.8), unique q_k's can be determined, i.e., q_k's also converge. □

Our numerical experiments show that τ and ω converge after only two to three iterations, indicating that the derivatives of τ and q_k with respect to ω are much smaller than 1.

3) Derivation of ϵ: The main difficulty in deriving ϵ_k's lies in the fact that the queue lengths in a buddy set depend on one another. Thus, the dependent LS environment is converted to an independent environment by using the Bayes theorem. To facilitate the description of our approach for an $(n+1)$-node buddy set, it is necessary to introduce the following variables.

- N_j^i: the jth preferred node of N_i.
- x_i: the N_i's queue length.
- x_{ij}: N_j^i's queue length.
- x_{jk}^i: the queue length of the kth preferred node of N_j^i.
- β_i: the rate of task transfer out of N_i.
- β_{ij}: the rate of task transfer out of N_j^i.
- γ_i: the rate of task transfer out of N_i given that N_i is not in sharing mode.
- γ_{ij}: the rate of task transfer out of N_j^i given that N_j^i is not in sharing mode.
- τ_i: the rate of task transfer into N_i.

It is easy to see that $\gamma_i > \beta_i$, since tasks are not actually transferred out of a node unless the node is in V state and β_i is the *average* transfer-out rate over the entire time period of interest.

Let N_0 be the node under consideration, then

$$\epsilon = P(x_{01} \geq \mathrm{TH}_f, \cdots, x_{0n} \geq \mathrm{TH}_f) \tag{4.10}$$

$$\begin{aligned}\tau_0 = {} & \beta_{01} P(x_0 \leq \mathrm{TH}_u) + \beta_{02} P(x_0 \leq \mathrm{TH}_u, x_{21}^0 \geq \mathrm{TH}_f) \\ & + \beta_{03} P(x_0 \leq \mathrm{TH}_u, x_{31}^0 \geq \mathrm{TH}_f, x_{32}^0 \geq \mathrm{TH}_f) \\ & + \cdots + \beta_{0n} P(x_0 \leq \mathrm{TH}_u, x_{n1}^0 \geq \mathrm{TH}_f, x_{n2}^0 \geq \mathrm{TH}_f, \\ & \cdots, x_{nn}^0 \geq \mathrm{TH}_f).\end{aligned} \tag{4.11}$$

Note that τ_0 derived from (4.11) would be identical to that derived from (4.9) if all nodes have the same external task arrival rate. Using (4.9) in such a case, for $j = 1, \cdots, n$

$$\gamma_{0j} = \sum_{i=2}^{4} \left[\sum_{k=5-i}^{\infty} (k-4+i)\alpha_k \right] \frac{q_i}{P^{\mathrm{nsh}}} + \sum_{i=5}^{\infty} \left[\sum_{k=0}^{\infty} k\alpha_k \right] \frac{q_i}{P^{\mathrm{nsh}}} \tag{4.12}$$

where $P^{\mathrm{nsh}} = 1 - q_0 - q_1$. Using the Bayes formula, the probability of both N_1^0 and N_2^0 not being in sharing mode can be calculated by

$$\begin{aligned}& P(x_{01} \geq \mathrm{TH}_f, x_{02} \geq \mathrm{TH}_f) \\ & \quad = P(x_{01} \geq \mathrm{TH}_f) P(x_{02} \geq \mathrm{TH}_f | x_{01} \geq \mathrm{TH}_f).\end{aligned} \tag{4.13}$$

Since the dependence between queue lengths is included in ω, its effect can be included by adjusting the rate of task transfer into N_2^0 given that N_1^0 is not in sharing mode. So, the conditional probability $P(x_{02} \geq \mathrm{TH}_f | x_{01} \geq \mathrm{TH}_f)$ can be equated to $P(x_{02} \geq \mathrm{TH}_f)$, while the ω of N_2^0 must be adjusted to reflect the effect of N_1^0's unavailability. As shown in (4.12), such an adjustment will increase the rate of task transfer out of N_1^0 given that it is not in sharing mode, which will, in turn, increase N_2^0's task transfer-in rate. Moreover, N_0 will select N_2^0 as the most preferred node given that N_1^0 is not in sharing mode, and thus, the task transfer-in rate of N_2^0 should be recalculated. For notational convenience, let N_2 represent the node N_2^0 under consideration, then

$$\begin{aligned}\tau_2 = {} & \beta_{21} P(x_2 \leq \mathrm{TH}_u) + \beta_{22} P(x_2 \leq \mathrm{TH}_u) \\ & + \beta_{23} P(x_2 \leq \mathrm{TH}_u, x_{31}^2 \geq \mathrm{TH}_f, x_{32}^2 \geq \mathrm{TH}_f) \\ & + \cdots + \beta_{2n} P(x_2 \leq \mathrm{TH}_u, x_{n1}^2 \geq \mathrm{TH}_f, \\ & x_{n2}^2 \geq \mathrm{TH}_f, \cdots, x_{nn}^2 \geq \mathrm{TH}_f).\end{aligned} \tag{4.14}$$

The first two terms of (4.14) represent the transferred-in tasks from the N_2's most preferred node and $N_0 (= N_2^2)$. Since N_1^0 is unavailable, N_2 becomes the most preferred node of both N_1^2 and N_0. Clearly, N_2's ω will be larger than those of N_0 and N_1^0. Hence, it is likely to switch to no-sharing mode when N_1^0 is in no-sharing mode. Similarly, the probability of all N_1^0, N_2^0, and N_3^0 not being in sharing mode can be calculated as

$$\begin{aligned}& P(x_{01} \geq \mathrm{TH}_f, x_{02} \geq \mathrm{TH}_f, x_{03} \geq \mathrm{TH}_f) \\ & \quad = P(x_{01} \geq \mathrm{TH}_f, x_{02} \geq \mathrm{TH}_f) \\ & \qquad \times P(x_{03} \geq \mathrm{TH}_f | x_{01} \geq \mathrm{TH}_f, x_{02} \geq \mathrm{TH}_f) \\ & \quad = P(x_{03} \geq \mathrm{TH}_f) P(x_{01} \geq \mathrm{TH}_f, x_{02} \geq \mathrm{TH}_f) \\ & \quad = P(x_{03} \geq \mathrm{TH}_f) P(x_{01} \geq \mathrm{TH}_f) P(x_{02} \geq \mathrm{TH}_f).\end{aligned}$$

The ω of N_2^0 and N_3^0 must be recalculated as described above. The correctness of (4.11) can be verified as follows. When all nodes in the system have the same distribution of QL and the same β, (4.11) can be simplified as

$$\begin{aligned}\tau = {} & \beta_{01} P(x_0 \leq \mathrm{TH}_u) + \beta_{02} P(x_0 \geq \mathrm{TH}_u) P(x_{21}^0 \geq \mathrm{TH}_f) \\ & + \cdots + \beta_{0n} P(x_0 \geq \mathrm{TH}_u) P(x_{n1}^0 \geq \mathrm{TH}_f) \\ & \cdot P(x_{n2}^0 \geq \mathrm{TH}_f) \cdots P(x_{nn}^0 \geq \mathrm{TH}_f) \\ = {} & \beta P(x_0 \leq \mathrm{TH}_u)[1 + P(x_{21}^0 \geq \mathrm{TH}_f) \\ & + P(x_{31}^0 \geq \mathrm{TH}_f)^2 + \cdots + P(x_{n1}^0 \geq \mathrm{TH}_f)^n] \\ = {} & \beta P(x_0 \leq \mathrm{TH}_u) \frac{1 - P(x_{21} \geq \mathrm{TH}_f)^{n+1}}{1 - P(x_{21} \geq \mathrm{TH}_f)} \\ \simeq {} & \beta P(x_0 \leq \mathrm{TH}_u) \frac{1}{1 - P(x_{21} \geq \mathrm{TH}_f)} \\ = {} & \beta P(x_0 \leq \mathrm{TH}_u) \frac{1}{P(x_{21} \leq \mathrm{TH}_u)} \simeq \beta.\end{aligned}$$

Consider a four-cube system as an example, in which, without loss of generality, N_0 can be viewed as the center node for the derivation of ϵ. From Fig. 2, N_0's preferred list is $N_1 N_2 N_4 N_8 N_6 N_{10} N_{12} N_3 N_5 N_9 N_{14} N_{13} N_{11} N_7$. Since the nodes near the end of the list are unlikely to be selected for LS, the adjusted task transfer-in rate of these four nodes can be

approximated by adding $\beta_0 P(x_0 \leq \mathrm{TH}_u)$ to the τ of these nodes. Since increasing task transfer-in rate will change QL, the q_k's of these nodes need to be recalculated for

$$\begin{aligned} &P(x_1 \geq \mathrm{TH}_f,\ x_2 \geq \mathrm{TH}_f,\ x_4 \geq \mathrm{TH}_f,\ x_8 \geq \mathrm{TH}_f) \\ &\quad = P(x_1 \geq \mathrm{TH}_f) P(x_2 \geq \mathrm{TH}_f \times P(x_4 \geq \mathrm{TH}_f) P(x_8 \geq \mathrm{TH}_f) \\ &\quad \equiv \epsilon^{(4)}. \end{aligned} \tag{4.15}$$

Note that the states of these four nodes are different from that of N_0, because their task transfer-in rates are higher than that of N_0. Thus, these nodes are more likely to be in V-state than N_0. Similarly, one can calculate the adjusted task transfer-in rates for N_5–N_{10}. As shown in Fig. 2, each of these nodes has two of the previous four nodes in its entry-1. Furthermore, as the number of V-state nodes increases, tasks will be transferred to a less preferred node of N_0. The adjusted task transfer-in rates of these nodes are

$$\begin{aligned} \tau_6 &= \tau + \gamma_2 P(x_6 \leq \mathrm{TH}_u) P(x_7 \geq \mathrm{TH}_f) \\ &\quad + \gamma_4 P(x_6 \leq \mathrm{TH}_u) P(x_7 \geq \mathrm{TH}_f) + \beta_0 P(x_6 \leq \mathrm{TH}_u) \\ \tau_{10} &= \tau + \gamma_8 P(x_{10} \leq \mathrm{TH}_u) P(x_{11} \geq \mathrm{TH}_f) + \beta_0 P(x_{10} \leq \mathrm{TH}_u) \\ &\quad + \gamma_2 P(x_{10} \leq \mathrm{TH}_u) P(x_{11} \geq \mathrm{TH}_f) P(x_{14} \geq \mathrm{TH}_f) \\ &\quad + \gamma_6 P(x_{10} \leq \mathrm{TH}_u) P(x_{11} \geq \mathrm{TH}_f) \\ &\quad \cdot P(x_{12} \geq \mathrm{TH}_f) P(x_{14} \geq \mathrm{TH}_f) \\ \tau_{12} &= \tau + \gamma_8 P(x_{12} \leq \mathrm{TH}_u) P(x_{13} \geq \mathrm{TH}_f) P(x_{14} \geq \mathrm{TH}_f) \\ &\quad + \beta_0 P(x_{12} \leq \mathrm{TH}_u) + (\gamma_4 + \gamma_6 + \gamma_{10}) P(x_{12} \leq \mathrm{TH}_u) \\ &\quad \cdot P(x_{13} \geq \mathrm{TH}_f) P(x_{14} \geq \mathrm{TH}_f) \\ \tau_3 &= \tau + \gamma_1 P(x_3 \leq \mathrm{TH}_u) + \gamma_2 P(x_3 \leq \mathrm{TH}_u) + \beta_0 P(x_3 \leq \mathrm{TH}_u) \\ &\quad + (\gamma_5 + \gamma_9) P(x_3 \geq \mathrm{TH}_u) P(x_7 \geq \mathrm{TH}_f) P(x_{11} \geq \mathrm{TH}_f) \\ &\quad + (\gamma_6 + \gamma_{10}) P(x_3 \leq \mathrm{TH}_u) P(x_7 \geq \mathrm{TH}_f) \\ &\quad \cdot P(x_{11} \geq \mathrm{TH}_f) P(x_{15} \geq \mathrm{TH}_f) \\ \tau_5 &= \tau + \gamma_4 P(x_5 \leq \mathrm{TH}_u) + \gamma_1 P(x_5 \leq \mathrm{TH}_u) \\ &\quad \cdot P(x_7 \geq \mathrm{TH}_f) + \beta_0 P(x_5 \leq \mathrm{TH}_u) \\ &\quad + \gamma_3 P(x_5 \leq \mathrm{TH}_u) P(x_7 \geq \mathrm{TH}_f) P(x_{13} \geq \mathrm{TH}_f) \\ &\quad + (\gamma_6 + \gamma_{12}) P(x_5 \leq \mathrm{TH}_u) P(x_7 \geq \mathrm{TH}_f) \\ &\quad \cdot P(x_9 \geq \mathrm{TH}_f) P(x_{13} \geq \mathrm{TH}_f) P(x_{15} \geq \mathrm{TH}_f) \\ \tau_9 &= \tau + \gamma_8 P(x_9 \leq \mathrm{TH}_u) + \gamma_1 P(x_9 \leq \mathrm{TH}_u) \\ &\quad \cdot P(x_{11} \geq \mathrm{TH}_f) P(x_{13} \geq \mathrm{TH}_f) + \beta_0 P(x_9 \leq \mathrm{TH}_u) \\ &\quad + (\gamma_3 + \gamma_5 + \gamma_{10} + \gamma_{12}) P(x_9 \leq \mathrm{TH}_u) \\ &\quad \cdot P(x_{11} \geq \mathrm{TH}_f) P(x_{13} \geq \mathrm{TH}_f) \\ &\quad \cdot P(x_{15} \geq \mathrm{TH}_f). \end{aligned}$$

Once the task transfer-in rates of entry-2 nodes are adjusted, the probability of having all entry-1 and entry-2 nodes in no-sharing mode can be calculated as

$$\begin{aligned} &P(x_1 \geq \mathrm{TH}_f,\ x_2 \geq \mathrm{TH}_f,\ \cdots,\ x_9 \geq \mathrm{TH}_f) \\ &\quad = P(x_1 \geq \mathrm{TH}_f) P(x_2 \geq \mathrm{TH}_f) \cdots P(x_9 \geq \mathrm{TH}_f) \\ &\quad \equiv \epsilon^{(10)}. \end{aligned} \tag{4.16}$$

Similarly, one can calculate the probability of all other nodes in a four-cube system being unavailable as $\epsilon^{(15)} \equiv \epsilon$.

B. Approximate Solution

Although q_k's and $\epsilon (= \epsilon_0)$ can be derived from the upper bound model, it is still very tedious to calculate ϵ_k, $\forall k > 0$, because there are too many possibilities to consider and each of them is difficult to analyze due to the independence of LS among the nodes in a buddy set. Moreover, the upper bound model solution fails to include the effects of buddy set size and threshold patterns on the capability of meeting deadlines, while the simulation results in Section V did show significant differences when these parameters were changed. So, it is necessary to derive a solution which is simple but closer to the exact solution to (4.4) than the upper bound solution. Since there are $n!/(n-k)!k!$ possibilities in calculating ϵ_k in an n-node buddy set, these possibilities can be approximated with only one possibility in which a node is in no-sharing mode with the largest probability. This possibility occurs when all other nodes in the buddy set are in no-sharing mode.

Consider the N_0's preferred list in Fig. 2 again. The probabilities of N_2–N_9 being in no-sharing mode are different from one another due to the adjustment of task transfer-in rates given that more preferred nodes are in no-sharing mode. As the number of no-sharing nodes increases, the adjusted task transfer-in rate of the next preferred node increases. Eventually, N_7, the least preferred node of N_0, will receive the largest number of transfer-in tasks, thus moving it in no-sharing mode with the highest probability within N_0's buddy set. Let P^{sh} and P^{nsh} denote the probabilities of N_7 being in sharing and no-sharing mode, respectively. Then, ϵ_k's can be approximated by

$$\epsilon_k = \frac{n!}{(n-k)!k!} (P^{\mathrm{nsh}})^{n-k} (P^{\mathrm{sh}})^k. \tag{4.17}$$

Plugging the ϵ_k's derived from (4.17) into (4.5) and applying the iterative method discussed in the previous subsection, we can easily obtain an approximate solution. The calculated results are listed in Tables I and III in comparison with the results derived from the upper bound model and simulations (to be discussed in the next section).

Note that the ϵ_k's derived from (4.8) are essentially the same as those derived from (4.7) since both have the same queue state equations for QL $<$ TH_u which are dominant in the probability calculation.

V. Performance Analysis

The performance of the proposed LS method is evaluated with the upper bound model, the approximate solution, and simulation. The first two are used to derive the distribution of QL at each node and the probability of meeting deadlines, and

TABLE I
COMPARISON OF DISTRIBUTIONS OF QUEUE LENGTH DERIVED FROM THE UPPER BOUND MODEL, THE APPROXIMATE SOLUTION, AND SIMULATION WITH THRESHOLD PATTERN "1 2 3" AND A BUDDY SET OF SIZE 10

($\lambda = 0.5$) Model / Queue Length	Simulation	Approximation	Upper bound	no load sharing
0	0.5037	0.4987	0.4992	0.5000
1	0.3316	0.3317	0.3321	0.3244
2	0.1254	0.1289	0.1278	0.1226
3	0.0387	0.0406	0.0398	0.0377
4	3.13×10^{-7}	1.43×10^{-8}	0.0010	0.0109
5	$< 10^{-7}$	1.21×10^{-9}	0.0001	0.0031
6	$< 10^{-7}$	9.09×10^{-11}	1.3×10^{-5}	0.0009
7	$< 10^{-7}$	5.96×10^{-12}	1.4×10^{-6}	0.0003
8	$< 10^{-7}$	3.35×10^{-13}	1.33×10^{-7}	7.16×10^{-5}
9	$< 10^{-7}$	1.46×10^{-14}	1.28×10^{-8}	2.04×10^{-5}

($\lambda = 0.8$) Model / Queue Length	Simulation	Approximation	Upper bound	no load sharing
0	0.2213	0.2264	0.2185	0.2004
1	0.3317	0.3194	0.3136	0.2456
2	0.2656	0.2675	0.2651	0.1898
3	0.1810	0.1862	0.1853	0.1278
4	0.0002	0.0003	0.0135	0.0834
5	2.01×10^{-5}	5.94×10^{-5}	0.0032	0.0542
6	8.98×10^{-6}	8.99×10^{-6}	0.0007	0.0353
7	3.21×10^{-6}	1.18×10^{-6}	0.0001	0.0229
8	7.81×10^{-7}	1.36×10^{-7}	1.86×10^{-5}	0.0149
9	3.90×10^{-7}	1.41×10^{-8}	2.38×10^{-6}	0.0097

analyze the effects of buddy set size, the frequency of state change, and the average system sojourn time of each task. On the other hand, simulation is used to verify the analytic results.

A. Analytic Results

The proposed model can be applied to any arrival process, but the transition probability α_{k_t} must be given prior to the calculation of q_k's with (4.4) and (4.8). To demonstrate the main idea of our LS method, we present some numerical results for the case when both arrivals of external and transferred-in tasks follow exponential distributions. (Note, however, that our LS method and models are not restricted to exponential distributions.)

1) Distribution of Queue Length: The distributions of QL for two different external task arrival rates in a 16-node system are calculated with the upper bound model and the approximate solution, and compared to simulation results as well as to the case of no LS (Table I). The q_k's calculated with the upper bound solution and the approximate solution are very close to each other when $k \leq \mathrm{TH}_v$. This was expected because the two differ only when $k > \mathrm{TH}_v$. This fact also ensures the accuracy in calculating ϵ, since it was computed with the q_k's derived from the upper bound model and then used to derive approximate q_k's from (4.4). Moreover, the distribution of QL obtained via simulation is shown to be very close to the approximate solution for all k and is bounded by the upper bound solution when $k > \mathrm{TH}_v$. Since the approximate solution is always very close to the exact solution, we will use it in the following discussions unless stated otherwise.

2) Probability of Meeting Deadlines: A task is said to be *missed* if its system sojourn time[5] exceeds a given deadline. According to our queueing model, the completion time of a newly arriving task is equal to the current queue length plus one unit of time. Since the probability of QL $> \mathrm{TH}_v$ is quite small, one can choose TH_v to be one less than the given deadline such that the probability of missing deadlines, or simply called the *missing probability*, becomes the probability of encountering QL $> \mathrm{TH}_v$ at the time of a task arrival. Clearly, the missing probability depends on the given deadline and system load. However, by selecting a proper threshold pattern and a buddy set size, it is possible to minimize the missing probability. Figs. 4 and 5 are the plots of missing probabilities versus task deadlines for different threshold patterns.

Generally, the missing probability increases as system load gets heavier (Fig. 6) and/or the deadline gets shorter. By choosing an appropriate threshold pattern, e.g., "1 2 3" in Figs. 4–6, the missing probability can be reduced to a small value even when system load fluctuates (except when the system is overloaded, e.g., $\lambda \geq 0.9$). The analytic results also

[5] The system sojourn time of a task is composed of its execution time, queueing time, and task transfer time.

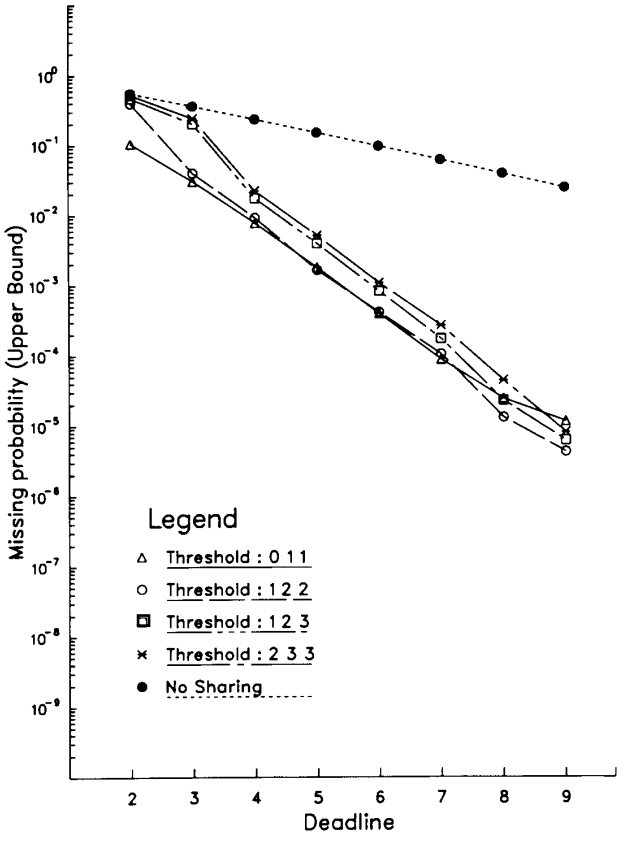


Fig. 4. Upper bound missing probabilities versus deadlines for different thresholds when $\lambda = 0.8$.

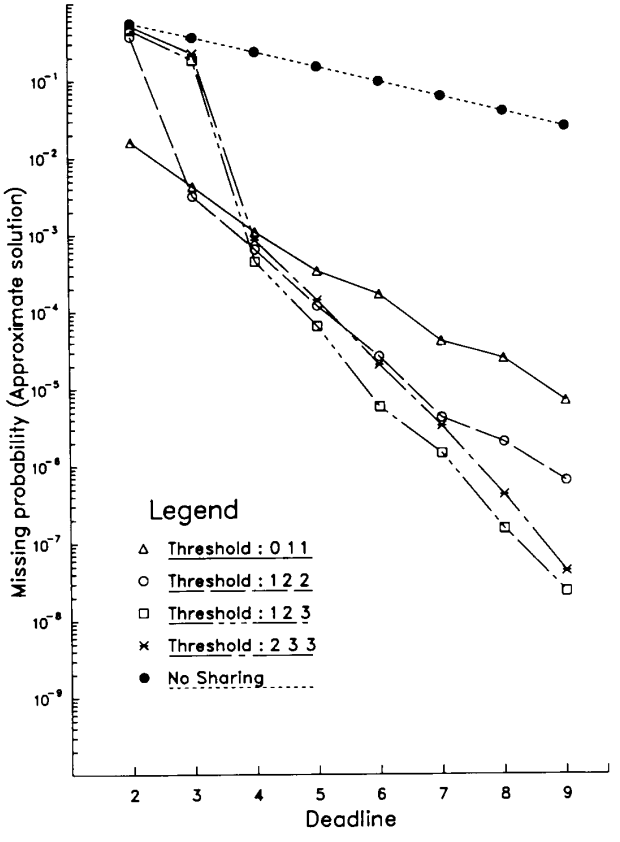


Fig. 5. Approximate missing probabilities versus deadlines for different thresholds when $\lambda = 0.8$.

show that the choice of a threshold pattern is sensitive to system load. For example, threshold pattern "0 1 1" results in a small missing probability when the system is underloaded, while resulting in a much higher missing probability as system load increases. Threshold pattern "1 2 3" is found to yield a reasonably small missing probability for a wide range of load density ($0 < \lambda < 0.8$). Fig. 4 shows an interesting result of the upper bound model: missing probabilities for different threshold patterns are quite close to each other, and thus, difficult to tell which pattern is better over the others. This is opposite to what has been shown by the approximate solution in Figs. 5 and 6. That is, the upper bound model exaggerates the probability of switching to a queue length greater than TH_v, and thus, the effect of threshold pattern becomes insignificant. Since threshold pattern "1 2 3" exhibits the best performance among the three patterns considered, the performance with this pattern is further compared to simulation results. As shown in Figs. 7 and 8, the missing probability obtained from the simulation is always upper bounded by those obtained from the upper bound model and is very close to the approximate solution.

3) Average System Sojourn Time Versus Missing Probability: The average system sojourn time can be obtained by dividing the sum of all tasks' system sojourn times by the total number of tasks processed. Mathematically, the average system sojourn time is equal to the expected task execution time, $\Sigma_{k=0}^{\infty}(k + 1)q_k$. The average system sojourn time is calculated for several different threshold patterns and buddy set sizes as presented in Table II. One interesting result is that the lower TH_u and TH_v, the smaller the average system sojourn time results, and that buddy set size has only minor effects on the average system sojourn time. This is in sharp contrast with the results reported in [4], where the average system sojourn time under the shortest queue policy was shown to be only slightly smaller than that under the threshold policy. In our LS method, the shortest queue (threshold) policy is equivalent to selecting $TH_u = 0$ and $TH_f = 1$ ($TH_u > 0$). As shown in Table II, the threshold pattern with $TH_u = 0$ and $TH_f = 1$ always results in a substantially smaller average system sojourn time than the pattern with $TH_u > 0$. This is the advantage resulting from our state-change broadcasts since the traffic overhead for collecting state information in the case of $TH_u = 0$ is essentially the same as the case of $TH_u > 0$. However, the traffic overhead associated with the shortest queue policy is higher than that of the threshold policy due to its required probing of other nodes [4], offsetting the potential gain to be made by transferring tasks to a node with the shortest queue. Consequently, our LS method outperforms other sender-initiated LS algorithms even when the average system time is used to measure their performance.

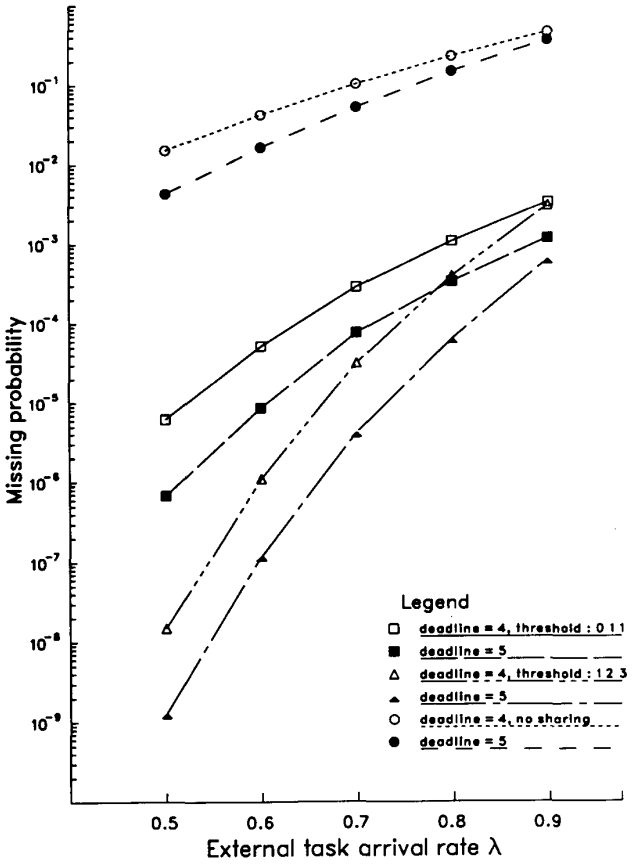


Fig. 6. Approximate missing probabilities versus load density for different thresholds and deadlines.

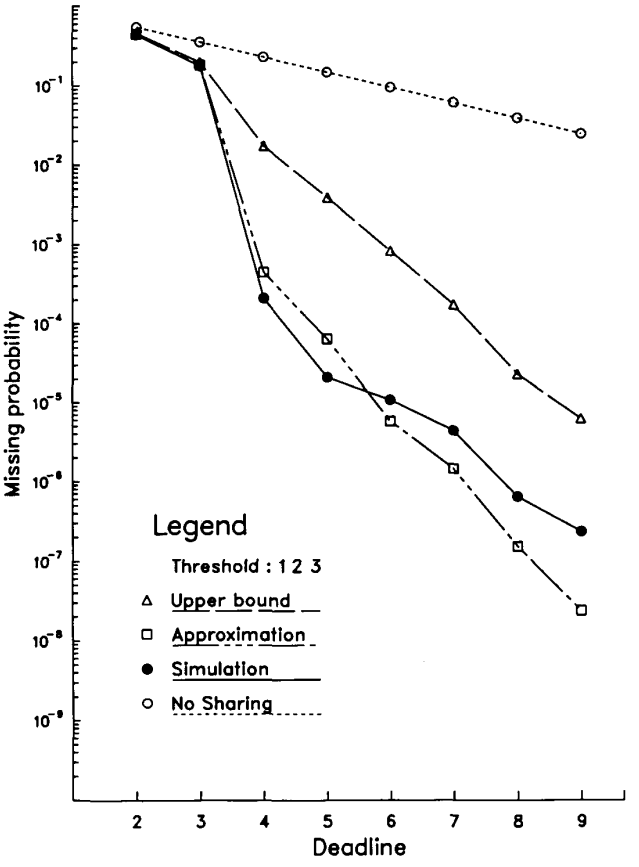


Fig. 7. Simulated, approximate, and upper bound missing probabilities versus deadlines when $\lambda = 0.8$.

Another important result is that a threshold pattern that results in a lower average system sojourn time does not always yield a lower missing probability. For example, consider the buddy sets of size 10 in Tables II and III. Pattern "0 1 2" results in a smaller average task system sojourn time than "1 2 2," but a larger missing probability than "1 2 2" when the deadline is greater than 2. Moreover, some thresholds may result in almost the same average task system sojourn time but yield quite different missing probabilities, e.g., "0 2 3" and "1 2 3" when the deadline is greater than 3 in Table III. Hence, those approaches based on minimizing the average task system sojourn time alone may not be applicable to the analysis of real-time systems.

4) System Utilization: The system utilization is defined as the ratio of external task arrival rate (λ) to the system service rate, which is unity in our LS model. (Thus, the system utilization is simply λ.) Since the missing probability depends on system workload (Fig. 8), we can solve (4.8) to derive λ as a function of q_k's and then the maximum system utilization can be obtained by equating $q_{\mathrm{TH}_{v+1}}$ to the specified missing probability. Some of the calculated results are plotted in Fig. 9. This is in sharp contrast to the common notion that real-time systems have to be designed to sacrifice utilization for a lower missing probability.

5) Buddy Set Size and Preferred List: The effect of changing buddy set size on the missing probability can best be explained by the approximate solution as shown in Fig. 10. Buddy set size affects the missing probability significantly when it grows from 4 to 10 and $\lambda > 0.7$, but its incremental effect becomes insignificant when buddy set size is greater than 10. Actually, there is little notable decrease in the missing probability when buddy set size grows beyond 15. Surprisingly, the missing probability for a four-node buddy set is about three orders of magnitude less than that without LS when the system is underloaded ($\lambda \leq 0.5$), and is about the same as those for buddy sets of size larger than 10 when the system is overloaded ($\lambda > 0.8$). So, buddy set size can be chosen to range from 10 to 15, regardless of the system size. The most interesting result is found to be that the missing probability in a large system (of 64 nodes in Table IV) is much smaller than that of a small system (of 16 nodes in Table III). For example, consider threshold "1 2 3" with a ten-node buddy set at $\lambda = 0.8$. The missing probability of a 64-node system is about 3, 4, and 20 times smaller than that of the 16-node system when the deadline is 4, 5, and 6, respectively. This significant improvement was found for all other threshold patterns, thus indicating that the larger the system size, the better the performance of the proposed LS method will result. (See the next paragraph for a reasoning about this.) Note that the traffic overhead for broadcasting state changes remains

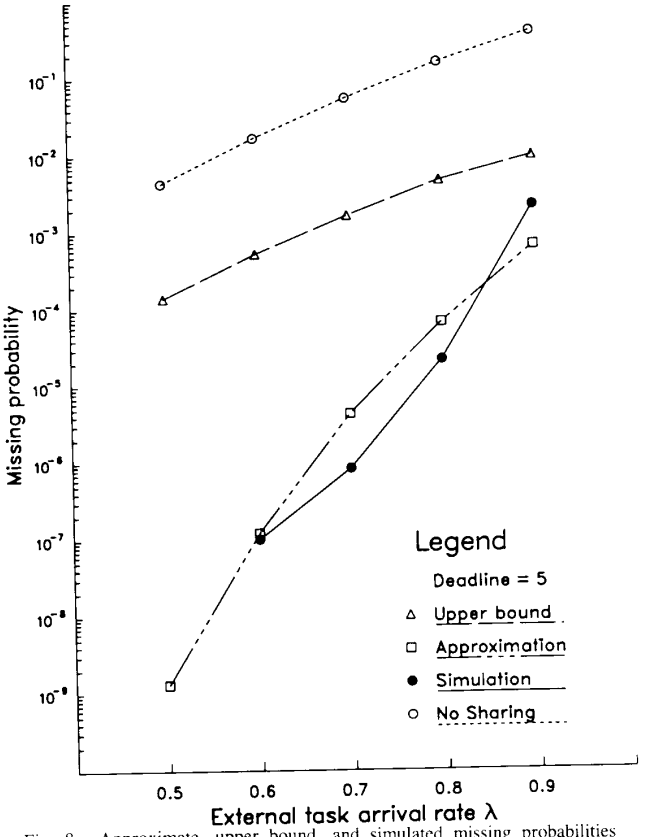


Fig. 8. Approximate, upper bound, and simulated missing probabilities versus load density with threshold pattern "1 2 3."

TABLE II
AVERAGE TASK SYSTEM TIME WHEN $\lambda = 0.5$ AND $\lambda = 0.8$

($\lambda = 0.5$) Threshold \ Buddy set	4	10	15
0 1 1	1.414	1.394	1.401
0 1 2	1.609	1.602	1.605
0 2 2	1.608	1.608	1.622
0 2 3	1.696	1.695	1.700
1 2 2	1.618	1.618	1.625
1 2 3	1.698	1.698	1.701
1 3 3	1.700	1.700	1.703
2 3 3	1.701	1.701	1.703
No sharing	1.750	1.750	1.750

($\lambda = 0.8$) Threshold \ Buddy set	4	10	15
0 1 1	1.918	1.692	1.651
0 1 2	2.236	2.065	1.033
0 2 2	2.120	2.075	2.071
0 2 3	2.426	2.382	2.380
1 2 2	2.123	2.109	2.110
1 2 3	2.423	2.407	2.405
1 3 3	2.427	2.422	2.421
2 3 3	2.475	2.472	2.473
No sharing	3.370	3.370	3.370

TABLE III
MISSING PROBABILITIES FOR VARIOUS THRESHOLD PATTERNS AND TASK DEADLINES WHEN $\lambda = 0.5$ AND $\lambda = 0.8$, AND BUDDY SET SIZE = 10

($\lambda = 0.5$)	Threshold / Deadline	0 1 1	0 1 2	1 2 2	1 2 3	2 3 3	No Sharing
simulation	2	0.0007	0.1296	0.1339	0.1642	0.1661	—
	3	4.81×10^{-5}	8.82×10^{-5}	1.73×10^{-5}	0.0388	0.0392	—
	4	3.10×10^{-6}	5.32×10^{-6}	1.45×10^{-7}	3.13×10^{-7}	6.59×10^{-7}	—
	5	2.85×10^{-7}	$< 10^{-7}$	$< 10^{-7}$	$< 10^{-7}$	$< 10^{-7}$	—
	6	$< 10^{-7}$	$< 10^{-7}$	$< 10^{-7}$	$< 10^{-7}$	$< 10^{-7}$	—
analytic (approximation)	2	0.0004	0.1370	0.1446	0.1681	0.1691	0.1756
	3	5.10×10^{-5}	8.41×10^{-5}	7.46×10^{-7}	0.0398	0.0406	0.0530
	4	6.25×10^{-6}	9.73×10^{-6}	7.11×10^{-8}	1.46×10^{-8}	1.87×10^{-9}	0.0152
	5	6.81×10^{-7}	9.51×10^{-7}	6.01×10^{-9}	1.31×10^{-9}	1.44×10^{-10}	0.0043
	6	7.61×10^{-8}	8.8×10^{-8}	9.11×10^{-11}	1.31×10^{-11}	1.01×10^{-11}	0.0012

($\lambda = 0.8$)	Threshold / Deadline	0 1 1	0 1 2	0 2 3	1 2 2	1 2 3	2 3 3	No Sharing
simulation	2	0.0745	0.3258	0.4414	0.3461	0.4468	0.4858	—
	3	0.0176	0.0305	0.1805	0.0019	0.1812	0.2007	—
	4	0.0045	0.0067	0.012	0.0003	0.0002	0.0002	—
	5	0.0014	0.0015	0.0003	8.37×10^{-5}	2.11×10^{-5}	2.28×10^{-5}	—
	6	0.0006	0.0004	0.0001	3.61×10^{-5}	1.11×10^{-5}	6.18×10^{-6}	—
analytic (approximation)	2	0.0161	0.3315	0.4342	0.3723	0.4541	0.5151	0.5539
	3	0.0043	0.0105	0.1772	0.0032	0.1866	0.2256	0.3641
	4	0.0011	0.0024	0.0013	0.0007	0.0004	0.0009	0.2363
	5	0.0004	0.0005	0.0003	0.0001	6.54×10^{-5}	0.0002	0.1528
	6	0.0002	0.0002	6.95×10^{-5}	2.64×10^{-5}	1.2×10^{-5}	2.58×10^{-5}	0.0986

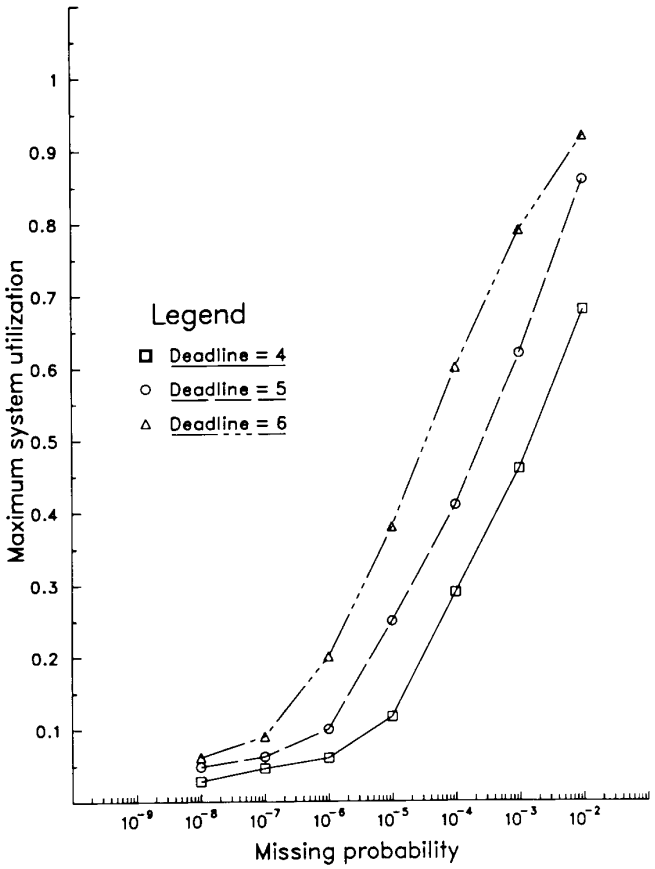


Fig. 9. Maximum system utilization versus missing probability for different deadlines.

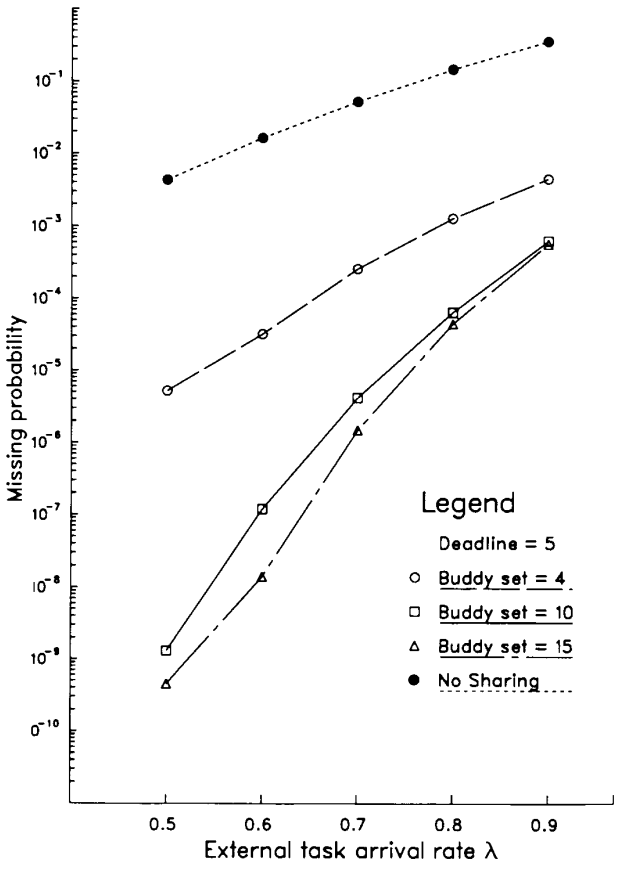


Fig. 10. Approximate missing probabilities versus load density with threshold "1 2 3" and different buddy set sizes.

unchanged and independent of system size because buddy set size is fixed (to 10–15). Furthermore, the incremental decrease in missing probability becomes insignificant when buddy set size is over 15 (Table IV).

Use of buddy sets and preferred lists in our LS method plays a major role in lowering the missing probability for a large system. As discussed in Section III, the buddy set of a node consists of those nodes in its physical proximity, and each node in the buddy set is selected according to the order of its preference. Moreover, preferred lists are constructed in such a way that each node is the ith ($i = 1, \cdots, n$) preferred node of only one other node and the preferred lists of the nodes in the same buddy set are completely different from each other. As a result, the surplus tasks within each *buddy set* will be evenly shared by all underloaded nodes in the *entire system*, rather than overloading a few underloaded nodes within the same buddy set. As the system size grows, the percentage of common nodes in the preferred lists of a buddy set gets smaller, and thus, the surplus tasks are more evenly distributed in the system, resulting in a better performance.

6) Frequency of State Change: In our LS method, each node needs to broadcast change of state to all the other nodes in its buddy set. Since a state change occurs when a node switches from U state to F state and vice versa, the probability of a state change becomes $P(x_{k_{t+1}} \leq \mathrm{TH}_u | x_{k_t} \geq \mathrm{TH}_f) + P(x_{k_{t+1}} \geq \mathrm{TH}_f | x_{k_t} \leq \mathrm{TH}_u)$. The computation results of Table V showed that the frequency of state change can be reduced to 10–15 percent of the total number of arrived tasks by setting $\mathrm{TH}_f - \mathrm{TH}_u = 2$. Note that this frequency becomes about 100 percent of the number of external task arrivals when $\mathrm{TH}_u = 0$ and $\mathrm{TH}_f = 1$. The resulting high frequency of state change should rule out this type of threshold pattern.

The traffic overhead for collecting state information in our LS method is determined by the frequency of state change and buddy set size, while it was determined by the number of task transfers and probing in [4]. Since the frequency of state change can be controlled by adjusting the difference between TH_u and TH_f, this frequency with threshold "1 3 3" and $\lambda > 0.7$ is found to be about the same as the percentage of external arrivals that are transferred out. Moreover, transferring one task may require us to probe 5–6 other nodes [4] and each probe generates two communication messages (one for request and the other for response) in sender-initiated methods, whereas each state-change broadcast in our LS method generates n messages, where n is the buddy set size. The traffic overhead for broadcasting state changes for threshold "1 3 3" and a ten-node buddy set is about the same as that in a sender-initiated approach. However, the time for selecting a destination node in our method is much smaller than that in any sender-initiated approach, because in our approach, a task

TABLE IV
MISSING PROBABILITIES IN A SIX-CUBE FOR DIFFERENT THRESHOLDS AND BUDDY SET SIZES

($\lambda = 0.8$) Buddy set	Threshold / Deadline	0 1 2	1 2 2	1 2 3	1 3 3	2 3 3
4	2	0.3702	0.3491	0.4506	0.4555	0.4873
	3	0.0950	0.0071	0.1856	0.1885	0.2018
	4	0.0320	0.0018	0.0047	0.0041	0.0011
	5	0.0107	0.0004	0.0008	0.0006	0.0002
	6	0.0034	9.1×10^{-5}	0.0002	0.0002	5.8×10^{-5}
10	2	0.3181	0.3448	0.4463	0.4526	0.4856
	3	0.0229	0.0012	0.1806	0.1860	0.2008
	4	0.0043	8.9×10^{-5}	6.4×10^{-5}	0.0004	0.0002
	5	0.0008	8.1×10^{-6}	5.2×10^{-6}	1.6×10^{-5}	6.6×10^{-6}
	6	0.0001	6.4×10^{-7}	4.7×10^{-7}	2.9×10^{-6}	5.8×10^{-7}
15	2	0.3073	0.3448	0.4463	0.4532	0.4856
	3	0.0089	0.0012	0.1806	0.1863	0.2006
	4	0.0013	5.7×10^{-5}	3.9×10^{-5}	0.0004	0.0002
	5	0.0002	5.1×10^{-6}	3.2×10^{-6}	1.1×10^{-5}	7.6×10^{-6}
	6	2.0×10^{-5}	2.4×10^{-7}	1.6×10^{-7}	8.4×10^{-7}	2.6×10^{-7}
21	2	0.3026	0.3445	0.4463	0.4526	0.4856
	3	0.0033	0.0012	0.1806	0.1860	0.2008
	4	0.0004	6.0×10^{-5}	3.7×10^{-5}	0.0004	0.0002
	5	9.2×10^{-5}	4.1×10^{-6}	2.2×10^{-6}	1.1×10^{-5}	4.6×10^{-6}
	6	1.1×10^{-5}	1.7×10^{-7}	$< 10^{-7}$	7.4×10^{-7}	2.9×10^{-7}

TABLE V
NUMBER OF TASK TRANSFERS VERSUS FREQUENCY OF STATE CHANGE FOR DIFFERENT THRESHOLDS.

($\lambda = 0.5$)	Transferred Tasks		Frequency of State Change	
Threshold	Simulation	Analytic	Simulation	Analytic
0 1 1	0.1071	0.1300	1.0792	1.1128
0 1 2	0.0304	0.0330	1.0203	1.0266
0 2 2	0.0484	0.0501	0.1327	0.1471
1 2 2	0.0332	0.0392	0.1574	0.1754
1 2 3	0.0092	0.0098	0.1511	0.1552
1 3 3	0.0097	0.0101	0.0407	0.0370
2 3 3	0.0089	0.0100	0.0472	0.0487

($\lambda = 0.8$)	Transferred Tasks		Frequency of State Change	
Threshold	Simulation	Analytic	Simulation	Analytic
0 1 1	0.2241	0.2230	0.6277	0.8118
0 1 2	0.1145	0.1600	0.5283	0.5984
0 2 2	0.2274	0.2015	0.1613	0.2531
1 2 2	0.1677	0.1891	0.2576	0.3316
1 2 3	0.0877	0.0811	0.2182	0.2404
1 3 3	0.1064	0.0934	0.1050	0.1204
2 3 3	0.0875	0.1050	0.1646	0.1782

can be transferred upon its arrival without probing any other nodes. Besides, each task transfer in our LS method will take less time than other LS methods, because use of a preferred list will usually locate a receiver in the sender's physical proximity.

7) Delays in Task Transfer and State-Change Broadcasts: When a node selects, and transfers a task to, a U-state node, the U-state node may receive an external task and switch to V state before the transferred task arrives. In this case, the transferred task will actually arrive at a V-state node. Thus, the probability of transition to V state with nonzero task arrivals (i.e., α_k for $k > 0$) is larger than that used in (4.5), where transferred tasks are assumed to be accepted only when a receiving node is in U state. One can estimate this probability and adjust the corresponding α_k's. The broadcasting delay has the same effect as the task transfer delay.

Although these delays may affect the distribution of QL, the missing probability can be made insensitive to them by properly choosing a threshold pattern. For example, a task which arrives when QL = TH_f will not be transferred again if $TH_v > TH_f$, e.g., threshold "1 2 3," but it will be retransferred if $TH_v = TH_f$, e.g., threshold "1 2 2." Since task retransfers induce traffic overheads without improving the capability of meeting deadlines, the threshold patterns that are sensitive to those delays may result in a higher missing probability than those that are not. The effect of these delays

TABLE VI
MISSING PROBABILITIES VERSUS TASK TRANSFER COSTS FOR DIFFERENT DEADLINES AND THRESHOLDS

($\lambda^e = 0.8$) Transfer Cost	Threshold / Deadline	0 1 2	1 2 2	1 2 3	2 3 3
5%	2	0.3227	0.3460	0.4473	0.4860
	3	0.0307	0.0013	0.1812	0.2007
	4	0.0067	0.0002	0.0003	0.0001
	5	0.0015	6.51×10^{-5}	4.26×10^{-5}	1.65×10^{-5}
	6	0.0004	2.83×10^{-5}	9.11×10^{-6}	5.98×10^{-6}
10%	2	0.3258	0.3461	0.4468	0.4858
	3	0.0305	0.0019	0.1812	0.2007
	4	0.0067	0.0003	0.0002	0.0002
	5	0.0015	8.37×10^{-5}	2.11×10^{-5}	2.28×10^{-5}
	6	0.0004	3.61×10^{-5}	1.11×10^{-5}	6.18×10^{-6}
20%	2	0.3300	0.3530	0.4504	0.4847
	3	0.0295	0.0046	0.1835	0.2017
	4	0.0065	0.0005	0.0004	0.0007
	5	0.0015	8.69×10^{-5}	5.58×10^{-5}	5.31×10^{-5}
	6	0.0004	3.8×10^{-5}	1.29×10^{-5}	6.33×10^{-6}
30%	2	0.3488	0.3817	0.4681	0.4914
	3	0.0320	0.0168	0.1956	0.2177
	4	0.0072	0.0023	0.0016	0.0050
	5	0.0016	0.0003	0.0002	0.0005
	6	0.0004	4.34×10^{-5}	2.93×10^{-5}	4.95×10^{-5}

on the missing probability is investigated further in our simulation.

B. Simulation Results

For our simulation, the average load density is varied from 0.5 to 0.9, and buddy sets of size 4, 10, and 15 are considered. Ten threshold patterns are chosen out of all possible combinations for the simulation of a four-cube system and the results are given in all tables except for Table II. A few selected thresholds for a six-cube system are also simulated and given in Table IV. The time for transferring a task between two nodes within a buddy set is assumed to be 10 percent of the task execution time and the time for informing a state change to one of the nodes in a buddy set is assumed to be 1 percent of the task execution time.

In most cases, simulation results are consistent with, and close to, the approximate solution. However, the analytically derived q_k's for $k > TH_v$ are always less than those obtained from simulation when the system is underloaded ($\lambda \leq 0.5$) or overloaded ($\lambda \geq 0.9$), especially in the threshold patterns with $TH_u > 0$. This discrepancy may have been caused by the delays in transferring tasks and broadcasting state changes. The effect of settting TH_v to be larger than TH_f is also observed in the simulation. The percentage of task retransfers for the case of $TH_f = TH_v$ is higher than that for the case of $TH_f < TH_v$. Hence, the threshold pattern with $TH_f < TH_v$ is a better choice than the pattern with $TH_f = TH_v$. This observation also explains why the missing probability associated with threshold "1 2 3" is smaller than that of "1 2 2" when $\lambda > 0.7$, deadline > 3, and buddy set size > 10.

To study the effect of changing task transfer costs, we ran simulations with task transfer costs 5, 10, 20, and 30 percent of the task execution time. As shown in Table VI, the missing probability of threshold "1 2 3" remains almost unchanged. Based on all the above results, it is concluded that threshold "1 2 3" is good for a wide range of system load. Note that, although the missing probability of threshold "1 2 2" is usually close to that of threshold "1 2 3," the task transfer rate associated with "1 2 2" is much higher than that with "1 2 3." Thus, considering cost-performance effectiveness, threshold "1 2 3" is a better choice than "1 2 2."

C. Advantages of Using Analytic Approaches

There are several advantages of using the upper bound model and the approximate solution, as compared to simulations. First, the result derived from the upper bound model can be used to guarantee the specified system reliability, because the actual missing probability is always less than that derived from the upper bound model. Second, system utilization can be analyzed by using the analytic models. Third, our analytic models provide, at almost no cost, many pieces of useful information with accuracy. For example, any meaningful simulation of our LS method requires hundreds of CPU hours (in a computer as powerful as VAX-11/780) to get an

accuracy of 10^{-6} in the calculation of q_k's for a system of moderate size. Moreover, simulation may be able to provide information only for a particular system workload; it is too costly to generate q_k's with simulation as a function of system workload.

VI. Conclusion

We have proposed and analyzed a new LS method based on state-change broadcasts. By selecting an appropriate threshold pattern and a buddy set, one can reduce the missing probability to a small number, and thus, the proposed LS method has high potential use for various real-time applications. The traffic overhead for broadcasting state changes can be controlled to an acceptable level by selecting an appropriate threshold pattern.

There are several issues which warrant further investigation. First, it is necessary, but difficult, to derive an exact analytic formula for the probability of a task missing its deadline. Second, if each task has a different execution time, QL is not sufficient to determine the workload of each node. In such a case, one must consider the actual task execution times and use the cumulative execution time to determine the load of each node. Furthermore, if a node thinks itself to be underloaded and broadcasts its availability to other nodes, it may receive a task whose computation is too involved for the node to complete in time. Thus, the state of a node must contain a sufficient amount of information to ensure that the underloaded node can process all transferred-in tasks in time. Third, if the task execution time is a random variable, a continuous-time Markov model must be used to simulate and analyze system performance.

Optimization of the tradeoffs existing in the proposed LS method is an interesting design problem of its own. For example, there is a tradeoff between the buddy set size and LS capability. The traffic overhead of state-change broadcasts can be reduced by shrinking the buddy set size, but this will limit the LS cpability. All of these issues are matters of our future inquiry.

References

[1] K. G. Shin, C. M. Krishna, and Y. -H. Lee, "A unified method for evaluating real-time computer controllers and its application," *IEEE Trans. Automat. Contr.*, vol. AC-30, pp. 357–366, Apr. 1985.

[2] D. W. Leinbaugh, "Guaranteed response times in a hard-real-time environment," *IEEE Trans. Software Eng.*, vol. SE-6, pp. 85–93, Jan. 1980.

[3] Y. -T. Wang and R. J. T. Morris, "Load sharing in distributed systems," *IEEE Trans. Comput.*, vol. C-34, pp. 204–217, Mar. 1985.

[4] D. L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Software Eng.*, vol. SE-12, pp. 662–675, May 1986.

[5] J. F. Kurose and R. Chipalkatti, "Load sharing in soft real-time distributed computer systems," *IEEE Trans. Comput.*, vol. C-36, pp. 993–999, Aug. 1987.

[6] P. S. Yu, S. Balsamo, and Y. -H. Lee, "Dynamic transaction routing in distributed database systems," *IEEE Trans. Software Eng.*, vol. SE-14, pp. 1307–1318, Sept. 1988.

[7] P. Krueger and R. Finkel, "An adaptive load balancing algorithm for a multicomputer," Comput. Sci. Tech. Rep. 539, Univ. of Wisconsin-Madison, 1987.

[8] A. Kratzer and D. Hammerstorm, "A study of load levelling," in *Proc. IEEE Real-Time Syst. Symp.*, 1980, pp. 647–652.

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

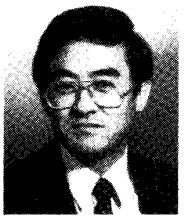
[10] L. M. Ni and K. Hwang, "Optimal load balancing in a multiple processor system with many job systems," *IEEE Trans. Software Eng.*, vol. SE-11, pp. 491–496, May 1985.

[11] Y. -C. Chow and W. H. Kohler, "Model for dynamic load balancing in a heterogeneous multiple processor system," *IEEE Trans. Comput.*, vol. C-28, pp. 354–361, May 1979.

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

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

[14] L. Kleinrock, *Queueing Systems Vol. I: Theory*. New York: Wiley, 1975.



Kang G. Shin (S'75-M'78-SM'83) received the B.S. degree in electronics engineering from Seoul National University, Seoul, Korea in 1970, and the M.S. and Ph.D. degrees in electrical engineering from Cornell University, Ithaca, NY, in 1976 and 1978, respectively.

He is a Professor in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, which he joined in 1982. He has been very active and authored/coauthored over 140 technical papers in the areas of fault-tolerant computing, distributed real-time computing, computer architecture, and robotics and automation. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called HARTS, to validate various architectures and analytic results in the area of distributed real-time computing. From 1970 to 1972 he served in the Korean Army as an ROTC officer and from 1972 to 1974 he was on the research staff of the Korea Institute of Science and Technology, Seoul, Korea, working on the design of VHF/UHF communication systems. From 1978 to 1982 he was an Assistant Professor at Rensselaer Polytechnic Institute, Troy, NY. He was also a visiting scientist at the U.S. Airforce Flight Dynamics Laboratory in Summer 1979 and at Bell Laboratories, Holmdel, NJ, in Summer 1980. During the 1988–1989 academic year, he was a Visiting Professor in the CS Division, Electrical Engineering and Computer Science, University of California, Berkeley.

Dr. Shin was the Program Chairman of the 1986 IEEE Real-Time Systems Symposiuim (RTSS), the General Chairman of the 1987 RTSS and the Guest Editor of the 1987 August special issue of IEEE TRANSACTIONS ON COMPUTERS on Real-Time Systems. He is a member of the Association for Computing Machinery, Sigma Xi, and Phi Kappa Phi. In 1987, he received the Outstanding Paper Award from the IEEE TRANSACTIONS ON AUTOMATIC CONTROL for a paper on robot trajectory planning.

Yi-Chieh Chang (S'84) was born in Shienchou, Taiwan, Republic of China, on September 14, 1957. He received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, Republic of China, in 1979 and 1984, respectively.

He is currently a Ph.D. candidate in the Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor. His research interests include computer architecture, parallel processing, and distributed and real-time systems.