Optimal Load Sharing in Distributed Real-Time Systems*

YI-CHIEH CHANG AND KANG G. SHIN

Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, Michigan 48109-2122

The main goal of this paper is to derive an approximate, closed-form solution for the decentralized, dynamic load sharing (LS) problem treated in an earlier paper. In that paper, whenever the load state of a node changes from underloaded to fully loaded and vice versa, the node broadcasts this change to a set of nodes in its physical proximity, called a *buddy* set. An overloaded node can select, without probing other nodes, the first available node from its *preferred list*, an ordered buddy set. Preferred lists are so constructed that the probability of more than one overloaded node sending tasks to an underloaded node may be made very small. In hard real-time systems, the problem of scheduling periodic tasks to meet their deadlines has been studied extensively, but scheduling aperiodic tasks has been addressed far less, due mainly to their random arrivals. We show that the proposed LS method can be used to effectively handle aperiodic tasks in distributed real-time systems. The probability of missing task deadlines can be kept below a specified level by choosing appropriate threshold patterns and buddy set sizes which are derived from the approximate closed-form solution. Specifically, "optimal" threshold patterns and buddy set sizes are derived for different system loads by minimizing the communication overhead subject to a constraint of keeping the probability of missing task deadlines below any given level. (One can also derive optimal solutions by minimizing the probability of missing deadlines while keeping the communication overhead below a specified level.) Several examples are presented to demonstrate the power and utility of the proposed LS approach. © 1993 Academic Press, Inc.

1. INTRODUCTION

Failure to complete a real-time task before its deadline could cause a disastrous accident [2–4]. Real-time applications are composed of periodic and aperiodic tasks. Liu and Layland proved that scheduling tasks based on *rate-monotonic priority assignment* is optimal for independent periodic tasks in a single processor system [5]. They also derived an upper bound of processor utilization, below which all periodic tasks can be guaranteed to meet their deadlines. The problem of scheduling real-time tasks on multiprocessor/distributed systems is shown to be NP-hard [6], thereby leading to the development of many heuristic approaches [7–12]. Although some of these approaches, such as those in [8, 12], considered the possible arrival of aperiodic tasks, they assumed/implied that periodic tasks form the major portion of the task system.

Since aperiodic tasks arrive randomly, it is in general impossible to guarantee the completion of aperiodic tasks before their deadlines. As was pointed out in [13, 14], using the same scheduling algorithm for both periodic and aperiodic tasks can meet the deadlines of all periodic tasks but cannot always meet the deadlines of aperiodic tasks. Keeping the probability of missing deadlines, called the *probability of dynamic failure*, P_{dyn} [2], below a certain required level while minimizing the ensuing overhead is the only meaningful course to take for scheduling aperiodic tasks.

Scheduling both periodic and aperiodic tasks in real-time systems has not been treated extensively until recently. Ramamritham *et al.* proposed combining local and global scheduling approaches in distributed systems for both periodic and aperiodic tasks [12]. In their approach, periodic tasks are assumed to be known a priori and can always be scheduled locally. On the other hand, an aperiodic task may arrive at a node at any time and will be scheduled locally on the node if its deadline can be met there; otherwise, the task will be transferred to a remote node, which was called *global scheduling* in [12]. If none of the remote nodes can guarantee the deadline of this aperiodic task, it will be rejected and may seriously affect the system performance. Hence, the main effort in [12] was to design a heuristic, global scheduling policy so as to reduce the number of rejected aperiodic tasks. Three algorithms, call *bidding, focused addressing*, and *flexible* algorithms, were used to select a remote node for each aperiodic task which cannot be guaranteed locally. The basic idea of these algorithms is to collect state information, such as the surplus processing power, from other nodes such that, when a node cannot guarantee an aperiodic task locally, it will attempt to locate a remote node which can guarantee the task. However, the impact of the rejected tasks on the system performance was not analyzed by the authors of [12]. It is also worth noting

* The work reported in this paper was supported in part by the Office of Naval Research under Grants N00014-85-K-0122 and N00014-92-J-1080, and 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.

0743-7315/93 $5.00

that these algorithms are variations of the bidding algorithm originally proposed for load balancing in general-purpose distributed systems [15–18], where the primary goal is to reduce the *average* task response time. The bidding algorithm is not efficient for real-time applications due to the long delay of the bidding process [1, 19].

In an early paper [1], we proposed a new load sharing method with state-change broadcast (LSMSCB) for distributed real-time systems, in which each node maintains state information of only a small set of nodes in its physical proximity, called a *buddy set*. The load state of a node is defined by three thresholds: TH_u, TH_f, and TH_v. 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, without probing other nodes, the first underloaded node in its preferred list and transfer a task to that node. Moreover, the buddy sets of the nodes in one buddy set are *different* but are *not* disjoint, thus allowing the surplus tasks in a buddy set to be transferred to many different buddy sets, i.e., system-wide (not local) load sharing. As a result, our method is shown to enable (aperiodic) tasks to be completed before their deadlines with much higher probability than other known methods.

The design of hard real-time systems with a large number of aperiodic tasks by using LSMSCB is quite different from most of the existing LS methods for the following reason. External or transferred-in tasks at each node are processed on a FCFS basis.[1] If the deadline of an arrived task cannot be met locally, it will be transferred to some other node that can complete the task before its deadline. As mentioned earlier, keeping P_{dyn} below a given required level while minimizing the resultant overhead is the only course to take for scheduling aperiodic tasks. Particularly, we will show that even the requirement of P_{dyn} to be as low as 10^{-9} can be met by employing the LSMSCB in a system of 16 nodes with a medium workload, buddy set size 10, and deadlines equal to six times of the average task execution time. Note, however, that the *numerical* solutions for QL used in [1] showed only a few examples, and it is practically too tedious and time-consuming to derive the numerical results for all possible threshold patterns and buddy set sizes. It is therefore important to derive a closed-form distribution of QL to characterize the behavior of a real-time system. Based on this closed-form distribution, "optimal" threshold patterns and buddy set sizes can be determined either by minimizing the communication overhead—such as the frequency of collecting state information and the number of task transfers—incurred by the LSMSCB while keeping P_{dyn} below a specified level, or by minimizing P_{dyn} while keeping the communication cost below a given level. An upper bound of processor utilization can also be derived while reducing P_{dyn} to any given level, and more important, the processor utilization is significantly improved as compared to the one derived from the upper bound model in [1].

The rest of this paper is organized as follows. For the purpose of completeness, Section 2 reviews some relevant aspects of the LSMSCB [1]. The distribution of QL is described as a function of threshold patterns and buddy set sizes in Section 3. Optimal threshold patterns and buddy set sizes are derived in Section 4. Finally, the paper concludes with Section 5.

2. DESCRIPTION OF LSMSCB

Since P_{dyn} (communication cost) must be kept below a given required level while minimizing the resultant overhead (P_{dyn}), the main issues of LSMCB are how to define the state of each node, how to collect state information, and how to redistribute loads among the nodes in the system, such that overloaded nodes will locate underloaded nodes to share their loads with a very high probability. Buddy sets, preferred lists, and threshold patterns are the most important features proposed in [1] to resolve these issues, and they are discussed briefly here for completeness.

The buddy set of a node is a set of nodes in its physical proximity. Since state information is exchanged only within a buddy set and since a constant buddy set size of 10 to 15 nodes is shown to work well regardless of the system size [1],[2] the communication overhead is reduced to a constant from $O(N^2)$, as compared to the case when state information is exchanged in the entire system of N nodes. In order to avoid more than one overloaded node "dumping" their loads on one underloaded node (*coordination problem*) or surplus tasks being confined in a certain region (*congestion problem*), the nodes in a buddy set are ordered into a preferred list such that each node will be selected as the kth preferred node by one and only one other node. It has been shown that the preferred lists proposed in [1] can effectively solve both the coordination and congestion problems [21, 19], thus meeting task deadlines with a high probability.

[1] As discussed in [20], more complex local scheduling algorithms like the minimum-laxity-first policy can be used and analyzed. But use of such a complex local scheduling algorithm will obscure the main point of this paper, i.e., global scheduling.

[2] So, there is no need to increase the buddy set size even if the system gets larger.

Three thresholds, denoted as TH_u, TH_f, and TH_v, are used to determine the load state of each node. Since only underloaded nodes can share surplus tasks, TH_u dictates the probability of an overloaded node finding an underloaded node. The difference between TH_u and TH_f determines the frequency of state change, and hence the frequency of state-change broadcasts. Those tasks arriving at a node whose $QL > TH_v$ will be transferred to other underloaded nodes.

An embedded Markov chain is used to model the performance of LSMSCB. Since (average) QL is used to measure each node's workload, without loss of generality, one can assume (average) task execution time to be one unit of time. Here we will use the same notation as in [1]:

- λ, external task arrival rate (Poisson process)
- τ, task transfer-in rate
- ω, arrival rate of combined external and transferred-in tasks ($=\lambda+\tau$)
- k_t, the number of task arrivals during the interval $[t, t+1)$
- α_{k_t}, the probability of k_t arrivals during the interval $[t, t+1)$ when there are λ arrivals per unit time
- $\alpha^*_{k_t}$, the probability of k_t arrivals during the interval $[t, t+1)$ when there are ω arrivals of combined external and transferred-in tasks per unit time
- x_t, QL at time t
- ε_i, the probability of having exactly i underloaded nodes available to share the surplus tasks within a buddy set
- θ_i, the probability of having at least i underloaded nodes available to share the surplus tasks within a buddy set.

Transition of states can be described [1] as

$$x_{t+1} = \begin{cases} k_t & \text{if } x_t = 0 \text{ and } k_t \le TH_v \\ TH_v \text{ with prob } \theta_{k_t - TH_v}, \text{ or } (TH_v + 1) \text{ with prob } \\ \quad \varepsilon_{k_t - TH_v - 1}, \ldots, \text{ or } k_t \text{ with prob } \varepsilon_0 & \text{if } x_t = 0 \text{ and } k_t > TH_v \\ x_t + k_t - 1 & \text{if } x_t > 0 \text{ and } x_t + k_t - 1 \le TH_v \\ TH_v \text{ with prob } \theta_{x_t + k_t - TH_v - 1}, \text{ or } (TH_v + 1) \text{ with prob } \\ \quad \varepsilon_{x_t + k_t - TH_v - 2}, \ldots, \text{ or } (x_t + k_t - 1) \text{ with prob } \varepsilon_0 & \text{if } x_t > 0 \text{ and } x_t + k_t - 1 > TH_v. \end{cases} \tag{2.1}$$

For notational convenience, let $u = TH_u, f = TH_f$, and $v = TH_v$. The exact state equations are

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

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

$$\vdots$$

$$q_u = \alpha_u^* q_0 + \alpha_u^* q_1 + \alpha_{u-1}^* q_2 + \cdots + \alpha_1^* q_u + \alpha_0 q_{u+1}$$

$$q_{u+1} = \alpha^*{}_{u+1} q_0 + \alpha_{u+1}^* q_1 + \alpha_u^* q_2 + \cdots + \alpha_1 q_{u+1} + \alpha_0 q_{u+2}$$

$$\vdots$$

$$q_f = \alpha_f^* q_0 + \alpha_f^* q_1 + \alpha_{f-1}^* q_2 + \cdots + \alpha_{f-u+1}^* q_u + \alpha_{f-u} q_{u+1} + \cdots + \alpha_1 q_f + \alpha_0 q_{f+1}$$

$$\vdots$$

$$q_v = \left(\alpha_v^* + \sum_{i=1}^{\infty} \theta_i \alpha_{v+i}^*\right) (q_0 + q_1) + \sum_{i=2}^{u} \left(\alpha_{v+1-i}^* = \sum_{j=1}^{\infty} \theta_j \alpha_{j+v+1-i}^*\right) q_i$$

$$+ \sum_{i=u+1}^{v+1} \left(\alpha_{v+1-i} + \sum_{j=1}^{\infty} \theta_j\alpha_{j+v+1-i}\right) q_i + \sum_{i=v+2}^{\infty} \left(\sum_{j=i-v-1}^{\infty} \theta_j\alpha_{j-i+v+1}\right) q_i$$

$$q_k = \sum_{i=0}^{n} \varepsilon_i\alpha^*_{i+k}(q_0 + q_1) + \sum_{i=2}^{u} \left(\sum_{j=0}^{\infty} \varepsilon^*_j\alpha_{j+k-i+1}\right) q_i + \sum_{i=u+1}^{k+1} \left(\sum_{j=0}^{\infty} \varepsilon_j\alpha_{j+k-i+1}\right) q_i$$

$$+ \sum_{i=k+2}^{\infty} \left(\sum_{j=0}^{\infty} \varepsilon_{j+i-k-1}\alpha_j\right) q_i \quad \text{for } k = v + 1, \ldots, \infty. \quad (2.2)$$

Note that in [1] we did not address how to derive a closed-form solution to the above equations; only numerical solutions for a few specific threshold patterns are derived there. However, we need a closed-form solution to Eq. (2.2) in order to determine optimal threshold patterns and buddy set sizes and to check whether or not P_{dyn} can be kept below a prespecified value.

3. DISTRIBUTION OF QUEUE LENGTH

Equation (2.2) cannot be solved in one step for two reasons. First, α^*'s in these equations depend on the task transfer-in rate, τ, which in turn depends on q_k's. Second, since q_k's for $k > v$ depend on those q_k's for $k \le v$, it is impossible to obtain the distribution of queue length in one step. Since the probability of $QL > v$ is very small, Eq. (2.2) can be divided into two parts which are then solved separately: q_k's for $0 \le k \le v$ and q_k's for $k > v$.

3.1. Solving q_k for $0 \le k \le v$

Equation (2.2) can be rewritten as

$$q_1 = \frac{1 - \alpha_0^*}{\alpha_0^*} q_0$$

$$\frac{\alpha_1^* - 1}{\alpha_0^*} q_1 + q_2 = -\frac{\alpha_1^*}{\alpha_0^*} q_0$$

$$\frac{\alpha_2^*}{\alpha_0^*} q_1 + \frac{\alpha_1^* - 1}{\alpha_0^*} q_2 + q_3 = -\frac{\alpha_2^*}{\alpha_0^*} q_0$$

$$\vdots$$

$$\frac{\alpha_{u-1}^*}{\alpha_0^*} q_1 + \frac{\alpha_{u-2}^*}{\alpha_0^*} q_2 + \cdots + \frac{\alpha_2^*}{\alpha_0^*} q_{u-2} + \frac{\alpha_1^* - 1}{\alpha_0^*} q_{u-1} + q_u = -\frac{\alpha_{u-1}^*}{\alpha_0^*} q_0$$

$$\vdots$$

$$\frac{\alpha_{v-1}^*}{\alpha_0} q_1 + \frac{\alpha_{v-2}^*}{\alpha_0^*} q_2 + \cdots + \frac{\alpha_{v-u}^*}{\alpha_0} q_u + \cdots + \frac{\alpha_1 - 1}{\alpha_0} q_{v-1} + q_v = -\frac{\alpha_{v-1}^*}{\alpha_0} q_0 \quad (3.1)$$

Equation (3.1) can also be expressed in vector form as $A_1Q_1 = Cq_0$, where A_1 is a $v \times v$ lower triangular matrix, $Q_1 = [q_1\ q_2 \cdots q_u \cdots q_v]^{\mathrm{T}}$, and

$$C = \left[\frac{1 - \alpha_0^*}{\alpha_0^*}\ -\frac{\alpha_1^*}{\alpha_0^*}\ -\frac{\alpha_2^*}{\alpha_0^*} \cdots -\frac{\alpha_{u-1}^*}{\alpha_0^*}\ -\frac{\alpha_u^*}{\alpha_0}\ -\frac{\alpha_{u+1}^*}{\alpha_0} \cdots -\frac{\alpha_{v-1}^*}{\alpha_0}\right]^{\mathrm{T}}.$$

Note that there are $v + 1$ variables in Q_1 and only v equations in Eq. (3.1). The normalization equation is needed to solve Eq. (3.1) for Q_1:

$$q_0 + q_1 + \cdots + q_v = 1 - \xi, \quad \text{where } \xi = \sum_{k=v+1}^{k_{\max}} q_k. \tag{3.2}$$

Since ξ is usually very small ($<10^{-4}$), Eq. (3.1) will give a good approximate solution even if ξ is set to zero.

We now want to compute $Q_1 = A_1^{-1}C$. Since A_1 is a lower triangular matrix, A_1^{-1} will also be a lower triangular matrix. For convenience, let $b_{i,j}$ and $c_{i,j}$ be the nonzero elements of A_1^{-1} and A_1, respectively, and let C_i be the elements of C. Then for $i, j = 1, \ldots, v$,

$$\sum_{k=1}^{v} b_{i,k}c_{k,j} = \begin{cases} 1 & i = j \\ 0 & i \neq j. \end{cases}$$

Solving Eqs. (3.1) and (3.2) for $b_{i,j}$, we obtain

$$b_{i,j} = 0, \quad j > i,$$

$$b_{i,i} = 1, \quad 1 \le i \le v,$$

$$b_{i,i-1} = -c_{i,i-1} = \frac{1 - \alpha_1^*}{\alpha_0^*}, \quad 1 \le i \le u,$$

$$b_{i,i-1} = -c_{i,i-1} = \frac{1 - \alpha_1^*}{\alpha_0}, \quad u + 1 \le i \le v,$$

$$b_{i,j} = \frac{1 - \alpha_1^*}{\alpha_0^*} - \sum_{k=2}^{i-j} \frac{\alpha_k^*}{\alpha_0^*} b_{i,j+k}, \quad 1 \le i \le u, \quad 1 \le j \le i - 1,$$

$$b_{i,j} = \frac{1 - \alpha_1^*}{\alpha_0^*} - \sum_{k=2}^{i-j} \frac{\alpha_k^*}{\alpha_0} b_{i,j+k}, \quad u + 1 \le i \le v, \quad 1 \le j \le i - 1.$$

Substituting $b_{i,j}$ into Eq. (3.1), we have

$$q_k = \left[\sum_{j=1}^{k} b_{k,j}C_j\right] q_0, \quad 1 \le k \le v. \tag{3.3}$$

Substituting q_k's in Eq. (3.3) into Eq. (3.2),

$$q_0 = \frac{1 - \xi}{1 + [\sum_{k=1}^{v} \sum_{j=1}^{k} b_{k,j}C_j]}. \tag{3.4}$$

Then, q_k's for $1 \le k \le v$ can be determined by substituting Eq. (3.4) into Eq. (3.3).

For example, when $u = 1, f = 2, v = 3$, the distribution of queue length becomes

$$q_0 = \frac{(1 - \xi)\alpha_0^*\alpha_2^*}{\alpha_0^* - \alpha_2^* + (1 + \alpha_0 - \alpha_1)(1 - \alpha_0^* - \alpha_1^*)\alpha_2^*}$$

$$q_1 = \frac{1 - \alpha_0^*}{\alpha_0^*} q_0 = \frac{(1 - \delta)(1 - \alpha_0^*)\alpha_2^*}{\alpha_0^* - \alpha_2^* + (1 + \alpha_0 - \alpha_1)(1 - \alpha_0^* - \alpha_1^*)\alpha_2^*}$$

$$q_2 = \frac{1 - \alpha_0^* - \alpha_1^*}{\alpha_0\alpha_0^*} q_0 = \frac{(1 - \xi)(1 - \alpha_0^* - \alpha_1^*)\alpha_2^*}{\alpha_0(\alpha_0^* - \alpha_2^* + (1 + \alpha_0 - \alpha_1)(1 - \alpha_0^* - \alpha_1^*)\alpha_2^*)}$$

$$q_3 = \frac{1}{\alpha_0\alpha_0^*}\left[\frac{(1 - \alpha_1)(1 - \alpha_0^* - \alpha_1^*)}{\alpha_0} - \alpha_2^*\right] q_0.$$

3.2. Solving q_k for $k > v$

We now want to derive q_k's for $k > v$. The last formula in Eq. (2.2) was

$$q_k = \sum_{i=0}^{n} \varepsilon_i\alpha_{i+k}^*(q_0 + q_1) + \sum_{i=2}^{u}\left(\sum_{j=0}^{\infty} \varepsilon_j^*\alpha_{j+k-i+1}\right) q_i + \sum_{i=u+1}^{k+1}\left(\sum_{j=0}^{\infty} \varepsilon_j\alpha_{j+k-i+1}\right) q_i + \sum_{i=k+2}^{\infty}\left(\sum_{j=1}^{\infty}\varepsilon_{j+i-k-1}\alpha_j\right) q_i \quad \text{for } k = v = 1, \ldots, \infty.$$

To simplify this expression, define

$$\nu_i \overset{\text{def}}{=} \sum_{j=0}^{\sigma} \varepsilon_j\alpha_{i+j}, \quad 0 \le i \le k_{\max}$$

$$B_k \overset{\text{def}}{=} \sum_{j=0}^{n} \varepsilon_j\alpha_{j+v+k}^* q_0 + \sum_{i=1}^{u}\left(\sum_{j=0}^{\infty} \varepsilon_j\alpha_{v+k+j-i+1}^*\right) q_i + \sum_{i=u+1}^{v}\left(\sum_{j=0}^{\infty} \varepsilon_j\alpha_{v+k+j-i+1}\right) q_i, \quad 1 \le k \le k_{\max}$$

Since $q_k \ll 1$ for $k > v$, the terms related to q_{k+2}, $k > v$, in Eq. (2.2) can be set to zero, yielding approximate equations,

$$\begin{aligned}
(1 - \nu_1)q_{v+1} \quad - \nu_0 q_{v+2} \quad\quad\quad\quad\quad &= B_1 \\
-\nu_2 q_{v+1} \quad + (1 - \nu_1)q_{v+2} \quad - \nu_0 q_{v+3} &= B_2 \\
&\vdots \\
-\sum_{i=1}^{k-2} \nu_{k-i}q_{v+i} \quad + (1 - \nu_1)q_{v+k-1} \quad - \nu_0 q_{v+k} &= B_{k-1}
\end{aligned}$$

$4 \le k \le k_{\max}$, where $q_{v+k_{\max}} \ll P_{\text{dyn}}$.

The above equations can be rewritten as $A_2Q_2 = B$, where $Q_2 = [q_{v+1}q_{v+2} \cdots q_{v+k_{max}}]^T$, $B = [B_1B_2 \cdots B_{k_{max}-1}]^T$, and A_2 is a $(k_{max} - 1) \times (k_{max} - 1)$ lower triangular matrix of v_0 is set to zero. The inverse of A_2 is calculated using the perturbation approach as shown below.

3.2.1. Calculation of A_2^{-1}

Let A_2^e be the original matrix and A_2 be the matrix after setting ν_0 to zero. Since A_2 is a lower triangular matrix, A_2^{-1} can be easily computed as follows. Let $a_{i,j}$ denote the nonzero (i, j)th element of A_2^{-1}, then

$$a_{i,i} = \frac{1}{1 - \nu_1}$$

$$a_{i,j} = \sum_{k=1}^{i-j} \frac{\nu_{k+1}}{1 - \nu_1} a_{i,j+k}, \quad 1 \le i \le k_{max}, \quad 1 \le j \le i - 1.$$

A few examples of $a_{i,j}$'s are

$$a_{i,i} = \frac{1}{1 - \nu_1}, \quad 1 \le i \le k_{max}$$

$$a_{i,i-1} = \frac{\nu_2}{(1 - \nu_1)^2}, \quad 2 \le i \le k_{max}$$

$$a_{i,i-2} = \frac{\nu_2^2}{(1 - \nu_1)^3} + \frac{\nu_3}{(1 - \nu_1)^2}, \quad 3 \le i \le k_{max}$$

Finally, the distribution of queue length can be calculated as

$$q_{v+k} = \sum_{j=1}^{k} \alpha_{k,j} B_j, \quad 1 \le k \le k_{max}. \tag{3.5}$$

3.2.2. Successive Approximation

The perturbation approach is applied to account for the effect of $\nu_0 \neq 0$. The basic idea of perturbation theory works as follows. Suppose the solution of a linear system $AX = B$ is known. When A and B change to $A + \varepsilon A$ and $B + \varepsilon B$, respectively, the new solution $X + \varepsilon X$ can be derived from A and X instead of inverting $A + \varepsilon A$ if ε is a small number [22].

Let Q_2^e and δA_2 be the exact distribution of queue length and the matrix that contains ν_0, respectively. The solution to $A_2^e Q_2^e = B$ can be successively approximated as follows. Let $Q_2^e = Q_2 + \delta Q_2^e$. Since A_2^{-1} has already been calculated, Q_2 can be computed first. Then, $(A_2 - \delta A_2)\delta Q_2^e = \delta A_2 Q_2$, where δA_2 is a matrix of the same size as A_2 with ν_0 at element $(i, i+1)$, $1 \le i < k_{max}$, and zero otherwise. The same method can be used to solve for δQ_2^e. Let $\delta Q_2^e = \delta Q_2^{(1)} + \delta Q_2^r$, then $(A_2 - \delta A_2)(\delta Q_2^{(1)} + \delta Q_2^r) = \delta A_2 Q_2$. This equation can be divided into two parts which are then solved separately. The first part is $A_2 \delta Q_2^{(1)} = \delta A_2 Q_2$ and the second part is $(A_2 - \delta A_2)\delta Q_2^r = \delta A_2 Q_2^{(1)}$. Since we have already found A_2^{-1}, $\delta Q_2^{(1)}$ can be calculated by $\delta Q_2^{(1)} = A_2^{-1}\delta A_2 Q_2$. This method is applied again to solve for δQ_2^r in the second part. Let $\delta Q_2^r = \delta Q_2^{(2)} + \delta Q_2^{r'}$, then there are two equations similar to the previous iteration:

$$A_2 \delta Q_2^{(2)} = \delta A_2 Q_2^{(1)}$$

$$(A_2 - \delta A_2)\delta Q_2^{r'} = \delta A_2 Q_2^{(2)}.$$

Again, $\delta Q_2^{(2)}$ is calculated first $(= A_2^{-1}\delta A_2 Q_2^{(1)})$, and $\delta Q_2^{r'}$ is decomposed into $\delta Q_2^{(3)}$ and $\delta Q_2^{r''}$. Eventually, δQ_2^e can be approximated as the sum of an infinite series of $\delta Q_2^{(k)}$, $k = 1, \ldots, \infty$, as

$$\delta Q_2^{(1)} = A_2^{-1}\delta A_2 Q_2$$

$$\delta Q_2^{(2)} = A_2^{-1}\delta A_2 Q_2^{(1)} = (A_2^{-1}\delta A_2)^2 Q_2$$

$$\vdots$$

$$\delta Q_2^{(k)} = A_2^{-1}\delta A_2 Q_2^{(k-1)} = (A_2^{-1}\delta A_2)^k Q_2$$

$$\delta Q_2^e = \delta Q_2^{(1)} + \delta Q_2^{(2)} + \cdots = \frac{(A_2^{-1}\delta A_2)}{I - A_2^{-1}\delta A_2}$$

$$Q_2^e = Q_2 + \delta Q_2^e = Q_2 + \frac{(A_2^{-1}\delta A_2)}{I - A_2^{-1}\delta A_2} Q_2$$

$$= \frac{Q_2}{I - A_2^{-1}\Delta A_2} = (I - A_2^{-1}\delta A_2)^{-1} Q_2. \tag{3.6}$$

In most calculations, Q_2^e converges to a fixed constant after two to three iterations, it is not necessary to invert $(I - A_2^{-1}\delta A_2)$ in Eq. (3.6). Combining Eqs. (3.3) to (3.6), the distribution of queue length can be derived for any threshold patterns.

3.3. Deriving the Combined Task Arrival Rate ω

Two unknown parameters, ω and ε's in Eqs. (3.3)–(3.6), need to be derived before solving these equations. Derivation of the combined task arrival rate ω is discussed in this section. In the proposed LSMSCB, a node is overloaded when its queue length is greater than v. Only an overloaded node can transfer tasks to other underloaded nodes in its buddy set. The task transfer-out rate (β) is shown as

$$\beta \equiv \left[\sum_{k=v+1}^{\infty} (k - v)\alpha_k^*\right](q_0 + q_1)$$
$$+ \sum_{i=2}^{v+1}\left[\sum_{k=v+2-i}^{\infty} (k - v - 1 + i)\alpha_k\right] q_i \tag{3.7}$$
$$+ \sum_{i=v+2}^{\infty}\left[\sum_{k=0}^{\infty} k\alpha_k\right] q_i.$$

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 using Eq. (4.11) in [1].

Combining τ and λ, the distribution of q_k's for $k \le v$ can be calculated by Eqs. (3.3) and (3.4). Substituting the newly calculated q_k's into Eq. (3.7), one can obtain a new τ. The same procedure is applied again to adjust q_k's. This procedure will repeat until q_k's and ω converge to a fixed number. We have proved in [1] that this procedure will always converge in a few iterations.

3.4. Relating q_k to Buddy Set Size

The size (σ) of a buddy set will influence the probability of an overloaded node finding an underloaded node from its buddy set. This is reflected in deriving ε_i's. From the definition, ε_i is the probability of having exactly i underloaded nodes available to share the surplus tasks within a buddy set. Let P^{nsh} be the probability of a node not in sharing mode, i.e., the probability when a node's queue length is greater than f. Then

$$\varepsilon_0 = f(\sigma) = \prod_{i=1}^{\sigma} P_{n_i}^{\text{nsh}} \approx (P^{\text{nsh}})^{\sigma}$$

$$\varepsilon_i \approx \binom{\sigma}{i}(P^{\text{nsh}})^{\sigma-i}(1 - P^{\text{nsh}})^i, \quad \text{where } P^{\text{nsh}} = 1 - \sum_{i=0}^{u} q_i. \tag{3.8}$$

Since ε_i's appear only in q_k's for $k > v$, the size of a buddy set is closely related to P_{dyn}.

The approximation in Eq. (3.8) is needed to simplify the derivation of ε's for the following reasons. Consider the case of ε_0; the probability that every node in a buddy set is not available to accept transferred tasks and can be expressed as

$$\varepsilon_0 = P(x_1 \ge f, x_2 \ge f, \cdots, x_\sigma \ge f), \tag{3.9}$$

where x_i is the queue length of the nodes in the buddy set. Since the nodes in a buddy set depend on each other, the combined task arrival rate, ω, will be changed for the nodes in a sharing mode when some nodes are not in a sharing mode, because the nodes in a sharing mode are very likely to receive more transferred tasks. Equation (3.9) can only be solved step-by-step due to the dependence among nodes. Given $x_1 \ge f$, the first step is to recalculate q_k's and ω for the range of x_2 to x_σ. Then, q_k and ω for x_3 to x_σ can be recalculated under the condiiton that $x_1 \ge f$ and $x_2 \ge f$. The same procedure will apply as long as all but one node in a buddy set are in a nonsharing mode. This procedure translates the node dependence within a buddy set to an independent state, with each node having a different combined task arrival rate. Then, we have

$$\varepsilon_0 = P(x_1 \ge f)P(x_2 \ge f) \cdots P(x_\sigma \ge f). \tag{3.10}$$

Note that in Eq. (3.10) the distribution of queue length at a node is different from that at other nodes, because the ω value of node i is adjusted, given that nodes one to $i - 1$ are not in a sharing mode.

Since there are 2^σ patterns to be considered to derive the exact value of ε_k's, each of these patterns needs to be treated similarly to the procedure of deriving ε_0. It is practically too tedious to use this approach. We have shown in [1] that the following approximation can simplify the derivation of ε_k's,

$$\varepsilon_k \approx \frac{n!}{(n-k)!k!}(P^{\text{nsh}})^{n-k}(P^{\text{sh}})^k, \tag{3.11}$$

where P^{nsh} is the probability of a node not in sharing mode, given that all other nodes in its buddy set are already in a nonsharing mode.

3.5. Approximation Accuracy

Due to the complexity of Eq. (3.1), the closed-form solution is derived by separating it into two parts and solving them approximately. It is desirable to analyze the accuracy of the derived solution.

The task transfer-in rate τ is approximated by Eq. (3.7). Since ε cannot be determined before deriving τ, only q_k's for $k < v$ are used in Eq. (3.7). Let $\delta\tau$ be the difference between the derived results and the actual value; then

$$\delta\tau = \left[\sum_{k=v+1}^{\infty}(k - v)\alpha_k^*\right](\delta q_0 + \delta q_1) + \sum_{i=2}^{v+1}\left[\sum_{k=v+2-i}^{\infty}(k - v - 1 + i)\alpha_k\right]\delta q_i + \sum_{i=v+2}^{\infty}\left[\sum_{k=0}^{\infty}k\alpha_k\right]\delta q_i,$$

where

$$\delta q_0 = \frac{-\xi}{1 + [\sum_{k=1}^{v}\sum_{j=1}^{k} b_{k,j}C_j]}$$

$$\delta q_k = \left[\sum_{j=1}^{k} b_{k,j}C_j\right]\delta q_0, \quad 1 \le k \le v,$$

$$\delta q_k = q_k, \quad k > v. \tag{3.12}$$

TABLE I
Variation of q_k's and τ By Omitting ξ

	$\xi = 0$	$\xi = 4.9 \times 10^{-4}$	Variation	Variation in percentage
β	0.59	0.65	0.06	10.17
q_0	0.2352	0.2327	−0.0025	−1.08
q_1	0.3203	0.3200	−0.0003	−0.1
q_2	0.2630	0.2641	0.0011	0.44
q_3	0.1816	0.1828	0.0012	0.66
q_4	4.03×10^{-4}	4.06×10^{-4}	3.08×10^{-6}	0.76
q_5	7.26×10^{-5}	7.30×10^{-5}	5.6×10^{-7}	0.77
q_6	1.10×10^{-5}	1.12×10^{-6}	8.72×10^{-8}	0.78
q_7	1.48×10^{-6}	1.49×10^{-6}	1.17×10^{-8}	0.79
q_8	1.70×10^{-7}	1.72×10^{-7}	1.4×10^{-9}	0.82

The variation of τ will in turn affect the distribution of queue length. As an example, the changes of τ and queue length are studied for the threshold pattern $u = 1, f = 2, v = 3$ with buddy set size 10 and system load $\rho = 0.8$. As shown in Table I, the transfer-in task rate is 0.059 (0.065) when ξ is omitted (considered); the difference is about 10%. The change of q_k's is around 1% due to the variation of τ. The beauty of the proposed approximate method in deriving τ is that omission of ξ increases q_k's for $k \leq v$ only by a small amount while q_k's for $k > v$ are completely ignored to compensate for the omission of ξ. So, τ can be derived quite accurately.

The variation of ε_k's versus the change of distribution of queue length is given in Table II. The variation of ε_k's is found to be significantly larger than the change of q_k's for $k \leq 3$. The variation of q_k's for $k > 3$ due to the change of ε_k's is found to be about 10%. Note that the variation of q_k's for $k > 3$ given in Table I is derived by considering only the variation of τ (and assuming ε_k's unchanged). It is clear that ε_k's dominate the variation of q_k's for $k > 3$.

Summarizing the above analysis, we found that the first part of q_k's (for $k \leq v$) is not sensitive to the second part of q_k's (for $k > v$) as long as ξ is small. However, the second part of q_k's is very sensitive to the accuracy of the first part of q_k's. Since the approximate closed-form solutions determine the first part of q_k's very accurately, the inaccuracy of the second part of q_k's and P_{dyn} is within 10% of the corresponding true value when $\xi \leq 10^{-4}$.

TABLE II
Variation of q_k's versus the Change of ϵ_k

	$\xi = 0$	$\xi = 4.9 \times 10^{-4}$	Variation	Variation in percentage
ε_0	0.001749	0.001932	0.000183	10.46
ε_1	0.015509	0.016771	0.001262	8.14
ε_2	0.061889	0.065514	0.003625	5.86
ε_3	0.146354	0.151663	0.005309	3.63
q_4	4.03×10^{-4}	4.35×10^{-4}	3.2×10^{-5}	7.9
q_5	7.26×10^{-5}	7.87×10^{-5}	6.1×10^{-6}	8.4
q_6	1.10×10^{-5}	1.21×10^{-5}	1.1×10^{-6}	10.0
q_7	1.48×10^{-6}	1.62×10^{-6}	1.4×10^{-7}	9.46
q_8	1.70×10^{-7}	1.87×10^{-7}	1.7×10^{-8}	10.0

4. DESIGN OF AN OPTIMAL LSMSCB

The QL derived from Eqs. (3.3) to (3.6) is verified/compared against our early results in [1]. As shown in Fig. 1, in most cases, the QL derived from the closed-form equations is closer to the true (i.e., simulation) result and is also consistent with the results calculated numerically. The QL derived from the numerical method is always smaller than the closed-form result, because some of the high-order coefficients were ignored in the numerical method due to the use of restricted matrix size.

Since P_{dyn} depends on the system utilization and task deadlines, there are upper bounds of system utilization and deadlines for any given value of P_{dyn}, denoted by P^*_{dyn}. For example, as shown in Fig. 2, both threshold patterns "1 2 3" and "2 4 5" failed to meet the specification if $P^*_{\text{dyn}} = 10^{-8}$ and deadline $D < 5$ with system load higher than 0.5. On the other hand, to ensure $P_{\text{dyn}} \leq P^*_{\text{dyn}}$

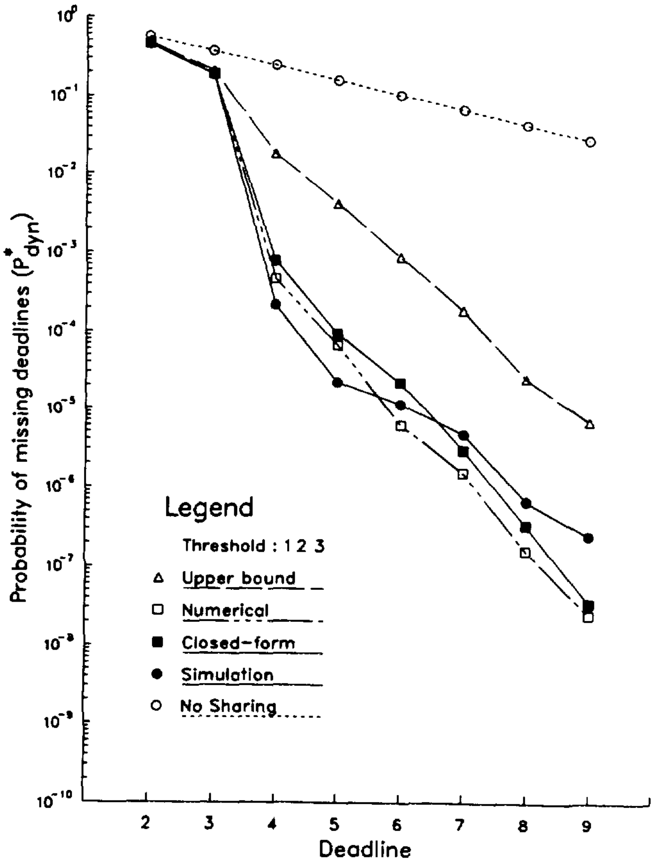

FIG. 1. Comparison of P_{dyn} derived from different solution methods.

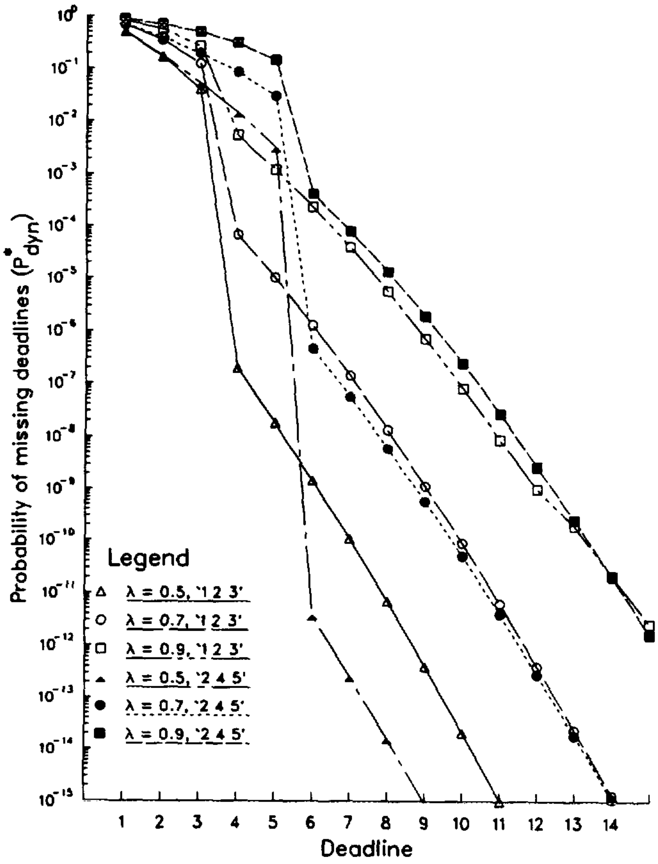

FIG. 2. P_{dyn} versus deadlines in different system loads and threshold patterns.

even with system load =0.9, the task deadlines must be greater than 12. Due to the nature of LSMSCB, minimizing P_{dyn} will increase the communication overhead. The communication cost, denoted as C_{com}, includes the overheads associated with task transfers and state-information collection. The cost of collecting state information is determined by the product of the frequency of state change (f_{sc}) and the size of buddy set (σ). The task transfer cost (β) can be controlled by adjusting TH_v; a higher TH_v will lower β. For example, minimizing C_{com} can be achieved by reducing the frequency of state change, the buddy set size, or the task transfer rate. The frequency of state change can be reduced by increasing the difference between TH_u and TH_f. A small buddy set size will reduce the cost for collecting state information, but will increase P_{dyn}. The task transfer rate can be reduced by increasing TH_v. However, P_{dyn} will generally increase with any attempt to reduce C_{com}. Thus, the primary goal in the design of an optimal LSMSCB is either to ensure the systems' P_{dyn} to be lower than P^*_{dyn} while minimizing the communication cost, or to minimize P_{dyn} while keeping the communication cost below a pre-specified level, C^*_{com}.

The problem of optimizing LSMSCB is formally stated as follows:

Optimal LSMSCB1. *Minimize* $C_{com} = \sigma f_{sc} + \beta$ subject to $P_{dyn} \le P^*_{dyn}$.

Optimal LSMSCB2. *Minimize* P_{dyn} subject to $C_{com} \le C^*_{com}$.

From [1], we know that f_{sc} is equal to the total number, T, of arriving tasks times the probability, P_{sc}, of state change. P_{dyn} can be computed as $\sum_{k=D}^{k_{max}} q_k$, where D is the given task deadline [1]. The closed-form equations can be used to derive the optimal threshold patterns and buddy set sizes for both the minimization problems.

Since P_{dyn} and C_{com} are determined by the threshold pattern and buddy set size used, the optimal threshold pattern and buddy set size can be found by an exhaustive search. In such a case, the search complexity is $O(D^3 N)$, where N is the total number of nodes and D is the given deadline. This complexity can be greatly reduced by utilizing the results in [1] as follows. First, the frequency of state change is 100% if $TH_f = TH_u$, and f_{sc} can be greatly reduced if $TH_f - TH_u \ge 1$, or $TH_f \ge TH_u + 1$.

Second, the number of unnecessary task transfers will increase if TH_f is close to TH_v, thus increasing the probability of missing task deadlines. This effect can be explained by the following example. Let $P^{(1)}_{dyn}$ and $P^{(2)}_{dyn}$ denote the probability of missing task deadlines under two threshold patterns with the same TH_u and TH_v, but $TH^{(1)}_f$ and $TH^{(2)}_f$, respectively. If $TH^{(1)}_f < TH^{(2)}_f < TH_v$ and $P^{(1)}_{dyn} < P^{(2)}_{dyn}$, then it is impossible to find a threshold pattern with the same TH_u and TH_v, but a TH_f greater than $TH^{(2)}_f$ that results in a lower P_{dyn} than $P^{(1)}_{dyn}$. In other words, the search for an optimal solution can skip all of the threshold patterns with such a TH_f.

Third, the effect of changing buddy set size on P_{dyn} is quite complicated due to the interaction between the nodes in a buddy set. As shown in Fig. 3, P_{dyn} continues to decrease with the increase of buddy set size when system load is 0.5, and P_{dyn} approaches a constant when system load is 0.7, but it may even increase with the increase of buddy set size when system load is higher than 0.9. Nevertheless, buddy sets of larger than 20 nodes will only reduce P_{dyn} infinitesimally and cannot offset the increasing cost of state-change broadcasts.

Based on the above observations, one can find the optimal threshold pattern and optimal buddy set size subject to the given D, P^*_{dyn}, and C^*_{com} by the following procedure. The first phase is to find a threshold pattern that satisfies the constraint. The search starts at $TH_u = 0$, $TH_f = TH_u + 1$, $TH_v = TH_f + 1$, and $\sigma = 20$ (or any other large number). If the resulting $P_{dyn} > P^*_{dyn}$, increase TH_v until it becomes equal to D. Then, increment TH_f or TH_u and restart the search until a pattern that satisfies the constraint is found. The next phase is either to minimize

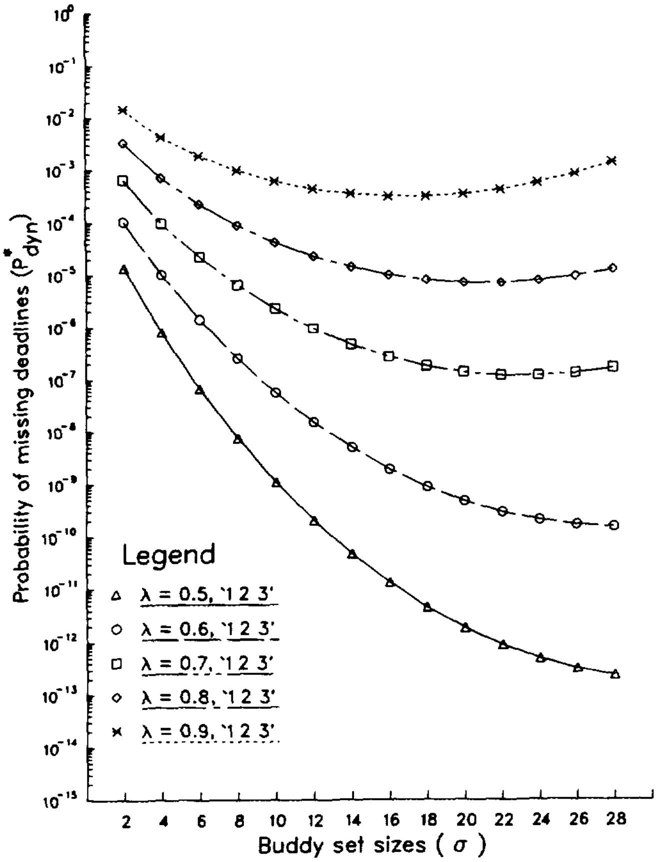

FIG. 3. P_{dyn} versus buddy set sizes (D = 6).

C_{com} for the optimal LSMSCB1, or to minimize P_{dyn} for the optimal LSMSCB2. The former is done by increasing the difference between TH_u and TH_f, or increasing TH_v, or reducing σ. Since any of these attempts will increase P_{dyn}, once the constraint cannot be satisfied, the minimizing procedure should stop. The optimal LSMSCB2 is solved by increasing TH_u and buddy set sizes until the constraint cannot be satisfied.

Many important and useful results are drawn from the above optimization process. First, the minimum achievable P_{dyn} with no constraint on C_{com} is given in Fig. 4. The results show that P_{dyn} can be reduced to below 10^{-15} when $D > 7$ and $\lambda = 0.8$. In most cases, P_{dyn} can be easily reduced to below 10^{-7}. The next figure shows the minimum achievable P_{dyn} subject to the constraint $C_{com} \leq C^*_{com}$. The cost shown in Fig. 5. is normalized to the total number of arrived tasks on a node, so $C_{com} = 1.0$ means that one control message will be generated per task arrival. Figure 6 shows the minimum achievable C_{com} while keeping $P_{dyn} \leq P^*_{dyn}$. When $\lambda = 0.5$ nd $D > 6$, LSMSCB can reduce C_{com} to below 1% while keeping $P_{dyn} \leq 10^{-6}$. Even when $\lambda = 0.8$ and $P^*_{dyn} = 10^{-10}$, C_{com} can still be reduced to below 0.1 if $D > 10$.

Another important result found in the derived optimal threshold pattern and buddy set is that the system utilization is significantly increased, as compared to the results calculated based on the upper bound model in [1]. For example, as shown in Fig. 7, the external task arrival rate is increased from 0.3, 0.06, 0.03 (from Fig. 9 of [1]) to 0.88, 0.7, 0.57 when D = 4 and $P_{dyn} = 10^{-4}, 10^{-6}, 10^{-8}$, respectively.

Some optimal solutions found for the various values of D and P^*_{dyn} are given in Table III. This table is useful in the design of real-time systems for the following reasons. First, the system can be easily checked if it can meet a given specification. The empty entries indicate the nonexistence of such a system. Second, a solution can always be found under any given D and P^*_{dyn} if such a solution exists. Third, any solution found from this table is optimal in the sense that the system constraint is satisfied and the communication cost is minimized.

5. CONCLUSION

Two main problems associated with load sharing in distributed real-time systems are addressed. First, an approximate closed-form expression for the distribution of queue length in the LSMSCB is derived. Second, using this distribution, optimal threshold patterns and optimal buddy set sizes, if any, can be found for any given dead-

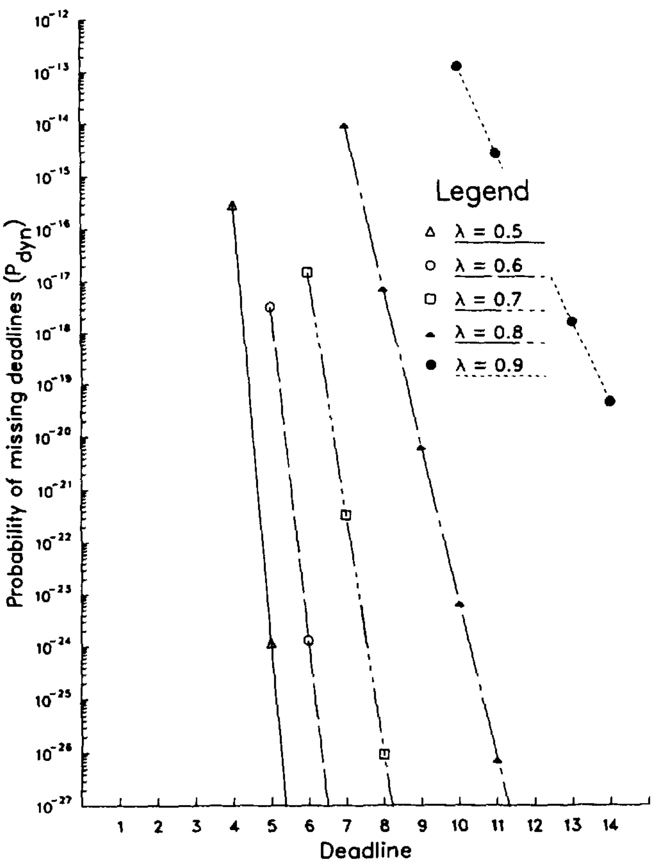

FIG. 4. Minimal P_{dyn} under no constraint on communication cost.

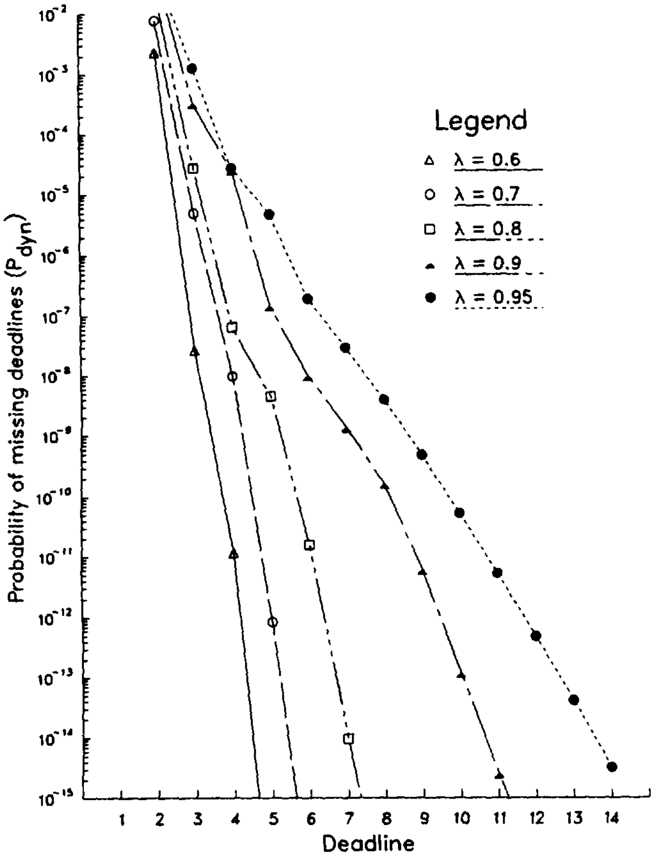

FIG. 5. Minimal P_{dyn} when $C_{com} \leq 1.0$.

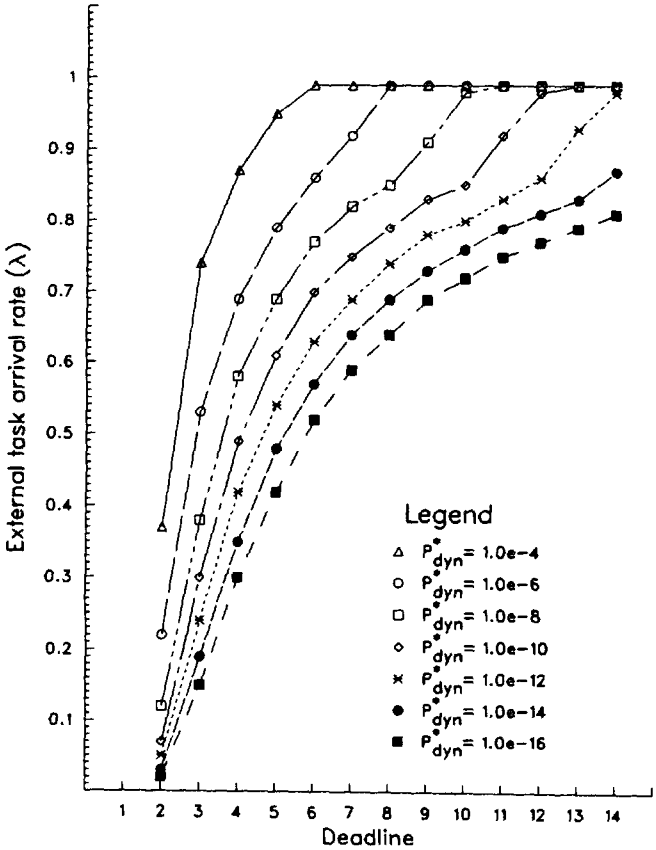

FIG. 7. System utilization versus deadlines.

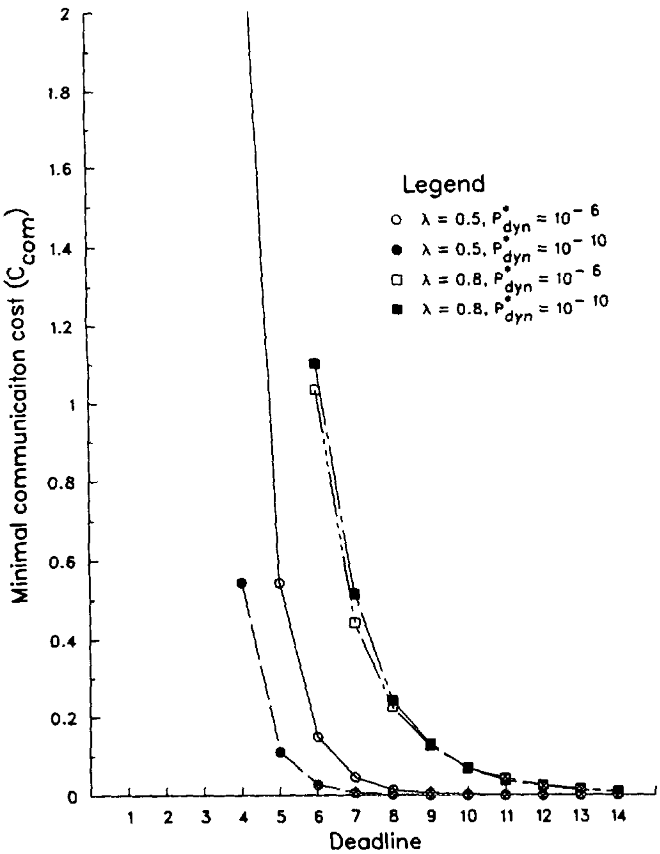

FIG. 6. Minimal C_{com} under different system loads and P^*_{dyn}.

TABLE III
Optimal Threshold Patterns and Buddy Set Sizes under Given D and P^*_{dyn}

	Load (λ)	0.5	0.7	0.9
P^*_{dyn}	Deadline	TH_u TH_f TH_v, σ	TH_u TH_f TH_v, σ	TH_u TH_f TH_v, σ
	≤2	—	—	—
	3	0 1 2, 10	—	—
	4	1 2 3, 5	2 3 4, 9	—
10^{-6}	5	1 2 3, 5	2 3 4, 7	2 3 4, 15
	6	2 4 5, 3	2 4 5, 6	3 4 5, 13
	7	2 5 6, 3	3 5 6, 6	3 5 6, 13
	8	4 6 7, 2	4 6 7, 5	4 5 6, 11
	≤3	—	—	—
	4	1 2 3, 9	1 2 3, 14	—
	5	1 3 4, 8	2 3 4, 12	—
10^{-8}	6	2 4 5, 6	2 4 5, 9	3 4 5, 17
	7	3 5 6, 5	4 5 6, 7	3 4 5, 14
	8	4 6 7, 5	4 6 7, 7	5 6 7, 14
	9	5 7 8, 4	5 7 8, 6	5 7 8, 13
	≤3	—	—	—
	4	1 2 3, 13	—	—
	5	1 3 4, 8	2 3 4, 16	—
10^{-10}	6	2 4 5, 7	2 4 5, 10	—
	7	4 5 6, 6	4 5 6, 9	—
	8	4 6 7, 6	5 6 7, 8	—
	9	5 7 8, 5	5 7 8, 8	6 7 8, 16

line and P^*_{dyn}. In most cases P_{dyn} can be reduced to below 10^{-9} at a reasonable cost for a wide range of system loads. hence, the LSMSCB enables a distributed real-time system to execute a large number of aperiodic tasks before their deadlines.

The LSMSCB reveals an interesting but contrary idea to many existing approaches which attempt to modify/generalize a scheduling algorithm for periodic tasks to the case of aperiodic tasks. Instead of trying to design a complex scheduling algorithm to guarantee aperiodic tasks, task arrivals at each node are processed on a FCFS basis in the LSMSCB, thus making the problem of local scheduling trivial. However, if a node cannot guarantee some tasks, the node will transfer these tasks to the nodes in its buddy set; task deadlines can thus be met by using the "combined" processing power of all nodes in the system, as opposed to using only local nodes. Both simulation and analytical results show that the LSMSCB is very efficient for meeting the deadlines of aperiodic tasks in distributed real-time systems.

REFERENCES

1. Shin, K. G., and Chang, Y.-C. Load sharing in distributed real-time systems with state change broadcsts. *IEEE Trans. Comput.* **C-38,** 8 (August 1989), 1124–1142.
2. Shin, K. G., Krishna, C. M., and Lee, Y.-H. A unified method for evaluating real-time computer controllers and its application. *IEEE Trans. Automat. Control* **AC-30,** 4 (April 1985), 357–366.
3. Leinbaugh, D. W. Guaranteed response times in a hard-real-time environment. *IEEE Trans. Software engrg.* **SE-6,** 1 (January 1980), 85–93.
4. Krishna, C. M., Shin, K. G., and Bhandari, I. S. Processor tradeoffs in distributed real-time systems. *IEEE Trans. Comput.* **C-36,** 9 (September 1987), 1030–1040.
5. Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard real-time environment. *J. Assoc. Comput. Mach.* **20,** (January 1973), 46–61.
6. Graham, R. L., *et al.*, Optimization and approximation in deterministic sequencing and scheduling: A survey. *Ann. Discrete math.* (1979), 287–326.
7. Kasahara, H., and Narita, S. Practical multiprocessor scheduling algorithms for efficient parallel processing. *IEEE Trans. Comput.* **C-33,** 11 (November 1984), 1023–1029.
8. Stankovic, J. A., Ramamritham, K., and Cheng, S. Evaluation of a flexible task scheduling algorithm for distributed hard real-time systems. *IEEE Trans. Comput.* **C-34,** 12 (December 1985), 1130–1143.
9. Peng, D., and Shin, K. G. Modeling of concurrent task execution in a distributed system for real-time control. *IEEE Trans. Comput.* **C-36,** 4 (April 1987), 500–516.
10. Zhao, W., and Stankovic, K. R. J. A. Preemptive scheduling under time and resource constraints. *IEEE Trans. Comput.* **C-36,** 8 (August 1987), 949–960.
11. Liu, J. W. S., Lin, K. J., and Natarajan, S. Scheduling real-time, periodic jobs using imprecise results. *Proc., 8th Real-time System Symposium, 1987*, pp. 252–260.
12. Ramamritham, K., Stankovic, J. A., and Zhao, W. Distributed scheduling of tasks with deadlines and resource requirements. *IEEE Trans. Comput.* **C-38,** 8 (August 1989), 1110–1123.
13. Thambidurai, P., and Trivedi, K. S. Transient overloads in fault-tolerant real-time systems. *Proc. 10th Real-time System Symposium, 1989*, pp. 126–133.
14. Lehoczky, J. P., Sha, L., and Strosnider, J. K. Enhanced aperiodic responsiveness in hard real-time environments. *Proc. 8th Real-time System Symposium, 1987*, pp. 261–270.
15. Farber, D. J., Feldman, J., Heinrich, F. R., Hopwood, M. D., Larson, K. C., Loomis, D. C., and Rowe, L. A. The distributed computing system. *IEEE COMPCON Spring, 1973*, pp. 31–34.
16. Smith, R. G. The contract net protocal: High-level communication and control in a distributed problem solver. *IEEE Trans. Comput.* **C-29,** 12 (December 1980), 1104–1113.
17. Hwang, K., Croft, W. J., Goble, G. H., Wah, B. W., Briggs, F. A., Simmons, W. R., and Coates, C. L. A unix-based local computer network with load balancing. *IEEE Trans. Comput.* **C-15,** 4 (April 1982) 55–64.
18. Ni, L. M., Xu, C. W., and Gendreau, T. B. A distributed drafting algorithm for load balancing. *IEEE Trans. Software Engrg.* **SE-11,** 10 (October 1985), 1153–1161.
19. Shin, K. G., and Chang, Y.-C. Coordinated load sharing in hypercube multicomputers, submitted for publication, 1991.
20. Shin, K. G., and Hou, C.-J. Analytic models of adaptive load sharing schemes in distributed real-time systems, 1992, *IEEE Trans. Parallel Distributed Systems* (June, 1993).
21. Shin, K. G., and Chang, Y.-C. Load sharing in hypercube multicomputers for real-time applications. *Proc. 4th Conf. on Hypercube Concurrent Computers and Applications, March 1989*, pp. 617–621.
22. Pease, M. C. *Methods of Matrix Algebra*, Academic Press, New York/London, 1965.

YI-CHIEH CHANG 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. In 1991 he received his Ph.D. degree in Electrical Engineering and Computer Science from the University of Michigan, Ann Arbor, Michigan. He is currently an assistant professor in the Electrical Engineering Department at the University of Texas at El Paso. He has published more than 10 technical papers in the areas of reconfigurable fault-tolerant array processors and distributed real-time systems. His current research interests include computer and architecture, VLSI systems, parallel processing, distributed and real-time systems.

KANG G. SHIN is a Professor and Chair of the Computer Science and Engineering Division, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, Michigan. He has authored/coauthored over 190 technical papers (about 90 of these in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, and robotics and automation. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he also received the Research Excellence Award from The University of Michigan. 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. He received the B.S. degree in Electronics Engineering from Seoul National University, Seoul, Korea in 1970, and both the M.S. and Ph.D degrees

in Electrical Engineering from Cornell University, Ithaca, New York in 1976 and 1978, respectively. From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at U.C. Berkeley, and International Computer Science Institute, Berkeley, CA. He is an IEEE fellow, was the Program Chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the General Chairman of the 1987 RTSS, the Guest Editor of the 1987 August special issue of *IEEE Transactions on Computers* on Real-Time Systems, and is a Program Co-Chair for the 1992 *International Conference on Parallel Processing*. He currently chairs the IEEE Technical Committee on Real-Time Systems, is a Distinguished Visitor of the Computer Society of the IEEE, an editor of *IEEE Transactions on Parallel and Distributed Computing*, and an area editor of *International Journal of Time-Critical Computing Systems*.

Received February 26, 1991; revised February 19, 1992; accepted July 16, 1992