Analytic Models of Adaptive Load Sharing Schemes in Distributed Real-Time Systems

Kang G. Shin, *Fellow, IEEE*, and Chao-Ju Hou, *Student Member, IEEE*

***Abstract*—In a distributed real-time system, nonuniform task arrivals may temporarily overload some nodes while leaving some other nodes idle. As a result, some of the tasks on an overloaded node may miss their deadlines even if the overall system has the capacity to meet the deadlines of all tasks. In a companion paper [1], we proposed, without any modeling analysis, a decentralized, dynamic load sharing (LS) scheme as a solution to this problem. In this paper, we develop analytic queueing models to comparatively evaluate the proposed LS scheme as well as three other schemes: no LS, LS with random selection of a receiver node, and LS with perfect information.**

The evolution of a node's load state is modeled as a continuous-time semi-Markov process, where cumulative execution time (CET), rather than the commonly-used queue length (QL), is employed to describe the workload of a node. Not only fundamental differences among the different LS schemes are addressed in the analytic models, but also implementation overheads are taken into account. Several metrics relevant to real-time performance are derived from these models: in particular, we evaluate the probability of a task missing its deadline, called the *probability of dynamic failure*. The proposed scheme is compared against other LS schemes using these performance metrics.

The validity of analytic models is checked with simulations. Both analytic and simulation results indicate that by using judicious exchange/use of state information and Bayesian decision mechanism, the proposed scheme makes a significant improvement over other existing LS schemes in minimizing the probability of dynamic failure.

***Index Terms*— Bayesian analysis, continuous-time Markov chains, deadlines, distributed real-time systems, load sharing, location and transfer policies, performance evaluation, random probing and selection.**

I. Introduction

DISTRIBUTED computing systems have long received considerable attention mainly due to their potential for high-performance and high-reliability, and the availability of inexpensive, powerful processors/memory chips. The central issue in realizing this potential is how to schedule the use of various resources in the system. Load sharing (LS) is known to be an essential element of scheduling tasks in a distributed system [2], [3], enabling underloaded/idle nodes to share the loads of overloaded ones so as to improve system performance.

Manuscript received January 15, 1991; revised August 20, 1992. This work was supported in part by the Office of Naval Research (ONR) under Contract N000114-85-K-0122 and the National Science Foundation (NSF) under Grant DMC-8721492. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views 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, MI 48109–2122.

IEEE Log Number 9209552.

The main goal of LS in a distributed real-time system is to minimize the probability of a task failing to complete execution before its deadline, which was termed the *probability of dynamic failure* (P_{dyn}) in [4]–[6]. Upon arrival of a real-time task, each node determines first whether or not it can complete this task in time. If it can, the task is executed locally; otherwise, some other "capable node" will be chosen to execute the task [3], [7]–[11]. By "capable node," we mean a node which has sufficient resource surplus to complete transferred-in task(s) in time.

To determine when and where to transfer a task, all dynamic/adaptive LS approaches need to use information on the workload of other nodes. If the cumulative execution time (CET) of a node is less than, or equal to, the laxity[1] of a newly arrived task, then the node can guarantee the task. Each node makes a decision on where to send a locally unguaranteed task using the state information collected by either periodic exchange of states [10]–[14], bidding/state probing at the time of making a LS decision [2], [15]–[23], or state-change broadcasts [1], [24]–[26]. No matter which strategy is used for collecting state information, the information may become out-of-date at the time of using it due to the delays in collecting it. That is, what a node *observes* about other nodes' states might be different from their *true* states at the time of making LS decisions. This inconsistency may cause performance degradation which is often ignored or assumed tolerable in previous research except for [22], [27], where the authors analyzed the effects of communication delays on the mean response time of dynamic LS, but did not propose how to deal with it.

To alleviate the negative effects of the delay in collecting state information and transferring tasks, Shin and Chang proposed a LS method based on state-change broadcasts in which each node needs to maintain state information of only a small set of nodes in its physical proximity, called a *buddy set* [25]. The buddy sets are systematically set up so as to overlap among some of them, thus allowing for system-wide (as opposed to only local) load sharing while significantly reducing the overhead in collecting state information. Three thresholds—based on queue length (QL) and denoted by TH_u, TH_f, and TH_v—are used to define the load state of a node. A node is said to be *underloaded* if $QL \leq TH_u$, *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

[1]The laxity of a task is defined as the latest time a task must start execution in order to meet its deadline.

(completion) of tasks, it will broadcast its change of state to all the other nodes in its buddy set. Every node that receives this broadcast 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 then select, without probing other nodes, the first available node from its preferred list. The main focus in [25] was to develop the basic concepts of buddy sets and preferred lists, and derive an approximate performance figure using QL as the measure of each node's workload.

In a companion paper [1], we proposed, without any modeling analysis, a new decentralized, dynamic LS scheme that uses the CET of each node and combines the preferred lists, region-change broadcasts, and Bayesian analysis both to minimize the probability of dynamic failure and to alleviate the performance degradation caused by communication delays. In this paper, we shall develop analytic models for this scheme as well as three other schemes: no LS, LS with random selection, and LS with perfect information. Not only the fundamental differences among the transfer and location policies used by different schemes are addressed, but also the computation/communication overheads in implementing these schemes are included in the analytic models. By taking into account these overheads, the analytic models provide a means of assessing the *absolute* real-time performance of the schemes considered. We shall derive several performance metrics, such as P_{dyn}, task transfer-out ratio, and maximum system utilization. These metrics are then used to assess the proposed LS scheme against the other schemes.

The first step in developing the analytic LS models is to define the (load) state of a node. For ease of analysis, the number of tasks queued at each node, or QL, is often used as the node's state [3], [7], [8], [10], [17], [25], [28]. Performance analysis based on QL would be accurate only if all tasks have an identical, or identically-distributed, execution time, *and* the mean task response time is used as the performance metric. If task execution times are neither identical nor identically-distributed, QL is no longer an adequate measure to characterize the load of a node. For real-time applications it is the CET, not QL, of a node that determines whether the node can guarantee a task or not. For example, a node with only a few tasks queued may not be able to guarantee a newly arrived task if a large amount of time is required to complete each queued task. On the other hand, a node with a large QL may still guarantee an arrived task as long as the total CET of that node does not exceed the laxity of this task.

Most LS schemes known to date are concerned with minimizing the mean response time (MRT) for general-purpose distributed systems, except for those in [13], [18], [21], [25], and [29], where the LS algorithms were evaluated with respect to either the percentage of tasks lost or the probability of dynamic failure. The performance of real-time LS algorithms is usually evaluated via simulations except for a few cases. For example, Shin and Chang [25] proposed an embedded Markov chain model, where QL, instead of CET, is used as the state of a node, but the exact solution to this model is very difficult to obtain. In [18], performance models were developed using CET as a node's state, but all tasks are assumed to have an identical deadline.

By contrast, we shall in this paper use P_{dyn} as the performance metric, and CET as the load state, and allow both task laxity and task execution time to be drawn from different probability distributions.

Most previous work has shown that simple LS algorithms can significantly reduce MRT for general-purpose systems, and the incremental benefits of employing complex LS algorithms become insignificant due to their communication/computation overheads. Using the fraction of tasks lost as the performance metric, Kurose *et al.* [18] extended this result to soft real-time systems. However, as both the analytic and simulation results indicate, this extension does not necessarily hold when P_{dyn} is used as the performance metric. By making judicious exchange/use of state information, complex schemes—though they incur more computation/communication overheads—achieve notable improvement in reducing P_{dyn} over those simple LS schemes.

The rest of this paper is organized as follows. Section II describes the system model used and details the operations of the proposed LS scheme and five other schemes, particularly focusing on the transfer policies that a node uses to handle locally unguaranteed tasks. Section III presents a mathematical model based on continuous-time semi-Markov chains that describes the state evolution of a node under different LS schemes. A two-step iterative algorithm used to solve the queueing model for different schemes is also given in Section III. The computation and communication overheads incurred in implementing the proposed LS scheme are dealt with in Section IV. In Section V, we derive several performance metrics, such as CET distributions, task transfer-out ratios, and P_{dyn} for the schemes under consideration, and comparatively evaluate the performance of the proposed LS scheme with these derived metrics and simulations. The paper concludes with Section VI.

II. System Model and LS Schemes

A. System Model

The nodes of a distributed system are assumed to be "homogeneous" in the sense that all nodes have the same arrival rate of external tasks[2] and are identical in processing capability and speed. Consequently, the task arrival/transfer activities experienced by each node are stochastically identical over a long term. Thus, we can adopt the general methodology—introduced in [3], and also used in [18], [25], and [30]—of first modeling the state (CET) evolution of a single node in isolation and then combining the node-level models into a system-level model. This decomposition was first verified (through simulation) in [3] to be valid for homogeneous systems of reasonably large size. We will also check its validity in Section V by comparing the analytic results with the results obtained from event-driven simulations.

External tasks (excluding transferred-in tasks) are assumed to arrive locally at node k according to a Poisson process with

[2]These exclude transfer-in tasks.

rate $\lambda_k = \lambda \ \forall k$. A task requires i units of time to execute and has j units of laxity time with probability q_{ij}, $1 \leq i \leq m$, $0 \leq j \leq T_m$, where m and T_m (measured in number of time units) are the largest task execution time and the largest task laxity in the system, respectively. $\{q_i = \sum_{j=0}^{T_m} q_{ij},\ 1 \leq i \leq m\}$ and $\{\hat{q}_j = \sum_{i=1}^{m} q_{ij},\ 0 \leq j \leq T_m\}$ are the probability distributions of task execution time and task laxity, respectively. All tasks are assumed to be independent of one another, so they do not communicate during their execution and thus have no precedence constraints among themselves. Note that aperiodic tasks in a real-time system are usually independent of each other. By contrast, periodic tasks often communicate with each other but their invocation, execution, and communication behaviors are usually known *a priori* and thus scheduled off-line.

B. LS Schemes Under Consideration

We shall develop models for the proposed LS scheme as well as three other schemes: no LS, LS with random selection, and LS with perfect information. Besides, LS with state probing and LS with focused addressing [17], [21] will be comparatively assessed in Section V using simulations. We shall describe first the operations of these schemes.

As was discussed in [3], a LS approach can be characterized by its transfer policy which determines *when* a task cannot be locally guaranteed, and its location policy which determines *where* a locally unguaranteed task should be transferred to. All LS schemes studied here employ the same transfer policy: a task with laxity d is transferred from node i if and only if node i's CET is greater than d. This transfer policy is of the threshold type as in [3], [8], [27], and [31], except that the threshold may change dynamically with the current state of the node and the time constraints of queued tasks. However, the location policies with which a node treats locally unguaranteed tasks are different as follows:

The noncooperative scheme: does not have LS capability and is used as a baseline in our analysis.

The quasi-perfect LS scheme: is used as another baseline where each node uses the same transfer policy but has complete information on the workload of other nodes without any overheads in collecting it.[3]

The random selection scheme: each locally unguaranteed task is sent to a randomly selected node. LS with random selection is simple to implement, requires no information exchange, and incurs little communication overhead. However, excessive task transfers may result since each unguaranteed task is sent blindly, without using any state information, to an arbitrarily chosen node.

The state probing scheme: a node with an unguaranteed task randomly probes up to some predetermined number of nodes and transfers the task to the first capable node found during the probing. LS with state probing gathers and uses the state information as needed, and thus may have the most up-to-date observation about other nodes. However, at least two additional messages are generated per probing, introducing time and communication overheads to the task to be transferred, and may thus be undesirable to the timely completion of real-time tasks. (Note that these overheads do occur when a task needs to be transferred.) Moreover, as was analyzed in [27] and [22], the performance of state probing is sensitive to the variation of communication delays.

The focused addressing scheme: each node exchanges state information periodically. A node sends its unguaranteed task to a node (called the *focused* node) which is randomly selected among those nodes "seen" to be capable of guaranteeing the task. (If such a capable node does not exist, the focused node is the node itself.) Meanwhile, the node also sends request-for-bid (RFB) messages to all the other nodes in the system, indicating that bids (which contains the CET of the bidding node) should be returned to the designated focused node. If the focused node cannot guarantee the task, it chooses, based on the bids received, a capable node for transferring the task (ties are broken randomly); otherwise, the task is queued on the focused node. The bids received at the focused node are also used to update the observation of other nodes' states. If neither the focused node nor the bidding nodes can guarantee the task, the task is declared to be lost and thrown out.

To avoid poor CPU utilization, RFB messages do not require nodes to reserve CPU cycles or any other resources needed to execute the task to be transferred. When a task arrives at a node whose bid has been accepted, the node will check again whether or not the task can be guaranteed. Note that this is a simplified version of the scheme proposed in [21] and [17]. It also differs slightly from that of [21] and [17] in the way a node chooses the focused node. The authors of [21] and [17] used the percentage of free time during the next window (which is a design parameter) and many other estimated parameters to determine the focused node or the node to which the task must be transferred again. However, we use the observed CET of other nodes to determine the nodes for transferring tasks.

The proposed scheme: a node transfers each locally unguaranteed task to one of the nodes in its buddy set by combining the preferred lists, state-region change broadcasts, and Bayesian analysis. This scheme was proposed in [1], without any performance modeling analysis, to i) minimize P_{dyn}, and ii) alleviate the performance degradation caused by communication delays.

The operation of the proposed LS scheme is outlined below for completeness. (See [1] for a detailed account.) Under this scheme, the task scheduler at each node is modeled as a Bayesian decision maker [32]: upon arrival of a real-time task, the scheduler first determines whether the node can guarantee the task or not. If it cannot, the scheduler on that node looks up the list of *loss-minimizing decisions*, and choose—based on the current observation about other nodes' states, $\underline{X}$—the best candidate receiver in a small set of n nodes in its physical proximity,[4] i.e., a buddy set. The list of loss-minimizing decisions is computed/updated periodically to minimize the expected Bayesian loss with respect to the posterior distribution of system state given the observation.

[3]This, however, cannot be modeled as an M/G/n queue, as compared to the perfect load sharing in [3]. This is because of the transfer policy used which incorporates the consideration of task laxities/deadlines into the LS decision. Hence, we label this scheme as quasi-perfect.

[4]For example, those nodes one or two hops away from the node of interest.

```
Each node performs the following four operations:

(a): When a task with execution time $T_i$ and laxity $D_i$ arrives:
    if current_time + CET $\geq D_i$ then
     begin
        receiver_node = table_lookup($\underline{x}$:observation)†;
        transfer the task to receiver_node;
     end
    else
     begin
        CET := CET + $T_i$;
        if CET crosses $TH_{2k}$, $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$ then
            /* $TH_1, \ldots, TH_{K-1}$ are thresholds */
            broadcast the state-region change to all nodes in its buddy set;
        queue the task locally;
     end

(b): When a message broadcast by node $i$, $1 \leq i \leq n$, arrives:
    update observation $x_i$;
    record the (observation, true state) pair needed for constructing probability
        distributions;

(c): At every clock tick,
    CET := CET - 1;
    if CET crosses $TH_{2k}$, $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$ then
        broadcast the state-region change to all nodes in its buddy set;

(d): At every $T_p$ clock ticks,
    update the probability distributions and the table of loss-minimizing
        decisions;
```

†If a node anticipates, based on the current observation $\underline{x}$, that no other nodes can guarantee the task, this task is declared to be lost and thrown away.

Fig. 1. Operations of the task scheduler on each node.

Fig. 1 summarizes the four (4) main operations of the task scheduler on each node.

Both the posterior distribution of system state given the node's observation and the list of loss-minimizing decisions are constructed/updated as follows. Each node communicates with, maintains the state information of, and/or transfers unguaranteed tasks to, the nodes in its buddy set *only*. K state regions defined by $(K-1)$ thresholds, $TH_1, TH_2, \cdots, TH_{K-1}$, are used to characterize the workload of each node. Each node will broadcast a *time-stamped* message, informing all the other nodes in its buddy set of a state-region change whenever its load crosses TH_{2k} for some k, where $1 \leq k \leq \lceil K/2 \rceil - 1$.[5] This time-stamped message contains node number i, state ω_i, and the time t_0 when this message is sent. When the message broadcast by node i arrives at node j, node i's state at t_0, denoted by ω_i, can be recovered by node j. Node j can also trace back to find its observation about node i, x_i, at time t_0. This observation x_i is what node j thought (observed) about node i when node i was actually in state ω_i. x_i's along with ω_i's $(1 \leq i \leq n)$ are used by node j to compute/update the posterior distribution, $P_{W_i|X_i}$, of node i's state W_i given node j's observation X_i periodically. Any inconsistency between the true state, W_i, and the state observed by node j, X_i, is characterized by this probability distribution. Besides, ω_i sent by node i at time t_0 is transformed to node j's new observation,[6] x_i, about node i *at the time node i receives this message* by the rule that $x_i = k$ if $TH_k \leq \omega_i < TH_{k+1}$, $k \geq 0$, and $TH_0 \overset{\Delta}{=} 0$.

For each possible $\underline{X} = \underline{x}$, and for each possible $T_d \in (0, T_{\max}]$—where $T_{\max}$ is the largest task laxity in the system—node j computes $P_{W_i|X_i}(W_i > T_d)$,[7] $i = 1, \cdots, n$ every time the posterior distribution is updated, and the node k which results in the smallest value is chosen as the receiver node, and recorded in the list of loss-minimizing decisions. When node j cannot guarantee locally a real-time task with laxity T_d, it decides to which other node this task will be transferred by looking up the list using $\underline{X}$ and T_d as the indexes.

A tie will be broken by using the preferred list: each node orders the nodes in its buddy set into a preferred list such that a node is the kth preferred node of *one and only one other* node, where k is some integer [25], [33], and a node chooses from the preferred list the first d_i with Bayesian risk. The preferred list thus provides an effective way of selecting a receiver among several possible candidate nodes

[5]The reason for not broadcasting the change of state region whenever a node's load crosses an odd-numbered threshold is to reduce the network traffic resulting from region-change broadcasts. On the other hand, the reason for not combining two adjacent state regions into one and then broadcasting the change of state region whenever a node's CET crosses any threshold is to include finer state information in each broadcast and thus construct more accurate posterior distributions needed for Bayesian decisions.

[6]The reason for transforming ω_i into x_i is to reduce the size of the observation space.

[7]which was shown in [1] to be the expected loss in the Bayesian decision model.

while minimizing the possibility of more than one node simultaneously sending unguaranteed tasks to the same node. The set of loss-minimizing decisions is a list of decisions indexed by each possible observation $\underline{x}$ and each possible task laxity T_d. Once these calculations are completed, the task scheduler needs only to perform a table lookup with both $\underline{x}$ and T_d as indexes to determine to which node a locally unguaranteed task should be transferred.

III. Analytic Models

Queueing models are developed to evaluate the performance of the proposed LS scheme as well as three other schemes: no LS, LS with random selection, LS with perfect information. We first model the state evolution of a node by a continuous time semi-Markov chain [34], [35] which will serve as the underlying model. The parameters of this model are derived for different LS schemes to characterize task arrival/transfer processes in the system level. A two-step iterative approach is then taken to obtain a numerical solution to the semi-Markov model.

A. The Underlying Model

The state of a node is defined as the CET of that node, and each node is modeled as an $M^{[x]}/D/1$ queue with bulk arrivals. (Arrival of a task with i units of execution time is viewed as the simultaneous arrival of i tasks, each requiring one unit of execution time.) The case in which all tasks require an identical execution time—and thus the state is QL—is a special case of this model.

The composite (both external and transferred) task arrival rate at a node is $\lambda(T)$, which depends on the node's CET, T, and the location/transfer policies used. Execution of a task requires i units of time with probability q_i, and such a task is called a *type-i task*, $1 \leq i \leq m$. Since at any time a node is either idle or busy executing a task, the node occupancy (by tasks) is divided into busy slots (measured in terms of system clock cycles) which are numbered as $B_1, B_2, \cdots$, relative to any reference point of time (see Fig. 2). Note that two adjacent busy slots of a node may be either contiguous or separated by idle periods. Let T_k denote a node's remaining CET at the end of B_k, and X_k^i represent the number of type-i arrivals during B_k, then

$$T_k = \begin{cases} T_{k-1} + \sum_{i=1}^m iX_k^i - 1 & \text{if } T_{k-1} > 0 \\ \sum_{i=1}^m iX_k^i + R - 1 & \text{if } T_{k-1} = 0 \end{cases} \tag{1}$$

where R is the number of clock cycles required for the node to complete a task which finds the node idle upon its arrival, and has the distribution $\{q_i, 1 \leq i \leq m\}$. If $T_{k-1} = 0$, then the node remains idle until a task with the required execution time, R, arrives. $\sum_{i=1}^m iX_k^i$ is the CET accrued during B_k with each type-i task contributing i units of time for execution. Equation (1) describes a semi-Markov chain because i) T_k depends only on T_{k-1}, and not on $T_{k'}$ $\forall k' < k-1$, and ii) the state residence times—the length of B_k—are deterministic with value 1, rather than exponentially-distributed.

Equation (1) can be used to get the z-transform of the CET distribution. Let T^+ and T denote the CET on a node at some *embedding* time instant and at some *random* time instant, respectively. Because of the embedding points (i.e., at the end of each busy slot) chosen for the embedded Markov chain, the distribution of T, denoted by $p_T(.)$, is not necessarily the same as that of T^+, denoted by $p_T^+(.)$ [35], [36]. However, $p_T(.)$ can be derived from $p_T^+(.)$ as described in [36]. So, let us derive $p_T^+(.)$ first.

Let $\Phi^+(z)$ denote the z-transform of CET distribution at the embedding time points, then

$$\Phi^+(z) = \sum_{n=0}^{\infty} p_T^+(n) z^n = E(z^{T^+})$$

where $p_T^+(i) \triangleq Pr(T^+ = i)$. Let $\delta(x)$ be the unit step function, then

$$T_k = T_{k-1} + \sum_{i=1}^{m} iX_k^i - 1 + (1 - \delta(T_{k-1}))R,$$

and

$$\begin{aligned} \Phi^+(z) &= E(z^{T^+}) = E(z^{T_k}) \\ &= E(z^{T_{k-1} + \sum_{i=1}^m iX_k^i + (1-\delta(T_{k-1}))R - 1}) \\ &= E(z^{T_{k-1} + (1-\delta(T_{k-1}))R-1}) E(z^{\sum_{i=1}^m iX_k^i}). \end{aligned} \tag{2}$$

Note that for mathematical tractability of (2), $\sum_{i=1}^m iX_k^i$—the CET arrival process during B_k—is approximated to be independent of T_{k-1}, which is unrealistic for the location and transfer policies of the proposed LS scheme as well as others. As will be discussed in Section V-B, this deficiency is remedied by figuring the dependency of the task arrival process on CET into task arrival rate. In other words, the task arrival rate λ on a node is determined by the node's CET, T, i.e., $\lambda(.)$ is a function of T.

The second factor of (2), $E(z^{\sum_{i=1}^m iX_k^i})$, is computed as follows. Since state residence times are all 1, we have

$$P(iX_k^i = in) = \frac{e^{q_i\lambda}(q_i\lambda)^n}{n!}$$

and

$$P(iX_k^i = \ell) = \begin{cases} \frac{e^{q_i\lambda}(q_i\lambda)^{\ell/i}}{(\ell/i)!} & \text{if } (\ell/i) \in \boldsymbol{N} \cup \{0\} \\ 0 & \text{otherwise} \end{cases}$$

where $1 \leq i \leq m$, and $\boldsymbol{N}$ is the set of natural numbers. In the above equation we used the fact that if each event of a Poisson process is classified independently of others to be any of type $1, 2, \cdots, m$ with probability q_i, then the number of type-i arrivals is independent of others, and is Poisson-distributed with λq_i. Let $U_k^i \triangleq iX_k^i$, then

$$\begin{aligned} \Phi_{U_k^i}(z) &= \sum_{\ell=0}^{\infty} z^\ell P(U_k^i = \ell) \\ &= \sum_{\ell=0}^{\infty} z^{i\ell} P(U_k^i = i\ell) \\ &= \sum_{\ell=0}^{\infty} z^{i\ell} \frac{e^{q_i\lambda}(q_i\lambda)^\ell}{\ell!} \\ &= e^{q_i\lambda(z^i - 1)}. \end{aligned} \tag{3}$$

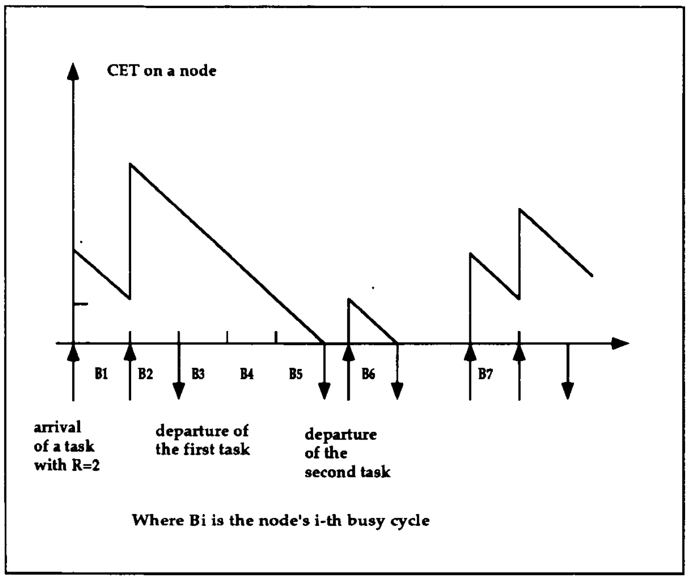


Fig. 2. A sample path for the evolution of remaining CET on a node.

Assuming independent arrivals of different types of tasks, we then have

$$\begin{aligned}\Phi_{\sum_{i=1}^{m} U_k^i}(z) &= \prod_{i=1}^{m} \Phi_{U_k^i}(z)\\ &= \prod_{i=1}^{m} e^{q_i\lambda(z^i-1)}\\ &= e^{\lambda(\sum_{i=1}^{m} q_i z^i - 1)}\\ &= e^{\lambda(\Phi_R(z)-1)} \qquad (4)\end{aligned}$$

where $\Phi_R(z)$ is the z-transform of the required task execution time.

We now compute the first factor of (2). The event, $\{T_{k-1} - 1 + (1 - \delta(T_{k-1}))R = i\}$, contains two mutually exclusive subevents: 1) $T_{k-1} = 0$ and $R = i + 1$, and 2) $T_{k-1} = i + 1$:

$$P(T_{k-1} - 1 + (1 - \delta(T_{k-1}))R = i) = q_{i+1}p_T^+(0) + p_T^+(i+1),$$

and thus,

$$\begin{aligned}&E(z^{T_{k-1}-1+(1-\delta(T_{k-1}))R})\\ &= \sum_{n=0}^{\infty} P(T_{k-1} - 1 + (1 - \delta(T_{k-1}))R = n)z^n\\ &= \sum_{n=0}^{\infty}(q_{n+1}p_T^+(0) + p_T^+(n+1))z^n\\ &= \frac{1}{z}p_T^+(0)\Phi_R(z) + \frac{1}{z}(\Phi(z) - p_T^+(0)). \qquad (5)\end{aligned}$$

Substituting (4) and (5) into (2), and rearranging the terms, we get

$$\Phi^+(z) = \frac{\frac{1}{z}p_T^+(0)(\Phi_R(z) - 1)e^{\lambda(\Phi_R(z)-1)}}{1 - \frac{1}{z}e^{\lambda(\Phi_R(z)-1)}}. \qquad (6)$$

$p_T^+(0)$ can be obtained from

$$\begin{aligned}1 &= \Phi^+(1)\\ &= p_T^+(0)\lim_{z\to 1}\frac{\frac{1}{z}(\Phi_R(z) - 1)e^{\lambda(\Phi_R(z)-1)}}{1 - (1/z)e^{\lambda(\Phi_R(z)-1)}}\\ &= p_T^+(0)\frac{E(R)}{1 - \lambda E(R)}\end{aligned}$$

where we have used L'Hopital's rule in evaluating the limit. Consequently,

$$\begin{aligned}p_T^+(0) &= \frac{1 - \lambda E(R)}{E(R)}\\ &= (1/E(R)) - \lambda \qquad (7)\end{aligned}$$

where $E(R)$ is the expected execution time, i.e., $E(R) = \sum_{i=1}^{m} iq_i$. Note that for the system to be stable (or for the CET at a node not to grow unboundedly), we must have $\sum_{i=1}^{m} i\lambda q_i \le 1$, or $\lambda \le (1/E(R))$, which is the necessary and sufficient condition for $p_T^+(0) \ge 0$. Thus,

$$\Phi^+(z) = \frac{(\frac{1}{E(R)} - \lambda)(\Phi_R(z) - 1)e^{\lambda(\Phi_R(z)-1)}}{z - e^{\lambda(\Phi_R(z)-1)}}. \qquad (8)$$

The above results for the embedded Markov chain do not directly apply to the total general-time stochastic process. However, the relation between the general-time steady-state distribution, $p_T(.)$, and the distribution at embedding points, $p_T^+(.)$, is shown in [36] to be

$$\Phi(z) = \frac{E(R)(z-1)}{\Phi_R(z) - 1}\Phi^+(z), \qquad (9)$$

for $M^{[x]}/G/1$ systems, where $\Phi(z)$ is the z-transform of the general-time CET distribution. Thus, we have

$$\Phi(z) = \frac{(1 - \lambda E(R))(z-1)e^{\lambda(\Phi_R(z)-1)}}{z - e^{\lambda(\Phi_R(z)-1)}} \qquad (10)$$

and

$$p_T(0) = 1 - \lambda E(R). \tag{11}$$

If all tasks require an identical execution time, i.e., $P(R = 1) = 1$, then the state reduces to QL, and (11) reduces to

$$p_T(0) = 1 - \lambda$$

which is exactly the utilization $U = \lambda$ since the service time is 1, and (10) [and also (8)] reduces to

$$\Phi(z) = \frac{(1-\lambda)(z-1)e^{\lambda(z-1)}}{z - e^{\lambda(z-1)}}. \tag{12}$$

Computing the inverse z-transform of (10) [(12)] numerically yields the CET (QL) distribution, $\{p_T(i), i \geq 0\}$. The discussion of a rather subtle technique for the inversion of (10) can be found in [36].

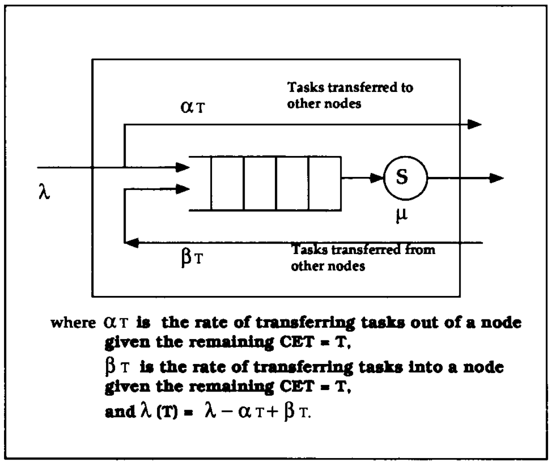
B. Derivation of $\lambda(T)$

The semi-Markov chain model derived above can be used to evaluate different LS schemes if $\lambda(T)$ characterizes both the corresponding task arrivals and/or task transfers in the system level. The following variables are necessary to facilitate the derivation of $\lambda(T)$:

- α_T: the rate of transferring tasks out of a node given that the node's remaining CET is T. Since the transfer policy determines whether or not a task can be guaranteed locally, this parameter characterizes the transfer policy used.
- β_T: the rate of transferring tasks *into* a node given that the node's remaining CET is T. This parameter corresponds to the location policy used, since the location policy determines where to send each unguaranteed task.
- γ_j: the probability that the remaining CET on a node is no less than j units of time, i.e., $\gamma_j = P(T \geq j)$.
- K_j: the number of nodes that can be chosen by a node, excluding itself, for transferring a task with laxity j. The distribution of K_j can be expressed in terms of γ_j as

$$P(K_j = \ell) = \binom{n-1}{\ell}(1-\gamma_{j+1})^{\ell}\gamma_{j+1}^{n-1-\ell}.$$

As shown in Fig. 3, $\lambda(T) = \lambda - \alpha_T + \beta_T$. By appropriately tailoring α_T and β_T to describe the transfer and location policies adopted and by approximating the combined (external and transferred-in) task arrivals at each node to follow a Poisson process, the above semi-Markov chain model can be used to express the operations of different LS schemes. This approximation is accurate only when 1) task-transfer out of each node is a Poisson process, implying that the decision on whether or not to transfer a task is independent of the current workload [37], which is not true for our LS scheme, and 2) task-transfer into each node—the superposition of task transfers out of other nodes—is Poisson. However, this approximation is shown by our simulation experiments to provide very good results; when the task transfer-out ratio is less than 40% of the task arrival rate, the combined task arrival processes at each node have the coefficients of variation of their interarrival times close to one, which is at least a


Tasks transferred to other nodes
α_T
λ
S
μ
β_T
Tasks transferred from other nodes

where α_T is the rate of transferring tasks out of a node given the remaining CET = T,
β_T is the rate of transferring tasks into a node given the remaining CET = T,
and $\lambda(T) = \lambda - \alpha_T + \beta_T$.

Fig. 3. A generic queueing model for each node.

necessary condition for the combined task arrival processes to be modeled as Poisson.

Moreover, for all LS schemes the following relationship between α_T and β_T results from the law of task conservation, $\sum_{k=0}^{\infty} \lambda(k)p_T(k) = \lambda$.

Theorem 1: If task flow of the system is conserved, then

$$\sum_{k=0}^{\infty} \alpha_k p_T(k) = \sum_{k=1}^{\infty} \beta_k p_T(k). \tag{13}$$

The transfer policy used in real-time systems is of the dynamic threshold type: a task is transferred to other nodes only if it cannot be guaranteed locally. Thus, given a node's remaining CET $= T$, all the tasks arrived with laxities smaller than T should be transferred, thus leading to

$$\alpha_T = \sum_{j=0}^{T-1} \lambda \hat{q}_j, \tag{14}$$

for all schemes except for no LS in which $\alpha_T = \beta_T = 0$, and $\lambda(T) = \lambda \ \forall T$.

Unlike the transfer policy, the location policy depends on each LS scheme. The random selection scheme selects randomly a receiver node in the system for each unguaranteed task without using any state information. Thus, we get β_T for this scheme as[8]

$$\beta_T = \frac{1}{N}\sum_{j=1}^{T_m}(N\gamma_{j+1})\lambda\hat{q}_j = \sum_{j=1}^{T_m}\gamma_{j+1}\lambda\hat{q}_j \tag{15}$$

where $N\gamma_{j+1}$ is the average number of nodes that cannot guarantee tasks with laxity j in an N-node system, $\lambda\hat{q}_j$ is the arrival rate of tasks with laxity j on a node, and the product of these two is the average rate of transferring tasks with laxity j in the system. Since all unguaranteed tasks are transferred randomly, each node shares $1/N$ of these tasks. The randomness property is reflected in the independence of β_T from T. The correctness of (15) is verified in the Appendix.

[8] β_T is derived under the assumption that the task-transfer into a node—the superposition of task-transfers out of other nodes—is Poisson-distributed.

Our proposed LS scheme uses the posterior distributions derived from the state information gathered from time-stamped region-change broadcasts to estimate the workload of other nodes, and chooses probabilistically, from the preferred list, the best candidate to which each unguaranteed task will be transferred. If these distributions are properly constructed, then β_T can be expressed as

$$\begin{aligned}\beta_T &= \sum_{j=T}^{T_m} \lambda \hat{q}_j \gamma_{j+1}(1+\gamma_{j+1}+\gamma_{j+1}^2+\cdots+\gamma_{j+1}^{n-1}) \\ &= \sum_{j=T}^{T_m} \lambda \hat{q}_j \gamma_{j+1} \frac{1-\gamma_{j+1}^n}{1-\gamma_{j+1}} \end{aligned} \tag{16}$$

where n is the number of nodes in a buddy set. Note that 1) a node i with CET $= T$ can guarantee all tasks with laxity greater than T, and thus, the summation is performed from T to T_m, and 2) the term $\lambda\hat{q}_j\gamma_{j+1}$ is contributed by the node whose most preferred node is node i, the term $\lambda\hat{q}_j\gamma_{j+1}^2$ is contributed by the node whose second preferred node is node i and whose most preferred node cannot guarantee tasks with laxity j, and the term $\lambda\hat{q}_j\gamma_{j+1}^n$ accounts for the situation when all nodes in a buddy set cannot guarantee tasks with laxity j. The correctness of (16) is verified in the Appendix.

Recall that a node is the kth preferred node of one and only one other node, and if node i is the kth preferred node of j, then j is also the kth preferred node of i [25], [33]. This property minimizes the possibility of multiple nodes simultaneously sending tasks to the same "capable" node, while ensuring unguaranteed tasks to be evenly shared by "capable" nodes. More formally, we have the following theorem:

Theorem 2: Using the preferred lists and proper prior/posterior distributions, our proposed scheme balances load in the sense that all unguaranteed tasks are evenly shared by those capable nodes.

Proof: This theorem is proved by deriving β_T based on the idea of even sharing of unguaranteed tasks among "capable" nodes and comparing the result with the β_T in (16). Even sharing of unguaranteed tasks gives

$$\beta_T = \sum_{j=T}^{T_m}\sum_{\ell=0}^{n-1} \frac{(n\gamma_{j+1})(\lambda\hat{q}_j)}{\ell+1} P(K_j=\ell)$$

where $\lambda\hat{q}_j$ is the arrival rate of tasks with laxity j, $n\gamma_{j+1}$ is the average number of nodes which cannot guarantee tasks with laxity j. The product of these two is the average rate of transferring tasks with laxity j, which will be shared evenly by ℓ other nodes (in addition to the node itself) with probability $P(K_j=\ell)$. β_T can be simplified to

$$\begin{aligned}\beta_T &= \sum_{j=T}^{T_m} \lambda\hat{q}_j\gamma_{j+1} \sum_{\ell=0}^{n-1} \frac{n}{\ell+1} P(K_j=\ell) \\ &= \sum_{j=T}^{T_m} \lambda\hat{q}_j\gamma_{j+1}\frac{1-\gamma_{j+1}^n}{1-\gamma_{j+1}}\end{aligned} \tag{17}$$

which is exactly the same as (16). □

The location policy of the quasi-perfect LS scheme is similar to that of our scheme except that i) accurate state information is obtained without incurring any communication cost, ii) there is no overhead associated with task transfers, and iii) the buddy set size is N, the number of nodes in the entire system. That is, unguaranteed tasks are transferred directly and instantaneously to "capable" nodes. $\lambda(T)$ can be expressed as

$$\begin{aligned}\lambda(T) &= \sum_{j=T}^{T_m}\sum_{\ell=0}^{N-1} \frac{N\lambda\hat{q}_j}{\ell+1} P(K_j=\ell) \\ &= \sum_{j=T}^{T_m} \lambda\hat{q}_j \frac{1-\gamma_{j+1}^N}{1-\gamma_{j+1}}.\end{aligned} \tag{18}$$

The correctness of (18) is also verified in the Appendix.

C. An Iterative Algorithm

$\lambda(T)$ (α_T and β_T) must be known before solving the Markov chain model for $p_T(.)$. However, $\lambda(T)$ depends on γ_j which in turn depends on $p_T(.)$. An iterative approach is taken to handle the difficulty associated with this recursion problem. Note that $\Phi_{\sum_{i=1}^m U_k^i}(z) = e^{\lambda(\Phi_R(z)-1)}$ [(4)] can be interpreted as the pdf of the number of arrivals[9] during one unit of service time (execution time). Thus, we modify (4) as

$$\Phi_{\sum_{i=1}^m U_k^i}(z) = \sum_{i=0}^{\infty} e^{\lambda(i)(\Phi_R(z)-1)} p_T(i),$$

to account for the effect that the task arrival rate varies with the current CET, T, of a node. Consequently, (10) is modified as

$$\Phi(z) = \frac{(1-\lambda E(R))(z-1)\sum_{i=0}^{\infty} e^{\lambda(i)(\Phi_R(z)-1)}p_T(i)}{z-\sum_{i=0}^{\infty} e^{\lambda(i)(\Phi_R(z)-1)}p_T(i)},$$

and (12) as

$$\Phi(z) = \frac{(1-\lambda)(z-1)\sum_{i=0}^{\infty} e^{\lambda(i)(z-1)}p_T(i)}{z-\sum_{i=0}^{\infty} e^{\lambda(i)(z-1)}p_T(i)}.$$

In the first step, the modified version of (10) [(12)] is solved for $p_T(.)$ with both α_T and β_T set to 0, or equivalently, $\lambda_T = \lambda$ $\forall T$. The resulting $p_T(.)$ is used to compute α_T and β_T in the second step. Then $p_T(.)$ is recalculated with the new α_T and β_T [and thus a new $\lambda(T)$] using the modified version of (4) and (10). This result will, in turn, change α_T and β_T. This procedure will repeat until $p_T(.)$ and $\lambda(T)$ converge to some fixed values.

IV. Computation/Communication Overheads

To develop a practical model for assessing the performance of different LS schemes, one should, in addition to addressing the fundamental differences among the LS schemes, take into account their implementation overheads: for example, the computational overheads of our scheme due to probability updates, and communication delays associated with state-information collection and task transfers. The tradeoff between the associated complexity and the resulting benefit can be

[9]As mentioned earlier, a type-i task arrival is viewed as i simultaneous arrivals, each with one unit of execution time.

analyzed accurately only if implementation overheads are included in the model. In this section, our model is extended to include the overheads of the proposed scheme due to *time-stamped region-change broadcasts*, *periodic updates of posterior distributions*, and *task transfers* by:

- Augmenting the original task set with a new type of tasks, i.e., the probability updating tasks.
- Modifying the underlying semi-Markov chain model to include the effect of region-change broadcasts on the CET of a node.
- Considering the effect of communication delay by modifying α_T and β_T.

A. Overheads of Probability Updates

As mentioned in Section II, each node updates the posterior distributions of other nodes' CET once every T_p units of time. Let u be the time required for updating the distributions, then we introduce a new type of task by modifying the parameters λ and q_{ij} (which characterize the task set) as

$$\lambda' = \lambda + \frac{1}{T_p}, \quad (\lambda + \frac{1}{T_p})q'_{uT_m} = \frac{1}{T_p},$$

and

$$(\lambda + \frac{1}{T_p})q'_{ij} = \lambda q_{ij}$$

where q'_{ij} and q_{ij} (λ' and λ) are the new and old values of the probabilities (task arrival rates), respectively. That is, a new type of tasks is added to the task set: the one with execution time u and laxity T_m. Moreover, q_{ij}'s are scaled in accordance with the new task arrival rate λ'. Note that 1) the laxity of the probability updating task is chosen to be T_m, since this task is not time-critical, and 2) for ease of analysis, this periodic task is modeled to have exponential time-to-event distributions with the rates equal to the reciprocal of the period. The error of this approximation relative to the exact limit-equivalent rates is bounded as indicated by Kitchin [38].

B. Overheads of Region-Change Broadcasts

Recall that each node broadcasts the change of state region to all the other nodes in its buddy set. Let a be the time needed for broadcasting a region-change. Since this broadcasting process is state-dependent, the overheads of region-change broadcasts can be included by modifying (1), the expression for state evolution as

$$T_k = \begin{cases} T_{k-1} + \sum_{i=1}^{m} iX_k^i - 1 + a \\ \quad \text{if } T_{k-1} > 0 \text{ and } T_{k-1} \in \{TH_{2\ell}, 1 \le \ell \le \lceil \frac{K}{2} \rceil - 1\} \\ T_{k-1} + \sum_{i=1}^{m} iX_k^i - 1 \\ \quad \text{if } T_{k-1} > 0 \text{ and } T_{k-1} \notin \{TH_{2\ell}, 1 \le \ell \le \lceil \frac{K}{2} \rceil - 1\} \\ \sum_{i=1}^{m} iX_k^i + R - 1 \qquad \text{if } T_{k-1} = 0. \end{cases} \tag{19}$$

In other words, whenever the remaining CET reaches TH_{2k} ($1 \le k \le \lceil K/2 \rceil - 1$), a units of time are added to the state to account for the CET increase due to broadcasts. The property of semi-Markov chain is retained, because whether or not to increase by a units of time depends only on T_{k-1}. Similarly, (2) should be rewritten as

$$\Phi^+(z) = E(z^{T_{k-1}+(1-\delta(T_{k-1}))R+a(\sum_{\ell=1}^{\lceil \frac{K}{2} \rceil -1} \delta'(T_{k-1}-TH_{2\ell}))-1}) \cdot E(z^{\sum_{i=1}^{m} iX_k^i})$$

where $\delta'(x)$ is the impulse function or the derivative of the unit step function, $\delta(x)$. Following the same (but more complex) derivation as in Section III, one can get a modified version of (10):

$$\Phi(z) = \frac{(1 - \lambda E(R) - \sum_{\ell=1}^{\lceil \frac{K}{2} \rceil -1} p_T(TH_{2\ell}))(z-1)e^{\lambda(\Phi_R(z)-1)}}{z - e^{\lambda(\Phi_R(z)-1)}} + \frac{e^{\lambda(\Phi_R(z)-1)}(z^a - 1)(\sum_{\ell=1}^{\lceil \frac{K}{2} \rceil -1} p_T(TH_{2\ell}) z^{TH_{2\ell}-1})}{1 - \frac{1}{z}e^{\lambda(\Phi_R(z)-1)}} \cdot \frac{E(R)(z-1)}{\Phi_R(z) - 1}. \tag{20}$$

Note that the above equations reduce to (10) if $a = 0$ (no broadcasting overhead). Before solving (20) for $p_T(.)$, one has to know the values of $p_T(TH_{2\ell})$, $1 \le \ell \le \lceil K/2 \rceil - 1$, which, in turn, depend on $\Phi(z)$. This recursive dependency can again be handled by using an iterative approach as follows. $\Phi(z)$ is first inverse z-transformed with $p_T(TH_{2\ell})$ set to 0. The resulting $p_T(.)$ (in particular, $p_T(TH_{2\ell})$'s) is then used to compute the new $\Phi(z)$ from which new $p_T(.)$ can then be calculated. This process will be repeated until $p_T(.)$ converges to a fixed value.

C. Effect of Communication Delays

Communication delays are composed of three components [39]: 1) the queueing delay, which is the time between the queueing of a task and the start of its transfer, 2) the transmission delay, which is the time between the first and last bits of the task transferred, and 3) the propagation delay, which is the time from the transmission of a bit at the sending node to its receipt by the destination node. The transmission delay depends on the size of the transferred task, and is thus assumed to be proportional (with ratio o_1) to the required execution time of the task. The propagation delay depends on the physical distance/characteristics of the communication medium between the sender and receiver nodes, is independent of traffic loads, and is thus assumed to be constant, o_2. The queueing delay depends heavily on traffic loads. Since region-change broadcasts and task transfers introduce additional traffic loads, the queueing delay under the proposed scheme is expected to be larger than others. However, the exact dependency of the queueing delay on these operations is difficult to model, because 1) the delay also depends on the capacity of the communication medium and the (contention) protocols used, both of which are application-dependent, and 2) the effect of region-change broadcasts on this delay depends on the state of the system, which changes dynamically with time. We thus assume in our model that the queueing delay due to task transfers and region-change broadcasts are proportional to (with ratio o_3 and o_4) the task transfer-out ratio (γ)

and external task arrival rate (λ), respectively.[10] Let $c_R(i)$ and $c_P(i)$ (r_R and r_P) denote the communication overheads encountered by a task with i units of execution time (task transfer-out ratios) in the random selection scheme and the proposed scheme, respectively, then we have $c_R(i) = o_1 \cdot i + o_2 + o_3 \cdot r_R$ and $c_P(i) = o_1 \cdot i + o_2 + o_3 \cdot r_P + o_4 \cdot \lambda$.

Considering the effect of communication delays, α_T in (14) should be modified as

$$\alpha_T = \sum_{i=1}^{m} \sum_{j=\lceil c(i) \rceil}^{T-1} \lambda q_{ij} \tag{21}$$

where $c(i)$ is $c_P(i)$ or $c_R(i)$, depending the scheme under consideration. Note that for those tasks whose laxity is less than the communication delay, there is no need to transfer them. Similarly, β_T is modified as

$$\beta_T = \sum_{i=1}^{m} \sum_{j=\lceil c_R(i) \rceil}^{T_m} \lambda \gamma_{j+1} q_{ij} \tag{22}$$

for the random selection scheme, and

$$\beta_T = \sum_{i=1}^{m} \sum_{j=\lceil T + c_P(i) \rceil}^{T_m} \lambda q_{ij} \gamma_{j+1} \frac{1-\gamma_{j+1}^n}{1-\gamma_{j+1}} \tag{23}$$

for the proposed LS scheme. Correctness of these expressions can be verified similarly to Corollaries 1–3 as shown in the Appendix.

V. Performance Analysis

To demonstrate the effectiveness of the proposed LS scheme and the validity of the analytic models, we present numerical results for the case when inter-arrival times of external tasks are exponentially distributed. Note, however, that the proposed LS scheme is not restricted to exponential distributions. The proposed scheme and four other LS schemes, i.e., no LS, LS with random selection, LS with state probing,[11] and LS with perfect information, are comparatively evaluated with both analytic models and simulation. Also, the simulation results were compared against those obtained from the analytical models.

A 16-node regular system[12] is used as an example. For convenience, the average task execution time, $E(R)$, is normalized to 1 throughout our analysis, so that all time-related parameters may be expressed in units of average task execution time. The external task arrival rate is varied from 0.2 to 0.9. The buddy set size is chosen to be 12, since the performance improvement by increasing the buddy set size beyond 10 was shown in [25] to be insignificant. The maximum number of nodes to be probed randomly for each locally unguaranteed task is restricted to 5 based on the finding in [3].

The computational overhead for *each* state probing, region-change broadcast (b), and probability distribution update (u) is assumed to be 1, 1, and 2% of $E(R)$, respectively. The transmission delay associated with each task transfer is assumed to be 10% of $E(R)$, i.e., $o_1 = 0.1$. The propagation delay is assumed to be 1% of $E(R)$, i.e., $o_2 = 0.01$. The coefficients associated with the queueing delay due to task transfers (o_3), region-change broadcasts (o_4), and state probes (o_5) are set to 0.1, 0.05, and 0.01, respectively.[13] These parameter values will be used throughout our analysis, unless specified otherwise. Besides, for notational convenience, $\{e_1, e_2, \cdots, e_m\}_{\{q_1, q_2, \cdots, q_m\}}$ is used to denote that a task requires execution time e_i with probability q_i, $1 \le i \le m$. If $q_i = q \forall i$, then $\{q_1, q_2, \cdots, q_m\}$ is condensed to q. Similar notation is used to describe the distribution of task laxity.

For each combination of system configuration and external task attributes, the simulation ran until it reached a confidence level 95% in the results for a maximum error (e.g., one half of the confidence interval) of 1) 2% of the specified probability if P_{dyn} is the measure of interest, 2) 0.2% of the specified response time value if mean response time is the measure, 3) 5% of the task arrival rate if the maximum system utilization is the measure, and 4) 5% of the ratio value if task transfer-out ratio or frequency of task collision is the measure. The number of simulation experiments needed to achieve the above confidence interval is calculated based on the assumption that the parameter to be estimated/measured has a normal distribution with unknown mean and variance.

We will first describe how to determine the values of those tunable parameters used in the proposed scheme. We will then evaluate and compare different LS schemes with respect to several important performance metrics derived from the analytic models and simulation. In spite of a large number of the parameters involved, the results are found to be quite robust in the sense that the conclusion drawn from the performance curves for a representative set of parameter values is valid over a wide range of parameter values. We will also analyze the effects of varying communication overheads and using QL (instead of CET) as the measure of workload on the performance of different schemes.

A. Determination of Tunable Parameters in the Proposed Scheme

The accuracy of prior/posterior distributions depends on the values of those tunable parameters, such as the probability update interval T_p, the probability updating ratio a, and the number (K) and values of thresholds of state regions. It is, however, difficult to objectively determine an optimal combination of these parameters which gives accurate prior/posterior distributions while incurring the least amount of overhead. The main reasons for this difficulty are:

- The choice of a and T_p depends on the variation of workload characteristics, which is application-dependent.
- The number and values of thresholds must be determined by optimizing the tradeoff between the resolution of state-space division and the overhead of the resulting region-change broadcasts. It is impossible to determine

[10]The reasons for assuming this linear relationship is that we can easily compute these coefficients using linear prediction and the data obtained from simulations.

[11]Only simulation results are shown for this scheme in our analysis.

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

[13]All overhead-related parameters are based on the data obtained from simulations.

TABLE I
CET DISTRIBUTIONS FOR DIFFERENT TASK SETS UNDER DIFFERENT SCHEMES ($N = 16$)

($\lambda = 0.8$)		No Sharing		State Prob.	Random Selection		Proposed Scheme		Perfect Scheme	
Task Attributes	$CET \leq$	Analytic	Simulation	Simulation	Analytic	Simulation	Analytic	Simulation	Analytic	Simulation
	0.0	0.2000	0.2138	0.2133	0.2121	0.2212	0.2609	0.2568	0.2914	0.2810
	0.8	0.3867	0.3710	0.5995	0.6216	0.6202	0.6920	0.6911	0.6474	0.6421
	1.6	0.5281	0.5246	0.8547	0.8602	0.8653	0.9019	0.9080	0.8858	0.8897
$ET = \{0.4, 0.8,$	2.4	0.6316	0.6362	0.9642	0.9611	0.9665	0.9811	0.9840	0.9820	0.9867
$1.2, 1.6\}_{0.25},$	3.2	0.7263	0.7222	0.9929	0.9887	0.9933	0.9975	0.9987	0.9998	0.9998
$L = \{1, 2, 3\}_{1/3}$	4.0	0.7816	0.7874	0.9981	0.9961	0.9979	0.9982	0.9995	1.0000	1.0000
	4.8	0.8412	0.8369	0.9996	0.9990	0.9997	0.9986	0.9998	1.0000	1.0000
	5.6	0.8824	0.8751	0.9999	0.9998	1.0000	0.9992	0.9999	1.0000	1.0000
	6.4	0.9086	0.9044	1.0000	1.0000	1.0000	0.9998	0.9999	1.0000	1.0000
	7.2	0.9293	0.9259	1.0000	1.0000	1.0000	0.9999	0.9999	1.0000	1.0000
	0.0	0.2000	0.2153	0.2094	0.2192	0.2287	0.2683	0.2853	0.3428	0.3787
	0.8	0.3867	0.3749	0.6473	0.7380	0.7210	0.8614	0.8409	0.8612	0.8469
	1.6	0.5281	0.5284	0.8890	0.9184	0.9262	0.9686	0.9770	0.9879	0.9961
$ET = \{0.4, 0.8,$	2.4	0.6316	0.6406	0.9581	0.9662	0.9741	0.9860	0.9910	0.9913	0.9991
$1.2, 1.6\}_{0.25},$	3.2	0.7263	0.7287	0.9857	0.9892	0.9930	0.9912	0.9957	0.9948	0.9997
$L = \{0.4, 0.8,$	4.0	0.7816	0.7902	0.9949	0.9923	0.9980	0.9930	0.9977	0.9968	0.9999
$1.2, 1.6\}_{0.25},$	4.8	0.8412	0.8419	0.9980	0.9961	0.9995	0.9948	0.9988	0.9984	1.0000
	5.6	0.8824	0.8803	0.9992	0.9984	0.9998	0.9978	0.9992	0.9996	1.0000
	6.4	0.9086	0.9128	0.9997	0.9996	1.0000	0.9984	0.9995	1.0000	1.0000
	7.2	0.9293	0.9301	0.9998	0.9999	1.0000	0.9990	0.9996	1.0000	1.0000

the optimal number and values of thresholds without a closed-form expression for this tradeoff. Moreover, the optimal number and values of thresholds also depend on both the laxity and the execution time distribution of the task set.

Thus, we shall determine the tunable parameters for each task set as follows.

S1. We fix all but one parameter of interest, and obtain the performance curve as a function of this parameter from which the optimal value for this parameter can be determined. Next, we keep the first parameter of interest fixed at its optimal value and vary another parameter of interest (while keeping all the rest parameters fixed at their originally chosen values). This process is repeated until all the parameters are exhausted.

S2. We check whether or not the simulation result (with P_{dyn} as the performance metric) with those tunable parameters chosen from S1 is consistent with, or close to, that computed from the queueing model. If the simulation result agrees with the analytic one, we tag this set of parameters as one of candidate parameter sets. Note that in the derivation of the performance model, we assume that the prior/posterior distributions be accurately constructed to obtain the analytic results. Consequently, the performance curves obtained from the queueing model serve as an upper bound. The set of parameters which gives the same performance as the analytic model is thus considered as a candidate because it yields the correct prior/posterior distribution given the computation/communication overheads.

S3. Since varying the order of parameters examined in S1 may give different values of parameters, we may end up with more than one candidate parameter set from which we choose the one with the smallest P_{dyn} and at the same time, reasonably small computation/communication overheads (e.g., the processing power used, the frequency of region-change broadcasts, or task transfer-out ratio) as an "optimal" parameter set.

Note that theoretically, the sets of parameters obtained through the above three steps may not be globally optimal, but our extensive simulations have shown them to yield good results, as compared to other schemes. Moreover, our simulation results indicate that the proposed scheme is robust to the variation of the tunable parameters. The change in P_{dyn} is shown to be less than 10^{-3} for any given change in either the threshold interval, the number of state regions, or the values of thresholds. The interested readers are referred to [1] for numerical examples and a detailed account of this. This robustness is an important advantage coming from the use of prior/posterior distributions and Bayesian analysis.

B. Evaluation of Important Performance Metrics

1) Distribution of CET, $p_T(.)$: The CET distribution can be obtained by using either (10) or (20), depending on the scheme under consideration. Table I gives some numerical examples of the CET distribution with respect to different distributions of task laxity for different LS schemes. The CET distribution obtained via simulation is shown to be very close to the analytic solution, with a 5% error in the cumulative distribution, indicating the validity of the analytic models. So,

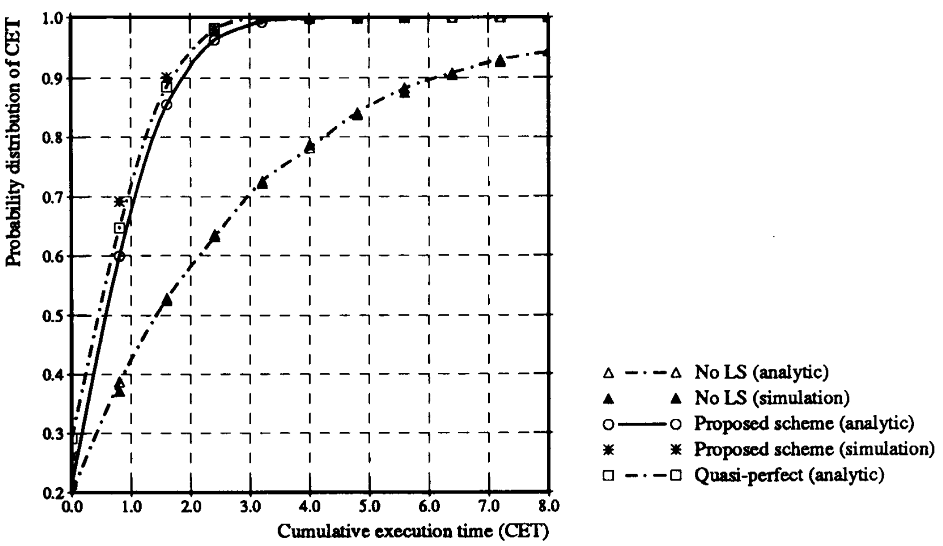


Fig. 4. Probability distribution of cumulative execution time for the task set with $\lambda = 0.8$, $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, and $L = \{1, 2, 3\}_{1/3}$.

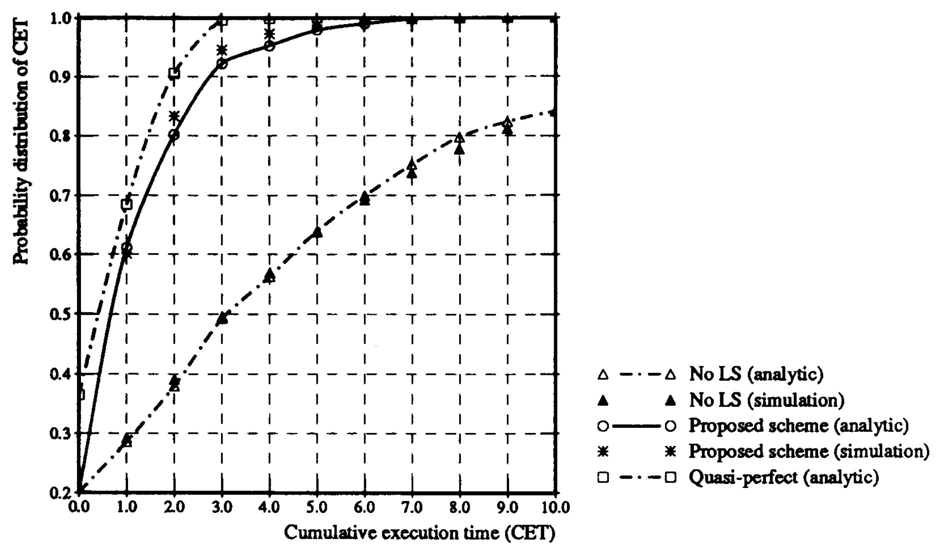


Fig. 5. Probability distribution of cumulative execution time for the task set with $\lambda = 0.8$, $ET = \{0.027, 0.27, 2.7\}_{1/3}$, and $L = \{1, 2, 3\}_{1/3}$.

we shall henceforth use the numerical results derived from the analytic models in the subsequent discussion, unless stated otherwise.

CET Distributions with respect to different distributions of task execution time are plotted in Fig. 4 and Fig. 5 with $\lambda = 0.8$. The numerical results are so close to one another among the state probing scheme, the random selection scheme, and the proposed scheme that only one curve corresponding to the proposed scheme is plotted. (Also, the results for no LS obtained from analytic modeling and simulations are so close to each other that they are almost indistinguishable in Figs. 4–5.) The CET distributions under different LS schemes approach unity much faster than those without LS, thus justifying the need of LS to handle bursty task arrivals in distributed systems. Besides, the CET distributions vary significantly as the distribution of task execution time varies; QL is thus not adequate to measure the workload of a node. More on this will be discussed later.

One interesting result is that the CET distribution associated with the proposed scheme does *not* approach unity with the fastest speed (Table I). However, the proposed LS scheme does have a higher $P(T \leq t)$ than others for $\forall t \in (0, T_{\max}]$, where $T_{\max}$ is the largest task laxity in the system. This is because the proposed scheme, instead of trying to minimizing the average CET on each node, aims to make each node's CET less than the laxity of any arrived task, so that P_{dyn} can be minimized.

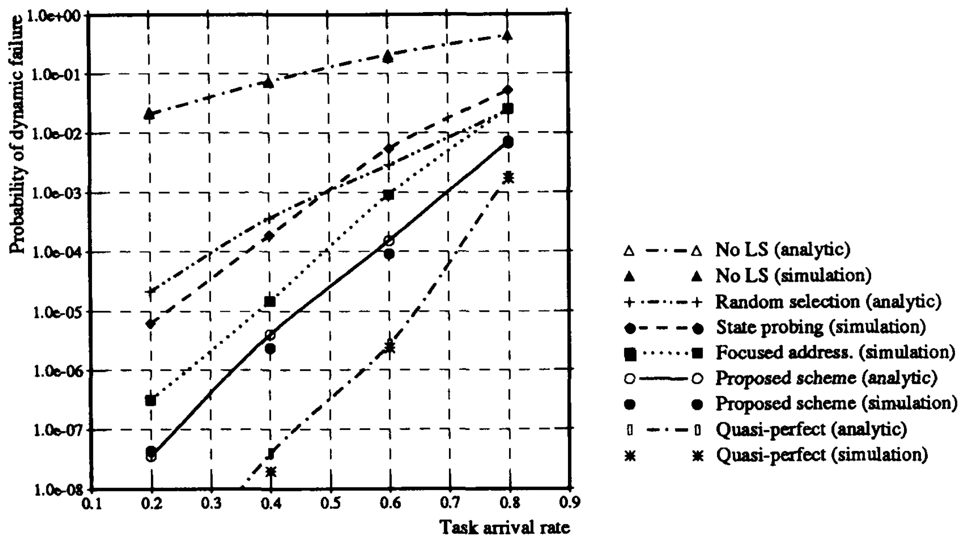


Fig. 6. Probability of dynamic failure (P_{dyn}) versus task arrival rate for a system with 16 nodes. (Task set: $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, $L = \{1, 2, 3\}_{1/3}$.)

Another interesting result is that CET distributions vary with the distributions of task laxity even when the distributions of task execution time are the same. This is because of the real-time application-oriented transfer policy used, where the laxity of an incoming task, rather than certain thresholds as in [3], [25], and [40], is used to determine whether or not to transfer a task. Consequently, we try to minimize P_{dyn} with respect to the distributions of *both* task laxity and task execution time, instead of balancing load *only* with respect to the distribution of task execution time.

2) Probability of Dynamic Failure: A dynamic failure occurs if the sum of the queueing-for-execution time and the task-transfer (if any) time exceeds the laxity of a task. Let $P_{dyn|d,e}$, $P_{dyn|d}$, and P_{dyn} denote the probability of missing the deadline of a task with execution time e and laxity d, the probability of missing the deadline of a task with laxity d, and the probability of dynamic failure, respectively. Then, $P_{dyn} = \sum_{j=0}^{T_m} P_{dyn|j}\hat{q}_j$, and, according to our queueing model, the other two probabilities can be expressed in terms of γ_j and $\hat{q}_j$ as:

1) *Noncooperative scheme (no LS):* $P_{dyn|d,e} = P_{dyn|d} = \gamma_{d+1}$, where γ_j is calculated from (10) with $\lambda(T) = \lambda$ $\forall T$.
2) *LS scheme with random selection:*

$$P_{dyn|d,e} = \prod_{i=0}^{n_d^e} \gamma_{(d+1-ic_R(e))},$$

and

$$P_{dyn|d} = \sum_{i=1}^{m} P_{dyn|d,i}\, q_i,$$

where $c_R(e)$ is the communication overhead associated with a task with execution time e under the random selection scheme, $n_d^e = \lfloor d/c_R(e) \rfloor$, and γ_j is obtained from (10) using α_T in (21) and β_T in (22).
3) *Perfect information scheme:* $P_{dyn|d,e} = P_{dyn|d} = \gamma_{d+1}^N$, where γ_j is calculated from (10) using $\lambda(T)$ in (18).
4) *The proposed scheme:*

$$P_{dyn|d,e} = \prod_{i=0}^{n_d^e} \gamma_{(d+1-ic_P(e))},$$

and

$$P_{dyn|d} = \sum_{i=1}^{m} P_{dyn|d,i}\, q_i,$$

where $c_P(e)$ is the communication overheads associated with a task with execution time e under the proposed LS scheme, $n_d^e = \lfloor d/c_P(e) \rfloor$, and γ_j is obtained from (20) using α_T in (21) and β_T in (23).

Figs. 6 and 7 are the plots of P_{dyn} versus task arrival rate (λ), and $P_{dyn|d}$ versus task laxity (d), obtained from both the analytic models and simulation. Table II shows numerical examples of $P_{dyn|d}$ under different schemes. The random selection scheme outperforms the state probing scheme when the system load gets heavy or the task laxity gets tight, e.g., $L = \{1, 2, 3\}$ as compared to $L = \{1\}$ in Table II(b). The reasons for this are: 1) under heavy loads, most nodes are likely to become unable to guarantee tasks, which will in turn make state probing unsuccessful most of the time, and 2) probing other nodes before sending an unguaranteed task introduces two communication messages (one for request and the other for response), whereas the random selection scheme does not require such message. This phenomenon becomes more pronounced under stringent time constraints.

Our analytic and simulation studies have shown the proposed LS scheme to be significantly better than both the state probing scheme and the random selection scheme in

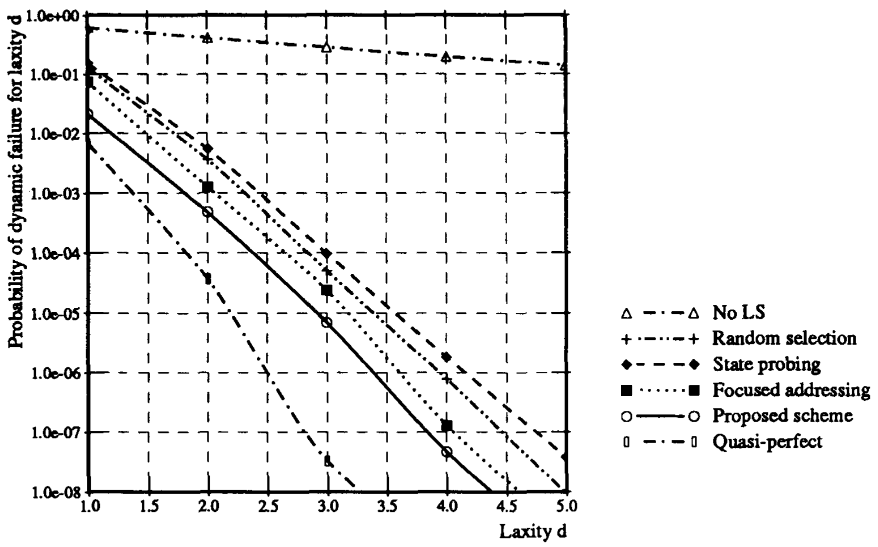


Fig. 7. Probability of tasks with laxity d missing deadlines ($P_{dyn|d}$) versus task laxity d for a system with 16 nodes. (Task set: $\lambda = 0.8$, $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, $L = \{1, 2, 3, 4, 5\}_{0.2}$.)

meeting deadlines. This is in sharp contrast to the common notion [3] that simple LS schemes perform nearly as well as complex ones for general-purpose systems. By making judicious exchange/use of state information, the proposed LS scheme—though it incurs more computation/communication overheads—achieves notable performance improvement over those simple LS schemes.

The proposed LS scheme is also superior to the focused addressing scheme, because in the latter

1) The focused node or its successor node—the node that the focused node will re-transfer the task to—among those "seen" capable is basically chosen randomly, thus increasing the chance of two nodes sending their unguaranteed tasks to the same node.
2) Not many RFB messages are issued under light loads, making a node unable to keep its observation of other nodes up-to-date and thus increasing the chance of transferring a task to an incapable focused node. This becomes intolerable for tasks with tight laxities.
3) Requests and replies for bids become excessive under heavy loads, thus increasing communication delays. The state information collected via periodic state exchange or the bids sent from other nodes may become out-of-date.

The proposed scheme performs, however, worse than the quasi-perfect LS scheme because of the processing overhead (introduced by the probability update process) and the communication delays (in task transfers and region-change broadcasts). See Table III for the case where all processing/communication overheads are set to zero except the task transmission delay which remains 10% of the task execution time. Without considering all the processing/communication overheads, the performance of the proposed scheme is very close to that of the quasi-perfect LS scheme. This implies that ignoring implementation overheads underestimates P_{dyn}, which is unacceptable for critical real-time applications.

3) Maximum System Utilization, $\lambda_{\max}$*:* The system utilization is defined as the ratio of external task arrival rate (λ) to the system service rate ($1/E(R)$). The service rate is normalized to 1 in our analysis, and thus, the system utilization is simply λ. Since P_{dyn} increases as system load gets heavier, there exists an upper bound for λ, termed as *maximum system utilization* $\lambda_{\max}$, below which $P_{dyn} \leq \epsilon$ can be guaranteed for some $\epsilon > 0$. Fig. 8 shows some numerical examples of $\lambda_{\max}$ versus ϵ. Among all LS schemes, the proposed scheme offers the performance closest to that of the quasi-perfect LS scheme, and usually outperforms both the state probing scheme and the random selection scheme by an order of magnitude.

4) Mean Response Time: Probabilistically, the mean response time (MRT) is the sum of the average CET on a node, $\sum_{i=0}^{\infty} ip_T(i)$, and the average required execution time, $E(R) = 1$, i.e., $S = \sum_{i=0}^{\infty} ip_T(i) + 1$. It is conventionally used as a global system performance index in general-purpose distributed systems, and many approaches have been developed under the goal of minimizing MRT. Table IV gives MRT with respect to different task attributes under different schemes. MRT increases as the system load increases [Table IV(a)], or the variance of either the distribution of task execution time or the distribution of task laxity gets large [Table IV(b) and (c)].

MRT associated with the proposed scheme varies least drastically with the change of distributions of task laxity and/or task execution time as compared to those associated with other LS schemes. This is due to the use of Bayesian analysis to choose a receiver node for task transfer. Moreover, our LS scheme outperforms others (except for the quasi-perfect LS scheme) even when all the computational overheads are taken into account and MRT is used to measure their performance.

TABLE II
$P_{dyn|d}$ VERSUS TASK LAXITY d FOR DIFFERENT TASK SETS UNDER DIFFERENT SCHEMES ($N = 16$). (a) $\lambda = 0.8$. (b) $\lambda = 0.4$

($\lambda = 0.8$) Task Attributes	Lax. d	No Sharing	State Probing	Random Selection	Focused Addressing	Proposed Scheme	Quasi-Perfect
$ET = \{0.4, 0.8,$	1	0.6184	0.1515	0.1286	8.649×10^{-2}	2.438×10^{-2}	5.247×10^{-3}
$1.2, 1.6\}_{0.25}$,	2	0.4336	4.779×10^{-3}	2.302×10^{-3}	9.746×10^{-4}	3.034×10^{-4}	6.316×10^{-5}
$L = \{1, 2, 3\}_{1/3}$	3	0.2894	3.514×10^{-5}	1.447×10^{-5}	1.026×10^{-5}	7.156×10^{-6}	5.604×10^{-8}
$ET = \{0.027,$	1	0.7121	0.2476	0.1856	0.1524	4.342×10^{-2}	3.274×10^{-2}
$0.27, 2.703\}_{1/3}$,	2	0.5896	5.086×10^{-2}	3.543×10^{-2}	2.249×10^{-2}	6.432×10^{-3}	2.316×10^{-3}
$L = \{1, 2, 3\}_{1/3}$	3	0.4923	4.994×10^{-3}	2.967×10^{-3}	9.594×10^{-4}	3.617×10^{-4}	1.604×10^{-4}
$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, $L = \{1\}$	1	0.5894	0.1293	8.242×10^{-2}	7.153×10^{-2}	2.094×10^{-2}	5.946×10^{-3}

(a)

($\lambda = 0.4$) Task Attributes	Lax. d	No Sharing	State Probing	Random Selection	Focused Addressing	Proposed Scheme	Quasi-Perfect
$ET = \{0.4, 0.8,$	1	0.1578	5.594×10^{-4}	7.645×10^{-4}	8.264×10^{-5}	1.187×10^{-5}	4.746×10^{-8}
$1.2, 1.6\}_{0.25}$,	2	0.0479	6.402×10^{-6}	1.178×10^{-5}	9.536×10^{-7}	2.658×10^{-7}	1.042×10^{-10}
$L = \{1, 2, 3\}_{1/3}$	3	0.0176	2.782×10^{-7}	4.765×10^{-7}	4.846×10^{-8}	2.184×10^{-9}	0
$ET = \{0.027,$	1	0.2976	4.270×10^{-3}	5.247×10^{-3}	9.079×10^{-4}	2.035×10^{-4}	3.462×10^{-6}
$0.27, 2.703\}_{1/3}$,	2	0.1846	2.346×10^{-5}	1.685×10^{-4}	9.896×10^{-6}	1.875×10^{-6}	1.395×10^{-8}
$L = \{1, 2, 3\}_{1/3}$	3	0.0796	2.693×10^{-7}	3.276×10^{-6}	7.903×10^{-8}	3.428×10^{-8}	0
$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, $L = \{1\}$	1	0.1736	8.163×10^{-4}	5.943×10^{-4}	3.818×10^{-4}	6.547×10^{-5}	8.746×10^{-8}

(b)

TABLE III
$P_{dyn|d}$ FOR A TASK SET WITH $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$ AND $L = \{1, 2, 3\}_{1/3}$ UNDER THE IDEAL CONDITION

Arrival Rate (λ)	Lax. d	No Sharing	State Probing	Random Selection	Focused Addressing	Proposed Scheme	Quasi-Perfect
	1	0.6184	3.027×10^{-2}	4.325×10^{-2}	2.878×10^{-2}	1.862×10^{-2}	5.247×10^{-3}
0.8	2	0.4336	3.161×10^{-4}	2.946×10^{-4}	2.874×10^{-4}	2.389×10^{-4}	6.316×10^{-5}
	3	0.2894	2.763×10^{-6}	8.846×10^{-6}	5.323×10^{-7}	1.024×10^{-7}	5.604×10^{-8}
	1	0.1578	1.875×10^{-7}	7.894×10^{-5}	4.275×10^{-7}	1.870×10^{-7}	4.746×10^{-8}
0.4	2	0.0479	8.764×10^{-8}	3.376×10^{-6}	9.619×10^{-8}	3.678×10^{-9}	1.042×10^{-10}
	3	0.0176	4.763×10^{-10}	5.416×10^{-8}	5.136×10^{-10}	0	0

This is due to the fact that, to minimize P_{dyn}, the proposed scheme aims to share all the unguaranteed tasks among capable nodes (as proven in Theorem 2), thus at the same time balancing the load in the system.

5) Task Transfer-out Ratio, r: The task transfer ratio, r, is defined as the fraction of arrived tasks (both external and transferred-in tasks) that have to be transferred out, and can be expressed as $r = \sum_{j=1}^{T_m} \gamma_{j+1} q_j$. r is a measure of the traffic overheads due to task transfers.

Table V gives r for various task attributes under different schemes. Generally, r increases as the system load gets heavier and/or the task laxity gets tighter. Moreover, as the variance of task execution time increases, the task transfer-out ratio increases. This is because a node easily becomes incapable of guaranteeing tasks upon the arrival of *long* tasks or tasks with a *short* laxity, thus resulting in more task transfers under these conditions.

Under light and medium loads, the task transfer ratios associated with different schemes are very close to one another. This is because most of the tasks can be guaranteed locally or with at most one task transfer; not many task-retransfers occur, and thus the location policies employed by different schemes have little influence on system performance. When the system load gets heavier, the way of choosing a node for task transfer/retransfer becomes more important. The state probing scheme has a task transfer ratio closest to that of the quasi-perfect LS scheme since it always checks the capability of a node before transferring a task. The proposed scheme is slightly inferior to the state probing scheme due to the use of imperfect observation to probabilistically decide on the location of task transfer. However, the proposed scheme does not require, at the time of decision, the additional round-trip communication associated with state probing which may be detrimental to critical-time applications.

The focused addressing scheme performs slightly better than the proposed scheme under light loads. However, when the system load gets heavier, the performance of the focused addressing scheme deteriorates due to the increased probability of making incorrect LS decisions based on out-of-date[14] information.

[14] as a result of the increased communication delay caused by an excessive number of bidding messages.

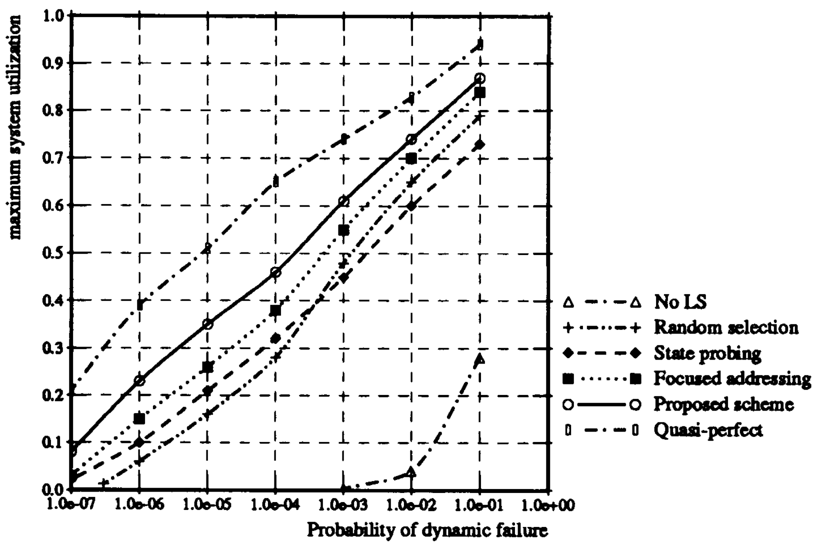


Fig. 8. Maximum system utilization versus the probability of dynamic failure P_{dyn} for a system with 16 nodes.

TABLE IV
COMPARISON OF MEAN RESPONSE TIME AMONG DIFFERENT LS SCHEMES. (a) MRT VERSUS TASK ARRIVAL RATE FOR A TASK SET WITH $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$ AND $L = \{1, 2, 3\}_{1/3}$. (b) MRT FOR DIFFERENT TASK SETS UNDER DIFFERENT LS SCHEMES ($\lambda = 0.8$). (c) MRT FOR DIFFERENT TASK SETS UNDER DIFFERENT LS SCHEMES ($\lambda = 0.4$)

Arrival Rate (λ)	No LS	State Prob.	Random Selection	Focused Address.	Prop. Scheme	Quasi-Perfect
0.2	1.154	1.117	1.115	1.117	1.118	1.108
0.4	1.406	1.302	1.268	1.265	1.257	1.236
0.6	1.868	1.498	1.483	1.466	1.446	1.439
0.8	3.521	1.872	1.836	1.789	1.720	1.668

(a)

($\lambda = 0.8$) Task Attributes		No LS	State Prob.	Random Selection	Focused Address.	Prop. Scheme	Quasi-Perfect
$L = \{1,2,3\}_{1/3}$	$ET = \{1\}$	3.024	1.796	1.782	1.763	1.704	1.687
	$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	3.521	1.872	1.836	1.789	1.720	1.668
	$ET = \{0.027, 0.27, 2.703\}_{1/3}$	6.106	2.174	2.101	1.979	1.822	1.804
$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	$L = \{1\}$	3.521	1.547	1.536	1.502	1.439	1.411
	$L = \{1,2\}_{0.5}$	3.521	1.688	1.629	1.679	1.576	1.547
	$L = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	3.521	1.812	1.604	1.579	1.418	1.386

(b)

($\lambda = 0.4$) Task Attributes		No LS	State Prob.	Random Selection	Focused Address.	Proposed Scheme	Quasi-Perfect
$L = \{1,2,3\}_{1/3}$	$ET = \{1\}$	1.348	1.267	1.264	1.259	1.248	1.240
	$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	1.406	1.302	1.268	1.265	1.257	1.236
	$ET = \{0.027, 0.27, 2.703\}_{1/3}$	1.806	1.350	1.335	1.314	1.316	1.301
$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	$L = \{1\}$	1.406	1.198	1.184	1.165	1.167	1.154
	$L = \{1,2\}_{0.5}$	1.406	1.250	1.242	1.233	1.228	1.214
	$L = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	1.406	1.211	1.190	1.162	1.163	1.148

(c)

6) Frequency of Task Collision, f_{tc}: The frequency of task collision is defined as the fraction of *transferred* tasks that are not guaranteed on remote nodes after their transfer. This is a measure for the capability of the LS algorithms in reducing the probability of task re-transfers. Fig. 9 shows the *simulation* results for different LS schemes.

Generally, f_{tc} increases as the system load gets heavier, the task laxity gets tighter, and/or as the variance of task execution time increases for the same reason that leads to the increase of r in Section V-B5. The state probing scheme has the lowest frequency of task collision, because it always checks the capability of a node (and thus maintains/uses the

TABLE V
COMPARISON OF TASK TRANSFER-OUT RATIO AMONG DIFFERENT LS SCHEMES. (a) r VERSUS TASK ARRIVAL RATE FOR THE TASK SET WITH $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$ AND $L = \{1, 2, 3\}_{1/3}$ UNDER DIFFERENT LS SCHEMES. (b) $\lambda = 0.8$ (c) $\lambda = 0.4$

Arrival Rate (λ)	State Prob.	Random Selection	Focused Address.	Prop. Scheme	Quasi-Perfect
0.2	0.019	0.024	0.020	0.021	0.019
0.4	0.058	0.068	0.056	0.056	0.052
0.6	0.114	0.156	0.116	0.112	0.107
0.8	0.185	0.338	0.241	0.224	0.184

(a)

($\lambda = 0.8$) Task Attributes		State Prob.	Random Selection	Focused Address.	Prop. Scheme	Quasi-Perfect
$L = \{1,2,3\}_{1/3}$	$ET = \{1\}$	0.162	0.296	0.232	0.227	0.158
	$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	0.185	0.338	0.241	0.224	0.184
	$ET = \{0.027, 0.27, 2.703\}_{1/3}$	0.278	0.524	0.398	0.362	0.274
$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	$L = \{1\}$	0.306	0.463	0.355	0.348	0.286
	$L = \{1,2\}_{0.5}$	0.221	0.383	0.286	0.266	0.216
	$L = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	0.289	0.454	0.367	0.322	0.282

(b)

($\lambda = 0.4$) Task Attributes		State Prob.	Random Selection	Focused Address.	Prop. Scheme	Quasi-Perfect
$L = \{1,2,3\}_{1/3}$	$ET = \{1\}$	0.039	0.041	0.038	0.040	0.035
	$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	0.058	0.068	0.056	0.056	0.052
	$ET = \{0.027, 0.27, 2.703\}_{1/3}$	0.129	0.163	0.130	0.131	0.122
$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	$L = \{1\}$	0.120	0.136	0.118	0.124	0.110
	$L = \{1,2\}_{0.5}$	0.076	0.082	0.075	0.074	0.073
	$L = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$	0.121	0.154	0.123	0.131	0.120

(c)

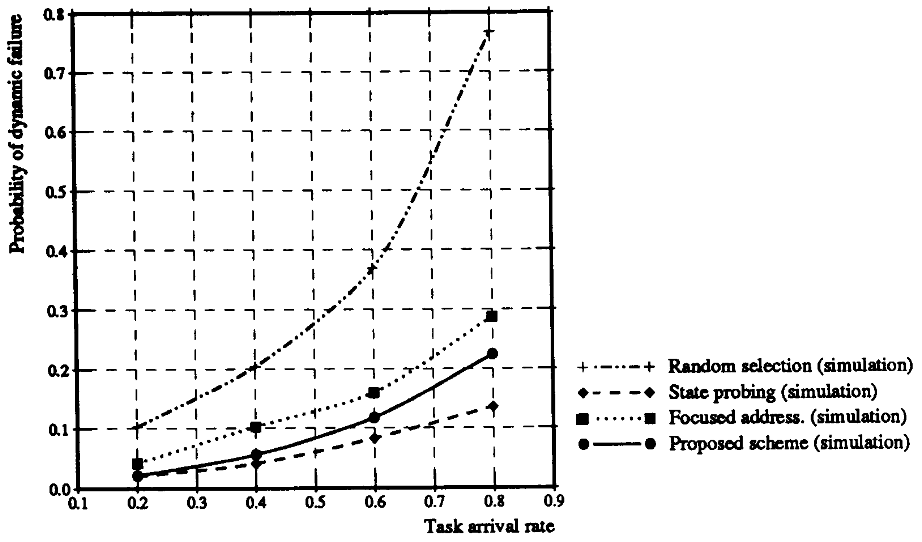


Fig. 9. Frequency of task collision versus external task arrival rate for a 16-node system with a task set: $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$ and $L = \{1, 2, 3\}_{1/3}$.

most up-to-date state information) before transferring a task. The random selection scheme, on the other hand, does not use any state information for LS decision, and thus necessarily has the highest frequency of task collision. The f_{tc} of the proposed scheme lies between that of state probing and that of focused addressing. The reason for the focused addressing scheme to be inferior to the proposed scheme is that the state information of other nodes is collected via periodic information exchanges and/or in the bids received in previous RFB activities, and may be obsolete at the time of choosing a focused node. That is, the focused node is likely to be unable to guarantee the task and needs to re-transfer the task based on the bids received in the current RFB activity. The proposed LS scheme may also use obsolete state information, but it neutralizes the undesirable

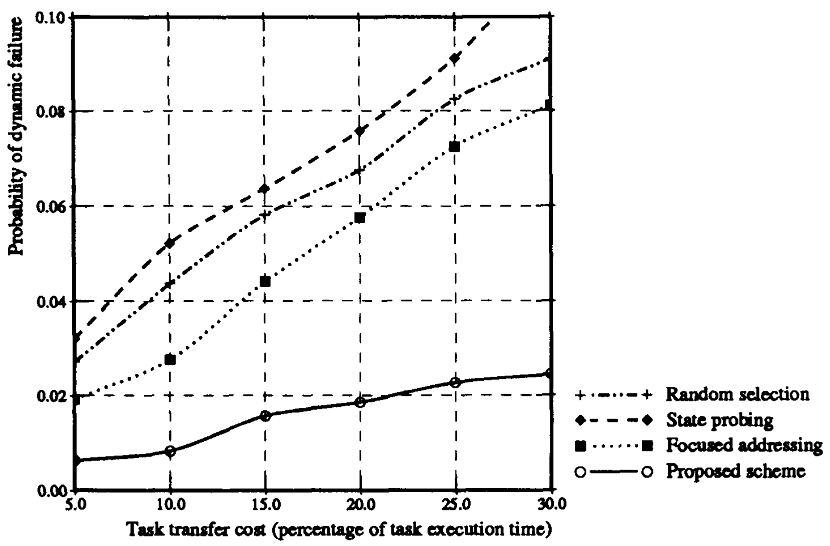


Fig. 10. Probability of dynamic failure P_{dyn} versus task transfer costs for a system of 16 nodes. (Task set: $\lambda = 0.8$, $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, $L = \{1, 2, 3\}_{1/3}$.)

effect of using out-of-date information with Bayesian decision analysis.

7) Sensitivity to Communication Delays: There are two types of communication delay needed to be considered in load sharing: i) the state-collection delay incurred from region-change broadcasts/state probes, where the queueing delay (or the queueing-related costs, o_3, o_4, and o_5) plays a dominating role; ii) the delay associated with task transfers, where both the queueing delay and the transmission delay (or task transmission cost o_1) dominate. To study the impact of communication delay on the performance, P_{dyn} was computed for each scheme with 1) the task transmission costs being 5, 10, 15, and 20% of the task execution time (i.e. $o_1 = 0.05, 0.10, 0.15, 0.20$), and 2) the queueing-related costs obtained from simulation (i.e. o_2, o_3, o_4, and o_5) being halved, doubled, and tripled.

As shown in Fig. 10 and Table VI(a), both the state probing scheme and the random selection scheme are more sensitive to the variation of transmission delay as compared to the proposed scheme. The performance degradation of the state probing scheme occurs, because, as the task transmission delay increases, other tasks may arrive at the node probed between the time it was probed and the time an unguaranteed task of the probing node arrived at the probed node. Thus, there is not much correlation between the state of the probed node when it was probed and the state of that node when an unguaranteed task arrived. (Similarly, one can reason about the performance degradation of the focused addressing scheme.) The performance of the random selection scheme degrades as the transmission delay increases, due to the combined effect of higher task transfer-out ratio (Table V) and larger transmission cost.

Fig. 11 [Table VI(b)] shows the effect of varying queueing-related costs on the performance of LS schemes. The state probing scheme is most sensitive to the variation of the queueing delay because, in addition to suffering the same effect as varying the transmission delay, the state probing scheme generates two additional messages per probe increasing the possibility of a task missing its deadline, especially when the queueing delay is large. Varying queueing-related costs have the same effect as varying transmission delay on the random selection scheme.

By contrast, the proposed scheme is made less susceptible to both the queueing delay and the transmission delay by using prior/posterior distributions to characterize the correlation between the observed state and the true state.

8) CET Versus QL as the Measure of Workload: All dynamic LS schemes use information on the workload of each node to determine when and where to transfer a task. As discussed before, QL is not appropriate to describe the workload state of a node for real-time applications. To show this, we ran simulations with QL as the state and compared the result with that obtained from the analytic solution with CET as the state. As shown in Table VII, the performance of the proposed scheme with QL as the state is close to (and sometimes worse than) that of the random selection scheme. This is because the proposed scheme essentially degenerates to the random selection scheme with 1) improper QL information which is correlated to CET in an unpredictable manner, and 2) more overheads due to region-change broadcasts and probability updates. The performance degradation becomes more pronounced as the variance of the distribution of task execution time gets larger.

VI. Concluding Remarks

Queueing models are developed to quantitatively assess the proposed scheme as well as three other schemes. Instead of the commonly-used QL, CET is used as the load state of each node. Each node's workload most relevant to a real-time task

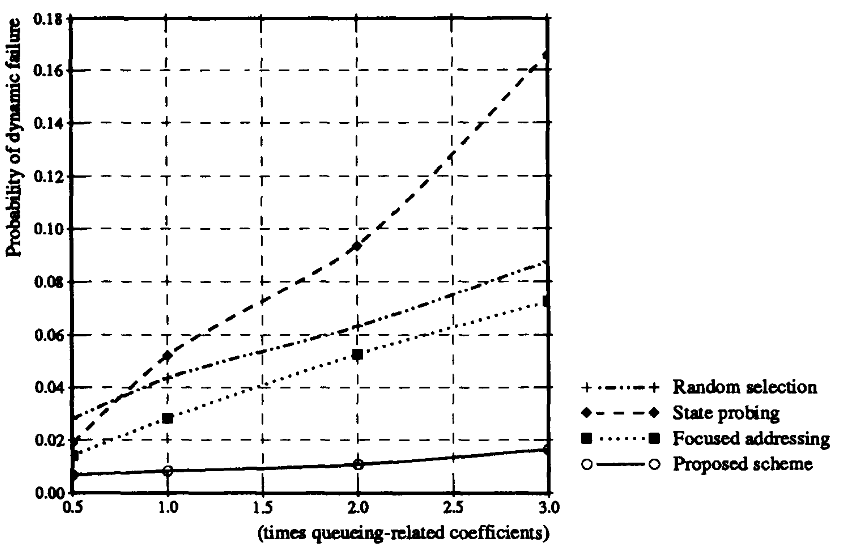


Fig. 11. Probability of dynamic failure P_{dyn} versus queueing delay coefficients for a system of 16 nodes. (Task set: $\lambda = 0.8$, $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, $L = \{1, 2, 3\}_{1/3}$.)

TABLE VI
EFFECTS OF COMMUNICATION DELAY ON P_{dyn} FOR A TASK SET WITH $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$ AND $L = \{1, 2, 3\}_{1/3}$ UNDER DIFFERENT SCHEMES. (a) EFFECT OF TASK TRANSFER COSTS ON P_{dyn}. (b) EFFECT OF QUEUEING DELAYS ON P_{dyn}

($\lambda = 0.8$) Transfer Costs	Lax. d	State Prob.	Random Selection	Focused Address.	Prop. Scheme
5%	1	0.1090	8.074×10^{-2}	5.623×10^{-2}	1.845×10^{-2}
	2	3.257×10^{-3}	7.316×10^{-4}	3.924×10^{-4}	1.654×10^{-4}
	3	1.167×10^{-5}	5.239×10^{-6}	5.311×10^{-6}	1.987×10^{-6}
10%	1	0.1515	0.1286	8.649×10^{-2}	2.438×10^{-2}
	2	4.779×10^{-3}	2.302×10^{-3}	9.746×10^{-4}	3.034×10^{-4}
	3	3.514×10^{-5}	1.447×10^{-5}	1.026×10^{-5}	7.156×10^{-6}
15%	1	0.1834	0.1678	0.1328	3.725×10^{-2}
	2	7.524×10^{-3}	6.423×10^{-3}	2.678×10^{-3}	9.845×10^{-4}
	3	5.851×10^{-5}	3.675×10^{-5}	2.436×10^{-5}	1.436×10^{-5}
20%	1	0.2068	0.1892	0.1708	6.029×10^{-2}
	2	1.154×10^{-2}	7.467×10^{-3}	3.849×10^{-3}	1.346×10^{-3}
	3	1.878×10^{-4}	8.386×10^{-5}	5.014×10^{-5}	2.112×10^{-5}
25%	1	0.2550	0.2463	0.2212	6.486×10^{-2}
	2	1.394×10^{-2}	1.018×10^{-2}	8.746×10^{-3}	3.421×10^{-3}
	3	3.869×10^{-4}	1.846×10^{-4}	9.249×10^{-5}	6.857×10^{-5}

(a)

($\lambda = 0.8$) Q–Delay Coefficients	Lax. d	State Prob.	Random Selection	Focused Address.	Prop. Scheme
halved	1	4.091×10^{-2}	8.415×10^{-2}	4.206^{-2}	1.978×10^{-2}
	2	3.758×10^{-4}	1.362×10^{-3}	6.270×10^{-4}	3.214×10^{-4}
	3	2.813×10^{-6}	7.145×10^{-6}	5.708×10^{-6}	3.758×10^{-6}
values from simulation	1	0.1515	0.1286	8.649×10^{-2}	2.438×10^{-2}
	2	4.779×10^{-3}	2.302×10^{-3}	9.746×10^{-4}	3.034×10^{-4}
	3	3.514×10^{-5}	1.447×10^{-5}	1.026×10^{-5}	7.156×10^{-6}
doubled	1	0.2134	0.1823	0.1538	3.129×10^{-2}
	2	2.801×10^{-2}	7.216×10^{-3}	1.537×10^{-3}	6.436×10^{-4}
	3	5.513×10^{-4}	2.875×10^{-5}	1.324×10^{-5}	9.578×10^{-6}
tripled	1	0.4194	0.2458	0.2173	4.245×10^{-2}
	2	7.475×10^{-2}	1.675×10^{-2}	1.267×10^{-2}	7.213×10^{-3}
	3	3.842×10^{-3}	2.334×10^{-4}	1.726×10^{-4}	2.046×10^{-5}

(b)

can thus be accurately modeled. Moreover, by including all computation/communication overheads, the proposed analytic models provide a means of evaluating the absolute real-time performance of LS schemes. The assumptions and approximations made in our analysis were checked with event-driven simulations.

TABLE VII
PERFORMANCE COMPARISON OF USING CET/QL AS THE MEASURE OF WORKLOAD

$(\lambda = 0.8)$ Task Attributes	laxity d	Random Selection (state:CET)	Proposed Scheme (state:QL)	Proposed Scheme (state:CET)
$ET = \{0.4, 0.8,$	1	0.1286	0.1085	2.438×10^{-2}
$1.2, 1.6\}_{0.25}$,	2	2.302×10^{-3}	1.625×10^{-3}	3.034×10^{-4}
$L = \{1, 2, 3\}_{1/3}$	3	1.447×10^{-5}	9.604×10^{-6}	7.156×10^{-6}
$ET = \{0.027,$	1	0.1856	0.2492	4.342×10^{-2}
$0.27, 2.703\}_{1/3}$,	2	3.543×10^{-2}	4.924×10^{-2}	6.432×10^{-3}
$L = \{1, 2, 3\}_{1/3}$	3	2.967×10^{-3}	3.142×10^{-3}	3.617×10^{-4}
$ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, $L = \{1\}$	1	8.242×10^{-2}	7.606×10^{-2}	2.094×10^{-2}

TABLE VIII
$P_{dyn|d}$ VERSUS TASK LAXITY d FOR A TASK SET WITH $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$ AND $L = \{1, 2, 3\}_{1/3}$ IN 64-NODE HOMOGENEOUS/HETEROGENEOUS DISTRIBUTED SYSTEMS

Task Arrival rate per node	System Attribute	Laxity d	$P_{dyn\|d}$
$\bar{\lambda} = 0.8$	Homogeneous System	1	3.326×10^{-3}
	$(\lambda_i = 0.8,$	2	8.798×10^{-6}
	$1 \leq i \leq 64)$	3	1.266×10^{-7}
	Inhomogeneous System	1	1.470×10^{-3}
	$(\lambda_j = 0.65, \lambda_{16+j} = 0.95,$	2	5.855×10^{-6}
	$\lambda_{32+j} = 0.65, \lambda_{48+j} = 0.95,$	3	9.872×10^{-8}
	$1 \leq j \leq 16)$		
$\bar{\lambda} = 0.6$	Homogeneous System	1	5.079×10^{-5}
	$(\lambda_i = 0.6,$	2	2.562×10^{-7}
	$1 \leq i \leq 64)$	3	6.745×10^{-9}
	Inhomogeneous System	1	1.565×10^{-5}
	$(\lambda_j = 0.65, \lambda_{16+j} = 0.95,$	2	9.782×10^{-8}
	$\lambda_{32+j} = 0.65, \lambda_{48+j} = 0.95,$	3	3.682×10^{-9}
	$1 \leq j \leq 16)$		

Both the analytic and simulation results indicate that by using judicious exchange/use of state information and Bayesian decision mechanisms, the proposed LS scheme makes a significant improvement in minimizing P_{dyn} over those simple LS algorithms. This is in sharp contrast to the common notion that simple LS algorithms yield performance close to that of complex algorithms for general-purpose systems where minimizing the mean response time is the main concern. Since missing a task deadline can cause a disastrous accident in a real-time environment, a more complex, but intelligent, LS scheme should be employed to minimize P_{dyn}.

We assumed a first-come-first-served policy on each node: a newly-arrived task is inserted at the end of task queue if it can be guaranteed on that node, and will otherwise be considered for transfer. This policy is simple and ensures to preserve the existing guarantees. However, to reduce P_{dyn}, the minimum-laxity-first-served policy is shown to be better [41] for queueing the incoming tasks at each node. That is, the tasks on a node are ordered by their laxities, and a task with the minimum laxity is always executed first by the node. If the minimum-laxity-first-served policy is employed, the transfer policy would not be simply of the threshold type. For example, a newly-arrived task may be inserted somewhere in the task queue, not necessarily at the end of the queue, thus possibly violating some of existing guarantees. (Such tasks, if possible, must be transferred to other capable nodes.) How to modify the parameters α_T and β_T to account for this transfer policy is currently under investigation.

Though we considered only homogeneous systems, the proposed scheme can also be applied to heterogeneous systems where different nodes may have different arrival rates of external tasks. Our simulation results indicate that the performance improvement is even more pronounced for heterogeneous systems than homogeneous ones with the same average task arrival rate (see Table VIII). This is because that increasing the degree of heterogeneity increases the possibility that uneven task arrivals temporarily make some nodes incapable of guaranteeing tasks while leaving other nodes idle/underloaded. This situation can be effectively handled, and the processing power of those idle/underloaded nodes can be fully utilized by using the proposed LS scheme. How to extend our analytic models to include the case for heterogeneous systems is an interesting, but difficult, matter.

APPENDIX
VERIFICATION OF FLOW CONSERVATION

To verify the correctness of (15), one has to show that flow conservation holds for the system with the random LS scheme.

Corollary 1: For the random LS scheme, $\sum_{k=0}^{\infty} \alpha_k p_T(k) = \beta_k$.

Proof:

$$\begin{aligned}
\sum_{k=0}^{\infty} \alpha_k p_T(k) &= \sum_{k=0}^{\infty} \Big(\sum_{i=1}^{m} \sum_{j=0}^{k-1} \lambda q_{ij}\Big) p_T(k) \\
&= \sum_{i=1}^{m} \sum_{j=1}^{\infty} \sum_{k=j+1}^{\infty} \lambda q_{ij} p_T(k) \\
&= \sum_{j=1}^{T_m} \lambda \sum_{i=1}^{m} q_{ij} \sum_{k=j+1}^{\infty} p_T(k) \\
&= \sum_{j=1}^{T_m} \lambda \hat{q}_j \gamma_{j+1} \\
&= \beta_k
\end{aligned}$$

where the second equality follows from interchanging the summation indexes while preserving the range of summation appropriately, and the third equality follows from $q_{ij} = 0$ for $j \geq T_m + 1$. □

To verify the correctness of (16), one has to show that flow conservation holds for the system with the proposed LS scheme, i.e.,

Corollary 2: When those tasks being rejected are not considered,

$$\sum_{k=0}^{\infty} \beta_k p_T(k) = \sum_{k=0}^{\infty} \alpha_k p_T(k),$$

for the proposed scheme.

Proof:

$$\begin{aligned}\sum_{k=0}^{\infty} \beta_k p_T(k) &= \sum_{k=0}^{\infty}\sum_{j=k}^{T_m} \lambda \hat{q}_j \gamma_{j+1} \frac{1-\gamma_{j+1}^n}{1-\gamma_{j+1}} p_T(k) \\ &= \sum_{j=0}^{T_m} \lambda \hat{q}_j \gamma_{j+1} \frac{1-\gamma_{j+1}^n}{1-\gamma_{j+1}} \sum_{k=0}^{j} p_T(k) \\ &= \sum_{j=1}^{T_m} \lambda \hat{q}_j \gamma_{j+1} (1-\gamma_{j+1}^n).\end{aligned}$$

However, from Corollary 1,

$$\sum_{k=0}^{\infty} \alpha_k p_T(k) = \sum_{j=1}^{T_m} \lambda \hat{q}_j \gamma_{j+1}.$$

Inconsistency results from the nonzero probability of dynamic failure, since the difference, γ_{j+1}^n, is the probability that a task with laxity j will fail to complete in time. If these failed tasks are not rejected or continuously transferred from node to node, thus conserving task flow, and/or the probability of dynamic failure is negligibly small, the summation in (16), $1+\gamma_{j+1}+\gamma_{j+1}^2+\cdots+\gamma_{j+1}^{n-1}$ can be replaced by $\sum_{k=0}^{\infty}\gamma_{j+1}^k = 1/(1-\gamma_j)$, thus $\beta_T = \sum_{j=T}^{T_m} \lambda\hat{q}_j\gamma_{j+1}/(1-\gamma_{j+1})$, and $\sum_{k=0}^{\infty}\beta_k p_T(k) = \sum_{k=0}^{\infty}\alpha_k p_T(k)$. □

To verify the correctness of (18), one has to show that flow conservation holds for the system with the quasi-perfect LS LS scheme.

Corollary 3: Without considering those tasks being rejected and declared not to meet their deadlines, we have

$$\sum_{k=0}^{\infty} \lambda(k) p_T(k) = \lambda$$

for the quasi-perfect LS scheme.

Proof:

$$\begin{aligned}\sum_{k=0}^{\infty} \lambda(k) p_T(k) &= \sum_{k=0}^{\infty}\sum_{j=k}^{T_m} \lambda \hat{q}_j \frac{1-\gamma_{j+1}^n}{1-\gamma_{j+1}} p_T(k) \\ &= \sum_{j=0}^{T_m} \lambda \hat{q}_j \frac{1-\gamma_{j+1}^n}{1-\gamma_{j+1}} \sum_{k=0}^{j} p_T(k) \\ &= \sum_{j=0}^{T_m} \lambda \hat{q}_j (1-\gamma_{j+1}^n) \\ &\simeq \sum_{j=0}^{T_m} \lambda \hat{q}_j \quad \text{if } \gamma_{j+1}^n \simeq 0 \\ &= \lambda.\end{aligned}$$

Inconsistency again results from the nonzero probability of dynamic failure. If this probability (or γ_j^n) is negligibly small and/or the failed tasks are continuously transferred among nodes, task flow will be conserved. □

ACKNOWLEDGMENT

The authors wish to thank Prof. M. Srinivasan at the University of Michigan, and the anonymous reviewers for their valuable comments on an early draft of this paper. Y-C. Chang's assistance in preparing event-driven simulation programs is also gratefully acknowledged.

REFERENCES

[1] K. G. Shin and C.-J. Hou, "Design and evaluation of effective load sharing in distributed real-time systems," in *Proc. Third IEEE Symp. Parallel and Distributed Processing,* Dec. 1991, pp. 670–677. Also *IEEE Trans. Parallel Distributed Syst.,* to be published.

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

[3] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Software Eng.,* vol. SE-12, no. 5, pp. 662–675, 1986.

[4] C. M. Krishna and K. G. Shin, "Performance measures for multiprocessor controllers," in *Performance '83,* A. K. Agrawala and S. K. Tripathi, Eds., North-Holland, 1983, pp. 229–250.

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

[6] M. H. Woodbury and K. G. Shin, "Evaluation of the probability of dynamic failure and processor utilization for real-time systems," in *Proc. IEEE Real-Time Syst. Symp.,* 1988, pp. 222–231.

[7] T. P. Yum and H.-C. Lin, "Adaptive load balancing for parallel queues with traffic constraints," *IEEE Trans. Commun.,* vol. COM-32, pp. 1339–1342, Dec. 1984.

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

[9] T. C. K. Chou and J. A. Abraham, "Distributed control of computer systems," *IEEE Trans. Comput.,* vol. C-35, June 1986.

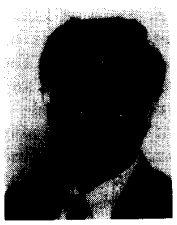
[10] C.-Y. H. Hsu and J. W.-S. Liu, "Dynamic load balancing algorithms in homogeneous distributed systems," in *IEEE Proc. 6th Int. Conf. Distributed Comput. Syst.,* 1986, pp. 216–223.

[11] A. Weinrib and S. Shenker, "Greed is not enough: Adaptive load sharing in large heterogeneous systems," in *IEEE INFOCOM'88–Conf. Comput. Commun. Proc.,* 1988, pp. 986–994.

[12] J. A. Stankovic, "Simulation of three adaptive, decentralized controlled, job scheduling algorithms," *Comput. Networks,* vol. 8, pp. 199–217, 1984.

[13] ——, "An application of Bayesian decision theory to decentralized control of job scheduling," *IEEE Trans. Comput.,* vol. C-34, pp. 117–130, Feb. 1985.

[14] A. B. Barak and A. Shiloh, "A distributed load-balancing policy for a multicomputer," *Software–Practice and Exper.*, vol. 15, no. 9, pp. 901–913, 1985.
[15] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. C-29, pp. 1104–1113, Dec. 1980.
[16] P. Krueger and R. Finkel, "An adaptive load balancing algorithm for a multicomputer," Tech. Rep. 539, Dep. Comput. Sci, Univ. Wisconsin–Madison, Apr. 1984.
[17] J. A. Stankovic, K. Ramamritham, and S. Chang, "Evaluation of a flexible task scheduling algorithm for distributed hard real-systems," *IEEE Trans. Comput.*, vol. C-34, pp. 1130–1141, Dec. 1985.
[18] 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.
[19] T. L. Casavant and J. G. Kuhl, "Analysis of three dynamic distributed load-balancing strategies with varying global information requirements," in *IEEE Proc. 7th Int. Conf. Distributed Comput. Syst.*, 1987, pp. 185–192.
[20] S. Zhou, "A trace-driven simulation study of dynamic load balancing," *IEEE Trans. Software Eng.*, vol. SE-14, pp. 1327–1341, Sept. 1988.
[21] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Trans. Comput.*, vol. C-38, pp. 1110–1123, Aug. 1989.
[22] R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Analysis of the effect of delays on load sharing," *IEEE Trans. Comput.*, vol. C-38, pp. 1513–1525, Nov. 1989.
[23] L. M. Ni, C. W. Xu, and T. B. Gendreau, "A distributed drafting algorithm for load balancing," *IEEE Trans. Software Eng.*, vol. SE-11, no. 10, pp. 1153–1161, 1985.
[24] A. Hać and X. Jin, "Dynamic load balancing in a distributed system using a decentralized algorithm," in *IEEE Proc. 7th Int. Conf. Distributed Comput. Syst.*, Sept. 1987, pp. 170–184.
[25] K. G. Shin and Y.-C. Chang, "Load sharing in distributed real-time systems with state change broadcasts," *IEEE Trans. Comput.*, vol. C-38, pp. 1124–1142, Aug. 1989.
[26] ——, "A coordinated location policy for load sharing in hypercube multicomputers," *J. and Parallel Distributed Comput.*, 1993, to be published.
[27] R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Adaptive load sharing in heterogeneous systems," in *IEEE Proc. 9th Int. Conf. Distributed Comput. Syst.*, 1989, pp. 298–306.
[28] L. M. Ni and K. Hwang, "Optimal load balancing in a multiple processor system with many job classes," *IEEE Trans. Software Eng.*, vol. SE-11, no. 5, pp. 491–496, 1985.
[29] H.-Y. Chang and M. Livny, "Distributed scheduling under deadline constraints: A comparison of sender-initiated and receiver-initiated approaches," in *Proc. IEEE Real-time Syst. Symp.*, 1986, pp. 175–181.
[30] K. J. Lee and D. Towsley, "A comparison of priority-based decentralized load balancing in distributed computer systems," in *Proc. Performance '86*, May 1986, pp. 70–78.
[31] R. Alonso and L. L. Cova, "Sharing jobs among independently owned processors," in *IEEE Proc. 8th Int. Conf. Distributed Comput. Syst.*, 1988, pp. 282–288.
[32] M. H. DeGroot, *Optimal Statistical Decisions.* New York: McGraw-Hill, 1970.
[33] K. G. Shin and Y.-C. Chang, "Load sharing in hypercube multicomputers for real-time applications," in *Proc. 4th Conf. Hypercube, Concurrent Comput., and Appl.*, 1989, pp. 617–622.
[34] S. Ross, *Applied Probability Models with Optimization Applications.* San Francisco, CA: Holden-Day, 1970.
[35] D. Gross and C. Harris, *Fundamentals of Queueing Theory*, second ed. New York: Wiley, 1985.
[36] M. L. Chaudhry and J. G. C. Templeton, *A First Course in Bulk Queues.* New York: Wiley, 1983, ch. 2–3, pp. 58–61, 127–130.
[37] L. Kleinrock, *Queueing Systems, Volume 1: Theory.* New York: Wiley, 1975.
[38] J. F. Kitchin, "Practical Markov modeling for reliability analysis," in *IEEE 1988 Proc. Annu. Reliability and Maintainability Symp.*, 1988, pp. 290–296.
[39] D. Bertsekas and R. Gallager, *Data Networks.* Englewood Cliffs, NJ: Prentice-Hall, 1987.
[40] S. Pulidas, D. Towsley, and J. A. Stankovic, "Embedding gradient estimators in load balancing algorithms," in *IEEE Proc. 8th Int. Conf. Distributed Comput. Syst.*, 1988, pp. 482–490.
[41] J. Hong, X. Tan, and D. Towsley, "A performance analysis of minimum laxity and earliest deadline scheduling in a real-time system," *IEEE Trans. Comput.*, vol. C-38, pp. 1736–1744, Dec. 1989.



Kang G. Shin (S'75–M'78–SM'83–F'92) 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, NY, in 1976 and 1978, respectively.

From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, NY. 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 Professor and Associate Chair of Electrical Engineering and Computer Science (EECS) for Computer Science and Engineering, The University of Michigan, Ann Arbor. He has authored/coauthored over 240 technical papers (more than 100 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 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.

In 1987, Dr. Shin 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. He 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 SYSTEMS, and an Area Editor of *International Journal of Time-Critical Computing Systems.*

Chao-Ju Hou (S'88) was born in Taipei, Taiwan, Republic of China. She received the B.S.E. degree in electrical engineering from National Taiwan University in 1987, the M.S.E. degree in electrical engineering and computer science, and the M.S.E. degree in industrial and operations engineering both from the University of Michigan, Ann Arbor, in 1989 and 1991, respectively.

She is now working as a Research Assistant in the Real-Time Computing Laboratory, and expects to receive the Ph.D. degree in electrical engineering and computer science from the University of Michigan, Ann Arbor, in 1993. Her research interests are in the areas of distributed and fault-tolerant computing systems, queueing systems, estimation and decision theory, and performance modeling/evaluation.