DESIGN AND EVALUATION OF EFFECTIVE LOAD SHARING IN DISTRIBUTED REAL–TIME SYSTEMS [1]

Kang G. Shin and Chao–Ju Hou

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

ABSTRACT

In a distributed real-time system, uneven task arrivals temporarily overload some nodes while leaving others idle or underloaded. Consequently, some tasks may miss their deadlines even if the overall system has the capacity to meet the deadlines of all tasks. An effective load sharing (LS) scheme is proposed as a solution to this problem.

Upon arrival of a task at a node, the node determines whether or not the node can complete the task in time under the minimum–laxity–first–served policy. If the task cannot be guaranteed or if guarantees of some other tasks are to be violated due to the addition of this task to the existing schedule, the node looks up the list of *loss–minimizing decisions*, and determines the best node among a set of nodes in its physical proximity, called its *buddy set*, to which the task(s) may be transferred. This list of decisions is periodically updated using Bayesian decision analysis and prior/posterior state distributions. These probability distributions are derived from the information collected via time–stamped state–region change broadcasts within each buddy set. By characterizing the inconsistency between a node's 'observed' state and the corresponding true state with prior and posterior distributions, the node can first estimate the states of other nodes, and then use them to reduce the probability of transferring a task to an 'incapable' node. Moreover, the use of prior/posterior distributions and Bayesian analysis has made the proposed scheme robust to the variation of design parameters that usually require fine–tuning for adaptive LS.

The performance of the proposed scheme is evaluated via simulation along with five other schemes: no LS; LS with state probing, random selection and focused addressing; and perfect LS. The proposed scheme is shown to outperform all but perfect LS scheme in (i) meeting task deadlines and (ii) tolerating the delays in state collection and task transfer. The impact of statistical fluctuations in task arrival patterns on the performance of the proposed scheme (in particular, the Bayesian decision analysis part) is also analyzed via simulation to show the robustness of the proposed scheme over a wide range of task arrivals.

Index Terms — Deadlines, real-time systems, load sharing, location and transfer policies, Bayesian decision theory, performance evaluation.

[1]The work reported in this paper was supported in part by the Office of Naval Research under Grant N00014–92–J–1080, and the National Science Foundation under Grant No. 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.

1 Introduction

The availability of inexpensive, high–performance processors and memory chips has made it attractive to use distributed computing systems for real–time applications. However, since task arrivals are usually uneven among the nodes of a distributed system and/or the processing power varies from node to node, some nodes may get temporarily overloaded while others are left underloaded/idle. Livny and Melman [1] showed that in a network of autonomous nodes, with a large probability at least one node is idle while many tasks are being queued at other nodes. Thus, we need to develop an effective method that will enable idle/underloaded nodes to share the loads of overloaded ones.

Load sharing (LS) in a distributed real–time system is different from that in a general-purpose system in that the latter tries to either achieve perfect load balancing among the nodes and/or minimize average task response time, whereas the former is intended to minimize the probability of failure to complete each real–time task in time — this was termed the *probability of dynamic failure*, P_{dyn}, in [2, 3]. Upon arrival of a real–time task, each node determines whether or not it can complete this task in time. If it can, the node will execute the task locally; otherwise, some other 'capable node' will be chosen to execute the task [4, 5, 6, 7, 8, 9, 10, 11]. By 'capable node,' we mean a node with unused resources enough to complete transferred–in task(s) in time. As was discussed in [6], LS in a distributed system is dictated by two basic policies: the *transfer policy* for determining when to transfer a task, and the *location policy* for determining where to transfer the task. In the context of real–time applications, the transfer policy determines whether or not a task can be guaranteed (i.e., completed in time) locally, and the location policy determines which other node is most likely to guarantee the task to be transferred.

According to the properties of these two policies, LS schemes can be classified into three categories: deterministic, probabilistic and dynamic/adaptive [12, 4, 8]. A deterministic approach allows an overloaded node to transfer unguaranteed tasks with a fixed pattern, e.g., all unguaranteed tasks on node i are transferred to node j. A probabilistic approach, on the other hand, transfers tasks with pre-specified probabilities, e.g., an overloaded node i will transfer its unguaranteed tasks to node j with probability P_{ij}. By contrast, an adaptive approach uses state information for their location policy. The state of a node may be the number (or queue length, QL), or the cumulative task execution time (CET), of tasks queued for execution on the node, the number and type of available resources, or a function or combination thereof. The node makes LS decisions based on the information collected via either periodic/aperiodic state broadcasting [8, 13, 14] or state probing or bidding [9, 11, 15, 16, 17].

Both deterministic and probabilistic approaches do not use the state information, and thus, cannot react to dynamic situations. Because an adaptive approach can adapt itself to dynamically–changing conditions, it is naturally expected to outperform non-adaptive approaches in meeting task deadlines. However, the required state probing/broadcasting could incur significant communication overheads, thus delaying the execution of tasks to be transferred. Moreover, the collected state information may be out–of–date due to the delay in collecting it [18]. That is, a node's *observed* states of other nodes may be different from their *true* states at the time of making LS decisions. This difference often degrades the performance of adaptive LS as was analyzed in [16, 17]. (Note, however, that the authors of [16, 17] did not propose any means to alleviate/eliminate this problem.)

To reduce the performance degradation caused by the delays in state collection and/or task transfer, we propose a new LS scheme using Bayesian decision theory as well as the concept of buddy sets, preferred lists, and state–change broadcasts in [14]. The basic ideas used here will be detailed in Section 2. Using several performance metrics, such as P_{dyn}, the task transfer–out ratio, and maximum system utilization, we comparatively evaluate the proposed scheme along with five other schemes: no LS; LS with state probing, focused addressing and random selection; and perfect LS. Our numerical results indicate that the proposed scheme outperforms all but the perfect LS scheme in minimizing P_{dyn}.

LS approaches proposed for general–purpose distributed systems are designed to minimize average system sojourn time or average response time, instead of minimizing P_{dyn}. Moreover, QL is usually used as the state of a node, which is obviously inadequate for real–time applications if task execution times are not identical. In this paper, we not only tailor both the transfer and location policies to handle real–time applications, but also use CET as the state of each node and P_{dyn} as the performance metric. Furthermore, we use Bayesian analysis to reduce the performance degradation caused by the delays in collecting state information and transferring tasks — which, despite its importance, is seldom addressed in literature (except for [16, 17]). We also study via simulation the potential impact of the time-varying behavior of task arrivals on the performance of the proposed scheme (especially, on the Bayesian analysis part).

The rest of this paper is organized as follows. The basic ideas of the proposed scheme are described in Section 2. The Bayesian decision model used is presented in Section 3. How both the components of the Bayesian decision model and the concepts presented in [14] can be accommodated into our LS scheme is also described there. Section 4 describes how each node constructs prior and posterior probability distributions, and updates loss–minimizing decisions. In Section 5, we evaluate via simulation (1) the performance of the proposed LS scheme along with five other schemes and (2) the effect of bursty task arrivals on the performance of the proposed LS scheme. The paper concludes with Section 6.

2 Basic Ideas of the Proposed Scheme

In order to reduce the overheads associated with state collection and task transfer, the LS scheme in [14] requires each node to collect and maintain the state information of only those nodes in its physical proximity, called a *buddy set*. When a node cannot complete a real–time task in time, only those nodes in its buddy set are considered for transferring this task. In [14] four state regions determined by three thresholds of QL are used to characterize the workload of each node: underloaded, medium–loaded, fully–loaded, and overloaded. A node will broadcast the change of state region to the nodes in its buddy set only when it switches from underloaded to fully–loaded and vice versa. The state information kept at each node is thus up–to–date as long as the broadcast delay is not significant. Based on the topological property of the system, each node orders the nodes of its buddy set into a *preferred list* such that a node is the k–th preferred node of one and only one other node, where k is some integer [19]. When a node is unable to complete a task in time, it will transfer the task to the first 'capable node' found in its preferred list. That is, the preferred lists are used as an effective means of selecting a receiver among several possible candidate nodes while minimizing the probability of more than one overloaded node simultaneously sending tasks to the same underloaded node.

Communication delays may still occur and thus degrade system performance unless the size of buddy set is kept very small, in which case the LS capability of the whole system may not be fully utilized. Thus, Bayesian decision theory is used to counter the communication delay problem as shown in Fig. 1.

Fig. 1 shows the actions that the scheduler on each node should take for the following four cases:

At each node:

When a task T_i with execution time E_i and laxity D_i arrives at the node:

    determine the position, j_p, in the task queue Q†such that $D_{j_p-1} \leq D_i \leq D_{j_p}$;
    if current_time + $\sum_{k=1}^{j_p-1} E_k \geq D_i$ **then**
     begin
        receiver_node := table_lookup($\underline{x}$:observation, D_i:laxity)‡;
        transfer task T_i to receiver_node;
     end
    else
     begin
        queue task T_i in position j_p;
        for $k = j_p + 1, length(Q)$
          begin
            if current_time + $\sum_{\ell=1}^{k-1} E_\ell \geq D_k$ **then**
             begin
                receiver_node := table_lookup($\underline{x}$:observation, D_k:laxity);
                dequeue and transfer T_k to receiver_node;
             end
          end
        if current_CET crosses TH_{2k}, $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$, **then**
            /* $TH_1, \cdots, TH_{K-1}$ are thresholds */
            broadcast the state–region change to all nodes in its buddy set;
     end

When a broadcast message arrives from node i, $1 \leq i \leq n$:

    update observation of node i's state, x_i;
    record the (observation, true state) pair needed for constructing probability
        distributions;

When current_CET crosses TH_{2k}, $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$:

    broadcast the state–region change to all nodes in its buddy set;

At every T_p clock ticks:

    update the probability distributions and the table of loss–minimizing
        decisions;

†The task queue Q is ordered by task laxities.

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

Figure 1: Operation of the task scheduler on each node.

(a) When a new task arrives,

(b) When a state–region change broadcast is received,

(c) When current CET crosses TH_{2k}, $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$, and

(d) At every T_p clock ticks.

Those tasks already queued at a node are sorted by their laxities and executed on a minimum–laxity–first–served (MLFS) basis. (Note that the laxity of a task is defined as the latest time a task must start execution in order to meet its deadline.) Upon arrival of a real–time task at a node, the scheduler checks if the CET on that node contributed by those tasks with laxity smaller than this task is less than, or equal to, the laxity of the new task. If it is not, the new task has to be transferred, and the node's task queue remains unchanged; if it is, the new task is inserted in the task queue, and if this insertion leads to violation of existing guarantees, those tasks whose guarantees are violated need to be transferred to other capable nodes. By 'guarantee', we mean the node has enough resources to complete the task in time upon its arrival. A granted guarantee may be deprived later because of the arrivals of tighter–laxity tasks under the MLFS policy.

K state regions obtained from $K-1$ thresholds, TH_1, TH_2,...,TH_{K-1}, are used to describe the workload of each node.[2] 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 $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$. The reason for not broadcasting the change of state region whenever a node's load crosses any threshold is to reduce the network traffic resulting from region–change broadcasts. Moreover, the reason for not combining two adjacent state regions into one and then broadcasting the change of state region whenever a node's load crosses any threshold is to include finer information in each broadcast and thus construct more accurate posterior distributions.

By collecting time–stamped state samples and by keeping track of the corresponding observations at the times these samples were taken, each node can construct the prior/posterior distributions. These distributions characterize the inconsistency between the node's observed and true states of other nodes, and are used to periodically (once every T_p clock ticks) update the loss–minimizing decisions with Bayesian theory. As will become clear, the undesirable effects of the delay in broadcasting state–region changes/transferring tasks are eliminated

[2] K is a tunable parameter which will be discussed later.

by using these prior/posterior distributions. Whenever a node cannot guarantee a task, the node's scheduler looks up the list of loss–minimizing decisions, and choose — based on the current state information — the best candidate node for transferring this task such that the expected loss is minimized with respect to the conditional (or posterior) probability distribution.

3 Bayesian Decision Model

Conceptually, the task scheduler of a node can be modeled as a Bayesian decision maker. In what follows, we shall describe how the proposed LS scheme can be cast into a Bayesian decision model.

3.1 Preliminaries

The Bayesian statistical structure is a very powerful modeling tool when one has to make decisions based on some observations. The elements of a Bayesian decision problem are a parameter space (a space of state of nature) Ω, a decision space D, and a real–valued loss function L which is defined on the product space $\Omega \times D$ [20, 21]. For any point $(\omega, d) \in \Omega \times D$, the quantity $L(\omega, d)$ represents the loss when the value of the outcome W of the space Ω is ω and d is the decision chosen.

If P is any given probability distribution of the parameter W, then for any decision $d \in D$, the expected loss or risk, $\zeta(P, d)$, is given by

$$\zeta(P, d) = \int_{\Omega} L(\omega, d) dP(\omega). \tag{3.1}$$

It will be assumed that the integral in Eq. (3.1) is finite for every $d \in D$.[3] We now want to choose a decision d which minimizes the risk $\zeta(P, d)$. The *Bayes risk* $\zeta^*(P)$ is thus defined to be the greatest lower bound for the risks $\zeta(P, d)$ $\forall d \in D$, i.e.,

$$\zeta^*(P) = \inf_{D} \zeta(P, d). \tag{3.2}$$

Any decision d^* whose risk is equal to the Bayes risk is called a *Bayes decision* with respect to the distribution P.

In many decision problems like the one we are going to discuss, before choosing a decision from D, we observe the value of a random variable X that is related to the parameter W.

[3] Any decision d for which this assumption is not true can usually be eliminated from the set D.

The observation of X provides us with some information about the value of W which may be useful in making a good decision. The essential component of problems of this kind is, in addition to a parameter space Ω, a decision space D, and a loss function L, a family of sampling functions $\{f(. \mid \omega), \omega \in \Omega\}$ of observation X. Let S denote the sample space of all possible values of X. With the family of sampling functions and the (prior) probability distribution, P, of W, we can calculate the conditional distribution of W given X, $P_{W|X}$, as:

$$P_{W|X=x}(\omega) = \frac{P_W(\omega)P_{X|W}(x \mid \omega)}{\int_\Omega P_W(\omega)P_{X|W}(x \mid \omega)d\omega}. \tag{3.3}$$

Now, we must choose a decision function δ which specifies, for every possible value $x \in S$, a decision $\delta(x) \in D$ with the expected loss given the observation x as:

$$\zeta(P_{W|X=x}, \delta(x)) = \int_\Omega L(\omega, \delta(x))dP_{W|X=x}(\omega). \tag{3.4}$$

Note that Eq. (3.4) is almost the same as Eq. (3.1) except that P has been replaced by $P_{W|X=x}$. That is, given the observation of X, the decision problem remains unchanged except that the distribution of W has changed from the prior to the posterior distribution. Thus, any minimizing decision $d^*(x)$ is simply a decision which yields the smallest expected loss under the conditional distribution of W when the observed value of X is x. In other words, $d^*(x)$ is a Bayes decision against the conditional distribution of W when $X = x$.

3.2 Components of the Bayesian Decision Model

This subsection describes how to apply Bayesian decision theory to adaptive LS, and how to accommodate both the components of the statistical model and the concept of [14] into our scheme.

3.2.1 Parameter Space

The parameter space is defined as $\Omega = \Omega_1 \times \Omega_2 \times ... \times \Omega_n$, where n is the number of nodes in a buddy set, and Ω_i is the parameter space for node i. Note that the size of state space and the overhead of broadcasting state–region changes are greatly reduced by using the buddy sets. The parameter space, Ω_i, may be defined by QL, CET, resource available time (RAT) on node i, or a combination thereof, depending on task characteristics and performance requirements. For example, if all tasks have an identical execution time, QL suffices to express each node's workload; otherwise, CET must be used. Since execution time varies from task to task, the state of a node is defined to be its CET. The dimension of the state can be augmented if more than one resource are needed to execute tasks.

3.2.2 Probability Distribution on Parameter Space

The probability distribution on parameter space is the joint probability distribution of Ω_i's, e.g., $P_W(\underline{\omega}) = P_W(\omega_1, \omega_2, ..., \omega_n)$, where ω_i is the CET of node i and n is the number of nodes in a buddy set. The marginal probability distribution on Ω_i, P_{W_i}, can be obtained from P_W by integration. We construct these probability distributions by collecting state samples through region–change broadcasts (to be discussed in Section 4).

3.2.3 Set of Available Decisions

The set of available decisions is $D = \{d_1, d_2, ..., d_n\}$, where d_i denotes the decision to move one task from the current node to node i. Other options are also possible. For example, if a locally unguaranteed task is extremely important, then one may want to move it simultaneously to two or more nodes so that the probability of dynamic failure can be minimized. In such a case, a decision d_{ij} is added to the set of available decisions, which denotes the transfer of a task to both node i and node j.

3.2.4 Set of Loss Functions

The set of loss functions is defined as $\{L^{T_d}, T_d \in (0, T_{max}]\}$, where L describes the 'loss' resulting from each combination of state and decision, given that the laxity — which equals deadline – execution time – current time — of a locally unguaranteed task is T_d. T_{max} is the largest task laxity in the system. If P_{dyn} is the main concern, the loss function may be defined as:

$$L^{T_d}(\underline{\omega}, d_i) = \delta(\omega_i - T_d) = \begin{cases} 1 & \text{if } \omega_i - T_d \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\delta(x)$ is the unit step function. In such a case, minimizing the expected loss is equivalent to minimizing the probability of dynamic failure. The loss function can also be defined as

$$L^{T_d}(\underline{\omega}, d_i) = \omega_i - T_d$$

if the task needs to be executed not only before its deadline but also as early as possible.[4]

[4] L^{T_d}, could be negative in this case; the more negative L^{T_d}, the more early the task is executed.

3.2.5 Sample Space of Observation

The sample space of observation, S, is the set of all possible observations. Specifically, $S = S_1 \times S_2 \times \cdots \times S_n$, where S_i is obtained by dividing the parameter space W_i for each node i into the K regions determined by $K-1$ thresholds, TH_1, TH_2,..., TH_{K-1}. Node i is said to be in the k-th region if $TH_k \leq \omega_i < TH_{k+1}$, where $k \geq 0$, and $TH_0 \triangleq 0$.

Note that the knowledge of a node's state region is not sufficient to determine accurately its capability of guaranteeing arbitrary tasks. For example, a node with its state in a high–numbered state region may still be able to guarantee an arriving task with a large laxity, whereas a task with a small laxity may not be guaranteed even by a first–region node if the CET on that node is greater than the task's laxity. Thus, unlike in [14, 22, 5, 6], these thresholds only serve as reference points, rather than indicating a node's capability of meeting task deadlines. As will be discussed in Section 5, the performance of the proposed scheme is rather insensitive to the choice of threshold values.

Each node will broadcast a time–stamped message, informing all the other nodes in its buddy set of a state–region change whenever its state crosses TH_{2k}, $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$. Upon receipt of a region–change broadcast, every node in the buddy set will update its observation of the broadcasting node accordingly. The delay in broadcasting a region–change may cause inconsistency between the observed and true states of a node. We will characterize this inconsistency by constructing prior/posterior distributions (to be discussed in Section 4). So, based on the observation x_i, a node can estimate the state of node i by using the prior/posterior distributions constructed from the samples collected through time–stamped region–change broadcasts.

3.2.6 Family of Sampling Functions

The family of sampling functions, $\{f_{X|W}(. \mid \underline{\omega}), \underline{\omega} \in \Omega\}$, describes the conditional probability distribution of the observation X given the state $W = \underline{\omega}$. These probability distributions are derived from the samples gathered through time–stamped region–change broadcasts. With the prior probability distribution P_W and these sampling functions $f_{X|W}$, one can derive the posterior probability distribution $P_{W|X}$ by using the Bayes rules [20] that is needed to compute the expected loss with observations.

4 Region–Change Broadcasting, Prior/Posterior Probability Distributions and Loss–Minimizing Decisions

As mentioned earlier, the delay in region–change broadcasts may cause the collected information to be out–of–date. For example, consider the following scenario: after broadcasting a state–region change, say from 3 to 1, node i switches back to region 3 due to the arrival of new tasks and/or transferred–in tasks.[5] Upon receipt of the broadcast from node i, node j may decide to send a task to node i, since it is unaware that node i has switched back to region 3 shortly after broadcasting the 3→1 region–change. If node j, instead of hastily believing in what it observed, can compute the probability that node i is indeed capable of guaranteeing task(s) and decide whether or not to send the task to node i, then the probability of dynamic failure could be significantly reduced. To this end, we shall characterize the inconsistency between the observed and true states with prior/posterior distributions.

The first step is to construct both the probability distribution on the parameter space and the conditional probability distribution of an observation. These two distributions, in general, vary over both nodes and time in a dynamic environment. Thus, to monitor the dynamics of the system, each node must collect state samples on-line and construct these distributions from the samples gathered via region–change broadcasts. The methods for collecting state samples, constructing probability distributions, and deriving loss–minimizing decisions are discussed in the following subsections.

4.1 Collection of State Samples

Whenever a node's state crosses TH_{2k} ($1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$), the node will broadcast, to all the other nodes in its buddy set, a time-stamped message which contains node number i, the state ω_i^b before the change of state region, the state ω_i^a after the change, and the time t_0 when ω_i^b was sampled. When the message broadcast by node i arrives at node j, node i's state ω_i^b can be recovered by node j using the node number field and the state field, from which P_W can then be calculated. Node j can also trace back to find out its observation x_i at time t_0. This observation x_i is node j's observation of node i's state at the time when node i was actually in state ω_i^b. x_i's along with ω_i^b's are used to construct $f_{X|W}$. (Here we assume that the node clocks are synchronized to establish a global time-base. A scheme for

[5] These tasks may have been sent by other nodes before the broadcast, but arrived at node i after the broadcast due to task–transfer delay.

achieving this synchronization is presented in [23].) Any inconsistency between ω_i^b and x_i at time t_0 is characterized by this probability distribution. The only effect of the delays in task transfers and region–change broadcasts is that messages (tasks) may not arrive at a node immediately after their broadcast (send), and thus, may become obsolete upon their arrival at other nodes. The correctness of all samples gathered is, however, not affected by these delays. Besides, ω_i^a sent by node i at time t_0 is considered as node j's new observation of node i at the time this message is received, rather than at time $t_0^+ > t_0$.

A primary advantage of region–change broadcasts over periodic state broadcasts is the elimination of the need to determine an "optimal" exchange period — a very difficult task since it depends on workload characteristics, and has to weigh the tradeoff between the resulting increase in network traffic and the negative effect of using out–of–date information. Moreover, as we shall see, the threshold values have only minor effects on system performance (as a result of using Bayesian analysis).

4.2 Derivation of Probability Distributions

Each node updates, once every T_p units of time, the probability distributions using all the samples gathered so far, and re-calculates the loss–minimizing decisions. T_p should be chosen to reflect the fluctuation of system load and the number of samples required for the specified level of confidence in the results obtained.

The general rule for updating the probability distribution of W is

$$P_U = aP_T + (1-a)P_O$$

where P_U is the updated probability distribution, P_T is calculated from the samples gathered over the last T_p units of time, and P_O is the old probability distribution. That is, the updated probability distribution is a weighted sum of the distribution calculated from the samples gathered within the last T_p units of time and the old probability distribution. The ratio a $(0 < a \leq 1)$ represents the tradeoff between obtaining better averages and reflecting load changes. One may increase (decrease) a if system load varies rapidly (slowly). The same rule may be applied to update the sampling functions, $f_{X|W}$.

Non-informative probability distributions (e.g., uniform distributions) or some default probability distributions (obtained from previous experiences) may be used as the initial distribution of W and the sampling functions. According to our simulation results, the performance of the proposed scheme is found to be rather insensitive to the choice of an initial probability distribution. Each node may initially rely on the preferred list for LS decisions. This is because both prior and posterior distributions will be iteratively updated as time goes on, and usually represents the true system characteristics after two or three updates. Besides, if the task arrival pattern on each node does not change drastically with time, the probability updating process need not be executed often once the probability distributions are well–tuned.

4.3 Calculation of Loss–Minimizing Decisions

With the prior distribution of W and the sampling function, $f_{X|W}$, one can calculate the posterior distribution $P_{W|X}$ using the Bayes rule (Eq. (3.3)). For each possible observation $\underline{x} \in S$ and for each possible laxity $T_d \in (0, T_{max}]$, a node then computes the expected loss associated with the decision d_i given the observation $\underline{x}$ and the laxity T_d as:

$$\zeta^{T_d}(P_{W|X=\underline{x}}, d_i) = \int_{\Omega} L^{T_d}(\underline{\omega}, d_i) dP_{W|X}(\underline{\omega}) \tag{4.1}$$

for $i = 1, ..., n$. The decision $d_i = \delta^{T_d}(\underline{x})$ that yields the minimum expected loss is chosen as the optimal decision given the observation $\underline{x}$. A tie will be broken by choosing, from the preferred list, the first d_i with the minimum expected loss.[6] Because of the way $L^{T_d}(\underline{\omega}, d_i)$ was defined and the assumption that W_i is stochastically independent of the state of node j for $j \neq i$ [6], the computation of the expected risk, $\zeta^{T_d}(P_{W|X=\underline{x}}, d_i)$, depends only on the marginal probability distribution, $P_{W_i|X_i}(\omega_i)$. That is, if $L^{T_d}(\underline{\omega}, d_i) = \delta(\omega_i - T_d)$, then

$$\begin{aligned}
\zeta^{T_d}(P_{W|X=\underline{x}}, d_i) &= \int_{\Omega} \delta(\omega_i - T_d)\, p_{W|X=\underline{x}}(\underline{\omega}) d\underline{\omega} \\
&= \int_{\Omega_i} \delta(\omega_i - T_d)\, p_{W_i|X=\underline{x}}(\omega_i) d\omega_i \\
&= \int_{\Omega_i} \delta(\omega_i - T_d)\, p_{W_i|X_i=x_i}(\omega_i) d\omega_i \\
&= P_{W_i|X_i}(W_i > T_d \mid X_i = x_i).
\end{aligned} \tag{4.2}$$

In other words, the expected loss of adopting decision d_i given the observation $\underline{x}$ and the task laxity T_d is the probability that node i's CET is greater than T_d. The second equality in Eq. (4.2) follows from the property of total probabilities, and the third equality results from the assumption that W_i is stochastically independent of the observation X_j of node j, $j \neq i$, i.e., $p_{W_i|(X_1, X_2, ..., X_n)}(\omega_i) = p_{W_i|X_i}(\omega_i)$. Similarly,

$$\zeta^{T_d}(P_{W|X=\underline{x}}, d_i) = \int_{\Omega_i} (\omega_i - T_d) \cdot p_{W_i|X_i=x_i}(\omega_i)\, d\omega_i,$$

[6]The nice property (of the preferred lists) in distributing unguaranteed tasks among capable nodes is thus maintained in the proposed scheme.

if $L^{T_d}(\underline{\omega}, d_i) = \omega_i - T_d$. The set of loss–minimizing decisions, $\{\delta^{T_d}(\underline{x}) : \underline{x} \in S, T_d \in (0, T_{max}]\}$, is a list of decisions to choose for each possible observation and each possible task laxity. Once these calculations are completed, a task scheduler only needs to look up a table when determining a LS decision for a given observation $\underline{x}$.

5 Performance Evaluation

To demonstrate the effectiveness of the proposed LS scheme, we simulated it under the assumption that inter-arrival times of external tasks are exponentially distributed. Note that periodic tasks constitute the "base load" of a real-time system and are usually *statically* allocated to the nodes in the system as shown in [24]. A LS scheme is then used to dynamically distribute aperiodic tasks as they arrive, and their arrival is known to be well modeled as a Poisson process. After their initial allocation, periodic tasks, albeit rare, may be redistributed subject to aperiodic task arrivals and the current system-wide state. In such a case, a Poisson task arrival model may prove appropriate for assessing the performance of the proposed LS scheme. Note, however, that the proposed LS scheme is not restricted to Poisson models.

The proposed scheme and three other LS schemes are comparatively evaluated. The schemes under consideration differ in the way a node treats locally unguaranteed tasks as follows:

- 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.
- The random selection scheme: each locally unguaranteed task is sent to a randomly selected node.
- 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 node itself becomes the focused node.) Meanwhile, the node also sends request–for–bid (RFB) messages to all the other nodes in the system, indicating that bids (which contain 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, and the received bids are used to locate the receiver nodes for those tasks, if any, whose guarantees become invalid as a result of accepting the transferred task. 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 (or any of those tasks whose guarantees are violated due to the acceptance of the transferred tasks), 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 until it actually arrives. When a task arrives at a node whose bid has already been accepted, the node will check again whether or not the task can be guaranteed. This is a simplified version of the scheme proposed in [11, 15]. It also differs slightly from that of [11, 15] in the way a node chooses the focused node. The authors of [11, 15] 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 node(s) for transferring tasks.

- The proposed scheme: a node sends each unguaranteed task to another node in its buddy set based on a technique that combines preferred lists, state–region change broadcasts, and Bayesian analysis.

These schemes are compared with one another as well as with two other baseline schemes. The first baseline scheme assumes no load sharing, while the second is an ideal scheme where each node has complete information on the workload of other nodes without incurring any overhead in collecting it[7].

A 16–node regular system[8] is used as an example for the simulations. For convenience, all time–related parameters are expressed in *units of average task execution time*, $E(R)$. The size of buddy set is chosen to be 10, since the performance improvement by increasing it beyond 10 was shown in [14] 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 [6]. The computational overhead for each bidding, state probing, region–change broadcast, and probability distribution update is assumed to be 1, 1, 1, and 2 % of $E(R)$, respectively.

[7]This cannot, however, be modeled as an M/M/n queue, because of the transfer policy used and the fact that the service time is fixed at 1.0, as compared to the perfect load sharing in [6].

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

Each communication medium/link is equipped with buffers, and transferred tasks or broadcast messages are queued and/or transmitted in order of their arrival. No priority mechanism regulates the transmission over the medium, i.e., a FCFS rule is assumed. Unless specified otherwise, the delay associated with each task transfer is assumed to be 10 % of the execution time of the task being transferred. The queueing delay due to task transfers, region–change broadcasts, requests and responses for bids, and state probes dynamically changes with system load and traffic, and is modeled as a linear function of both (1) the average external task arrival rate and (2) the number of tasks/messages queued in the particular medium/link. (The linear coefficients are denoted as queueing delay coefficients.)

Let q_i $(1 \leq i \leq m)$ and $\hat{q}_j$ $(0 \leq j \leq T_{max})$ represent respectively the probability that an external (local) task requires i units of time to execute and that a task has laxity of j units of time. For notational convenience, $\{e_1, e_2, ..., e_k\}_{\{q_{e_1}, q_{e_2}, ..., q_{e_k}\}}$ is used to denote the task set in which a task requires execution time e_i with probability q_{e_i}, $\forall\, i$. If $q_{e_i} = q \;\forall\, e_i$, then $\{q_{e_1}, q_{e_2}, ..., q_{e_k}\}$ is condensed to q. Similarly, $\{\ell_1, \ell_2, ..., \ell_n\}_{\{\hat{q}_{\ell_1}, \hat{q}_{\ell_2}, ..., \hat{q}_{\ell_n}\}}$ is used to describe the distribution of task laxity. The simulation was carried out for a task set with the external task arrival rate on each node varying from 0.2 to 0.9, the ratio of $\frac{e_{j+1}}{e_j}$ $(1 \leq j \leq k-1)$ varying from 2 to 10 units of average execution time,[9] and the ratio of of $\frac{\ell_{j+1}}{\ell_j}$ $(1 \leq j \leq n-1)$ varying from 2 to 6. Due to space limitation, we present only representative results. In spite of a large number of possible combinations of task arrival rates, task execution time distributions, and task laxity distributions, the results are found to be quite robust in the sense that the conclusion drawn from the performance curves for a task set with the given task execution and laxity distributions is valid over a wide range of combinations of task execution time and laxity distributions.

For each combination 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 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, frequency, or fraction value if task transfer–out ratio, frequency of broadcasts/state probes, or fraction of idle time, is the measure.[10]

We first determine the tunable parameters used in the proposed scheme. Second, we

[9] For convenience, $E(R)$ is normalized to 1.

[10] 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.

evaluate and compare different schemes with respect to several important performance metrics obtained from the simulations. Then, we analyze the effect of (1) varying processing/communication overheads, (2) using FCFS (instead of MLFS) as the local scheduling policy, (3) excluding the Bayesian decision analysis from the proposed scheme, and (4) statistical fluctuations in task arrival (representing bursty task arrivals) on the performance of the proposed LS scheme.

5.1 Determination of Tunable Parameters in the Proposed Scheme

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

- The choice of a and T_p depends on the application–dependent variation of workload.
- The number and values of thresholds must be determined by optimizing the tradeoff between the resolution of state–region division and the overhead of the resulting region–change broadcasts. It is impossible to determine the optimal number and values of state–region thresholds without a closed–form expression for this tradeoff. Moreover, the optimal number and values of thresholds depend on both the laxity and execution time distributions of the task set.

Thus, we shall determine the tunable parameters for each task set with the following two steps:

S1. We first fix all but one parameter of interest at a time, and obtain the performance curve as a function of this parameter from which its optimal value can be determined. Next, we vary another parameter of interest while keeping the first parameter fixed at its optimal value and the rest of the parameters fixed at their originally–chosen values. This process will be repeated until all the parameters have been varied.

S2. Since a different order of examining parameters in **S1** may lead to different results, there may may be more than one parameter set from which we choose the one with the smallest P_{dyn} and at the same time, reasonably small processing/communication overheads (e.g., the processing power used for updating distributions and calculating Bayesian decisions, the frequency of region–change broadcasts, or the task transfer–out ratio) as the 'optimal' parameter set.

The sets of parameters obtained through the above two steps may not be globally optimal, but our simulation results have shown them to yield good results as compared to other schemes. Moreover, as will be shown later, our simulation results indicate that the proposed scheme is robust to the variation of the tunable parameters. Thus, the result with a set of sub-optimal parameters can still give a representative performance of the proposed scheme.

Summarized below are some of the important findings in determining the tunable parameters. The task set with the task arrival rate $\lambda = 0.8$, the distribution of task execution time $ET = \{0.4, 0.8, 1.2, 1.6\}_{0.25}$, and the distribution of task laxity $L = \{1.0, 2.0, 3.0\}_{1/3}$ is arbitrarily chosen for an illustrative purpose.

Probability Update Interval T_p and Probability Update Ratio a: Frequent probability updates generate more accurate distributions and thus provide each node with better knowledge of the inconsistency between its observation of other nodes and the corresponding true states. This benefit must be weighed against the associated overheads. As shown in Table 1, P_{dyn} for the proposed scheme with the threshold pattern {0.2, 0.6, 1.0} decreases first as T_p increases from 10 to 100 (in units of the average task execution time), and then increases as T_p increases further beyond 100. (The task set with $\lambda = 0.4$ exhibits a similar behavior except that T_p which results in the smallest P_{dyn} is slightly increased.) The probability update ratio a is chosen to be 0.5, since the task arrival rate is constant over time, and thus, the distribution constructed from the state samples gathered before is as good as that constructed from the state samples collected over the last T_p units of time.

Selection of the default probability distributions (or equivalently, the default risk function ζ^{T_d} in Eq. (4.2)) at the system startup is shown to have only minor effects on system performance as long as T_p is properly chosen. So, the initial posterior distribution of CET is chosen to be uniform. The default risk function can then be expressed as:

$$\begin{aligned}
\zeta^{T_d}(P_{W|X=\underline{x}}, d_i) &= P_{W_i|X_i}(W_i > T_d \mid X_i = x_i) \\
&= \begin{cases} 0 & T_d \geq TH_{x_i+1} \\ 1 & T_d < TH_{x_i} \\ (TH_{x_i+1} - T_d)/(TH_{x_i+1} - TH_{x_i}) & \text{otherwise} \end{cases}
\end{aligned}$$

where $i \geq 0$ and $TH_0 \stackrel{\triangle}{=} 0$.

<u>**Threshold Patterns**</u>**:** We first analyze the effect of changing the resolution of state–space division with the number of regions fixed at 4. As shown in Table 2 (a), P_{dyn} decreases as the interval between two thresholds gets smaller, which will henceforth be called the *threshold interval*. (For convenience, all threshold intervals are assumed to be identical; even if this assumption does not hold, the following discussion will remain the same except for more complicated descriptions.) However, the decrease in P_{dyn} becomes insignificant when the threshold interval gets smaller than the least task execution time within the task set. Thus, the threshold interval must not be smaller than the least task execution time. Once the threshold interval is selected, one can analyze the effects of changing TH_1 (and thus, other thresholds). It is shown in Table 2 (b) that P_{dyn} decreases as TH_1 decreases. The change in P_{dyn} again becomes insignificant as TH_1 gets smaller than the least task execution time. Note that in our proposed scheme, selecting $TH_1 <$ the least task execution time $< TH_2$ is essentially equivalent to implementing the shortest queue policy in [6] except that the ways of collecting state information are different (the latter with state probing, and the former with region–change broadcasting). Thus, a modified shortest queue policy (or $TH_1 \leq$ the least task execution time) is preferred for the proposed scheme to minimize P_{dyn}.

One interesting phenomenon is that the frequency of region–change broadcasts is relatively high when the thresholds coincide with the task execution times. This is because the acceptance/completion of a task makes a node's CET easily cross a certain threshold, thus resulting in an excessive number of region–change broadcasts. Such type of thresholds are ruled out. Thus, to implement the shortest queue policy while avoiding excessive broadcasts, TH_1 must be slightly smaller than the least task execution time.

<u>**Number of State Regions**</u>**:** To analyze the effects of the number of state regions on system performance, we ran simulations while changing the number of regions from 2 to 6. A node broadcasts the change of state region (i) whenever the node's CET crosses TH_1 if only two state regions are used, and (ii) whenever the node's CET crosses TH_{2k} if $K \geq 3$ state regions are used, where $1 \leq k \leq \lceil \frac{K}{2} \rceil - 1$. The reason for not broadcasting the change of state region whenever a node's CET crosses any threshold is to reduce the network traffic resulting from region–change broadcasts.

As shown in Table 3, P_{dyn} decreases as the number of state regions grows, and the decrease in P_{dyn} becomes insignificant when the number of regions grows beyond 4. Moreover, the frequency of region–change broadcasts remains essentially unchanged when the number of regions is below 4, but increases as the number of regions grows beyond 5. This is because a finer resolution of state intervals results in more reference points for region–change broadcasts. Thus, 3 or 4 state regions suffice to give a satisfactory performance.

One significant result is that the proposed scheme is shown to be robust to the variation of the tunable parameters, as compared to the other schemes reported in [14, 17, 22]. The change in P_{dyn} is shown to be less than 10^{-3} for any given change in either the threshold interval, or the number of state regions, or the values of thresholds. This is an important advantage coming from the use of prior/posterior distributions and Bayesian analysis. The proposed LS scheme can thus provide good performance even with not well–tuned parameters as long as the general rules discussed above are followed.

5.2 Evaluation of Performance Measures

5.2.1 Probability of Dynamic Failure

A task is said to be *missed* and a dynamic failure occurs if the sum of its queueing–for–execution time and the delay in transferring the task exceeds its laxity. Let $P_{dyn|d}$ denote the probability of missing deadlines for a task with laxity d. Then,

$$P_{dyn} = \sum_{j=0}^{T_{max}} P_{dyn|j} \cdot \hat{q}_j .$$

Figs. 2 and 3 are the plots of P_{dyn} vs. task arrival rate (λ), and $P_{dyn|d}$ vs. task laxity d, respectively. Table 4 shows some numerical results of $P_{dyn|d}$ under different schemes. As was expected, P_{dyn} increases as the system load gets heavy and/or the task laxity gets tight.

The random selection scheme outperforms the state probing scheme when the system load gets heavy (e.g., Table 4 (a) vs. (b)) or the task laxity gets tight (e.g., $L = \{1, 2, 3\}$ vs. $L = \{1\}$ in Table 4 (b)). This is because (1) under heavy loads, most nodes are likely to become unable of guaranteeing tasks, which will in turn make state probing unsuccessful most of the time, and (2) probing other nodes before sending an unguaranteed task requires two communication messages (one for request and the other for response), whereas the random selection does not require such messages. This negative effect becomes more pronounced as laxities get tighter.

The focused addressing scheme outperformed the state probing and random selection schemes, but was inferior to the proposed scheme, especially when the task laxity is tight. This is because:

- 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.
- Not many RFB messages are issued under light loads, thus making a node unable to keep its observation of other nodes up–to–date and increasing the chance of transferring a task to an incapable focused node. This is intolerable to tasks with tight laxities.
- Requests and replies for bids become excessive under heavy loads, thus increasing communication delays. The state information collected via periodic state exchanges or the bids sent from other nodes may thus become out–of–date.

In all cases simulated, the proposed LS scheme is shown to outperform all but the perfect information scheme in meeting task deadlines, showing its effectiveness achieved by using the judicious collection and use of state information. It does not perform as well as the perfect information scheme due to the additional processing overhead introduced by the probability update process and the communication delays incurred in task transfers and region–change broadcasts. See Table 5 for the case where all processing/communication overheads are set to zero while holding the task–transfer delay at 10% of task execution time. If the processing/communication overheads were ignored, all the realistic schemes would perform much better than they actually do, thus under–estimating P_{dyn} — which is undesirable for real-time applications.

5.2.2 Maximum System Utilization

The system utilization is defined as the ratio of the 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 simply becomes λ. Since P_{dyn} increases with system load (Fig. 2), there exists an upper bound for λ, termed as *maximum system utilization* λ_{max}, below which $P_{dyn} \leq \epsilon$ can be guaranteed for some prespecified $\epsilon > 0$. Fig. 4 shows plots of the maximum system utilization vs. ϵ for a 16–node regular system. One important result is that we do not have to sacrifice system utilization to lower P_{dyn}, which is in contrast to the common notion of trading system utilization for real–time performance. Moreover, the proposed scheme has the performance closest, among all LS schemes considered, to the perfect information scheme, and usually outperforms the other realistic schemes by almost an order of magnitude.

5.2.3 Task Transfer–Out Ratio

A task arriving at a node has to be transferred if and only if the CET of that node exceeds the laxity of the task. The task transfer ratio, r, is defined as the portion of arriving tasks (both external and transferred–in tasks) that must be transferred. r is a measure of the traffic overhead caused by task transfers. Table 6 shows the values of r for the various task attributes under different schemes.

Under light to medium loads (e.g., λ = 0.2—0.4 in Table 6), the task transfer ratios associated with different schemes are very close to one another. This is because most tasks can be guaranteed locally or with at most one task transfer, and thus, the location policies employed by different schemes have little influence on system performance. On the other hand, when the system load gets heavy, the way of choosing a node for task transfer/re-transfer results in notable differences in performance. The state probing scheme has the task transfer ratio closest to the perfect information scheme, since it first checks a node's LS capability before transferring a task to that node. The proposed scheme is inferior to the state probing scheme due to its use of imperfect observation to probabilistically decide the receiver node. However, it does not require any probing overhead at the time of making a location decision.

The focused addressing scheme performs slightly better than the proposed scheme under light load. However, when the system load gets heavy, the performance of the focused addressing scheme deteriorates quickly due to the increased probability of making incorrect LS decisions based on out–of–date[11] state information.

5.2.4 System Size

It is found from the simulations that the larger the system size, the better the performance of the LS schemes will result. (See Table 4 and Table 7 for numerical results.) This is because a larger system has a larger processing capacity to handle bursty task arrivals at one or more

[11] as a result of the increased communication delays caused by excessive bidding messages.

nodes in the system. Besides, task transfers, state probes, RFB[12], and/or region–change broadcasts do not significantly increase the queueing delay (as compared to a system with only a few nodes) due to the increased communication capacity and the decreased chance of transferring two or more tasks through the same link or to the same node.

Due to the way the buddy sets and the preferred lists constructed [14], the unguaranteed tasks within each buddy set will be evenly shared by all capable nodes in the entire system as the system size grows, rather than overloading a few capable nodes within the same buddy set. Moreover, the preferred list of a node is different from those of other nodes' in its buddy set. Consequently, the percentage of common nodes in the preferred lists of the nodes in a buddy set gets smaller as the system size grows, and thus, there is a greater chance that the unguaranteed tasks of a node may be transferred to some other nodes not in its buddy set, leading to more even distribution of the unguaranteed tasks.

5.2.5 Frequency of Region–Change Broadcasts vs. Frequency of State Probing/Bidding

In the proposed LS scheme, each node has to broadcast a change of state region to all the other nodes in its buddy set. Thus, the frequency of region–change broadcasts, f_b, determines the traffic overhead in collecting state information. On the other hand, the traffic overhead in both the state probing and focused addressing schemes is determined by the frequency of state probing, f_p, and the frequency of RFB, f_r, respectively. Table 8 summarizes the simulation results on f_b, f_p, and f_r in terms of number of messages per $E(R)$.

Under light to medium loads (e.g., λ = 0.2—0.6 in Table 8 (a)), the proposed scheme introduces more traffic overhead (in the worst case, about 0.5 broadcast per $E(R)$) than the other two schemes. However, the additional traffic introduced by broadcasts may not impede the transmission of tasks and/or other messages, since only about 2%—13% of the arriving tasks (Table 6) are transferred to other nodes. Besides, the effect of the increased communication delay (as a result of broadcasts) on the inconsistency between a node's observed and true states of other nodes is taken care of by Bayesian analysis.[13] When the system load is heavy, the state probing scheme and the focused addressing scheme perform worse than the proposed scheme (Table 8 (b)). This phenomenon becomes more pronounced when the

[12] To reduce traffic overheads, RFB messages are sent to 10 randomly selected nodes, instead of all the other nodes in the system.

[13] This is evidenced by the fact that the queueing delays due to state broadcasts/task transfers as well as the processing overheads required for probability update/state broadcasts were all included in our simulation.

variance of task execution time is large or when the task laxity is tight.

5.2.6 FCFS vs. MLFS

Under the MLFS local scheduling policy, the task queue at each node is ordered by task laxities, and a task with the minimum laxity on the node is always executed first. Another commonly–used local scheduling policy is the first–come–first–served (FCFS) discipline. A newly–arrived task is added at the end of task queue if it can be guaranteed on that node, and will otherwise be considered for transfer. The FCFS discipline is simple and ensures not to alter the existing guarantees, and the transfer policy under this discipline becomes a simple dynamic threshold type [5, 6, 17, 25].[14]

Although the MLFS discipline does not always perform better than the other on a per–task basis, it is shown in [26] to perform better *on the average* in reducing P_{dyn}. To quantitatively compare the performance of these two disciplines, we ran simulations with MLFS and FCFS as the local scheduling policy for a wide range of task attributes. As shown in Table 9, all schemes with FCFS perform worse than their counterparts with MLFS. No matter which local scheduling policy is used, the proposed scheme is shown to outperform the other realistic schemes in meeting deadlines. This demonstrates the quality of the location policy in the proposed scheme which uses both Bayesian analysis and the preferred lists.

One interesting result is that the performance degradation of the FCFS discipline is not so significant when (1) the system load is light, and thus, the task queue is often short (e.g., task set II in Table 9), or (2) task laxities are not distributed very widely (e.g., task set III in Table 9); that is, either all laxities are very large or all are very tight. This makes the quality of local scheduling less important in meeting task deadlines.

5.2.7 Sensitivity to Communication Delays

There are two types of communication delay to consider: one is the state–collection delay incurred from region–change broadcasts/state probes, where the queueing delays play a dominating role, and the other is the delay associated with task transfers, where both the queueing delays and the transmission delays dominate. To study the effect of communication delays, P_{dyn} was computed with (1) the transmission cost associated with each task transfer

[14] The threshold may change dynamically with the current state of the node and the time constraints of tasks.

being 5, 10, 15, and 20 % of the task execution time, and (2) the queueing delay coefficients being halved, doubled, and tripled. (Recall that the queueing delay coefficients are the coefficients in the linear expression used to model the effect of both the average external task arrival rate and the medium traffic on the queueing delay.)

As shown in Fig. 5, the state probing scheme, the random selection scheme, and the focused addressing scheme are all more sensitive to the variation of the transmission cost than the proposed scheme. Also, see Table 10 (a) for numerical examples. The performance degradation by the state probing scheme occurs because, as the task transmission delay increases, other tasks may arrive at a probed node during the period between the time it was probed and the time an unguaranteed task (of the probing node) arrives at that node. Thus, there is not much correlation between the state when a node was probed and the state when an unguaranteed task arrived at the node. (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 ratios (Table 6) and large transmission costs.

Fig. 6 (Table 10 (b)) shows the effect of varying queueing–related costs on the performance of several LS schemes. The state probing scheme is most sensitive to the variation of queueing delay, because, in addition to suffering the same effect as varying transmission delays, the state probing scheme generates two additional messages per probe, thus increasing the possibility of a task missing its deadline, especially when the queueing delay is large. Varying queueing–related costs has the same effect as varying transmission costs on the random selection scheme as well as on the focused addressing scheme.

In contrast, our proposed scheme is less sensitive to the communication delays (both queueing and transmission delays) because of the use of prior/posterior distributions to characterize the correlation between the observation and the corresponding true state.

5.2.8 Benefit of Using Bayesian Decision Analysis

To actually measure the benefit of using Bayesian decision analysis, we ran a set of experiments using a scheme which is identical in all aspects to the proposed scheme except that no Bayesian analysis is used to capture the inconsistency between the observed and true states. As the numerical results in Table 11 indicate, the proposed scheme with the Bayesian analysis outperforms the one without the Bayesian analysis in minimizing P_{dyn}, especially when

1. task execution times vary over a wide range (Table 11 (a)),
2. the distribution of task laxity gets tight (Table 11 (a)),
3. the state-collection/task-transfer delays get large (Table 11 (b)).

The possibility of 'improper' task transfers as a result of using outdated state information increases under condition 1 and 3, while under condition 2 tasks with tight laxities are less immune to improper task transfers. All these conditions can be handled by characterizing the inconsistency between the true state and the outdated observation with Bayesian analysis.

5.2.9 Statistical Fluctuations in Task Arrival

One issue in using a Bayesian decision model is to what extent the proposed scheme remains effective when the task arrival pattern randomly fluctuates. This effect is evaluated by simulating different task sets with hyperexponential interarrival times. This represents a system potentially with bursty task arrivals, and the degree of fluctuation over short periods is modeled well by varying the coefficient of variation (CV) of the hyperexponential task interarrival times. Specifically, let T_t be the task interarrival time. By Chebyshev's inequality,

$$P(\mid T_t - E(T_t) \mid \geq \ nE(T_t)) \leq \frac{CV^2}{n^2};$$

i.e., the smaller CV^2, the less likely T_t will deviate from its mean, $E(T_t)$. Fig. 7 shows the simulation results under heavy system loads ($\lambda = 0.8$) where the LS performance is sensitive to the variation of CV. From Fig. 7, we draw the following conclusions: (1) the two curves labeled as the proposed scheme and the proposed scheme without using Bayesian analysis give another evidence that LS does benefit from the use of Bayesian decision theory; (2) the performance of the proposed scheme degrades as CV increases. However, the proposed scheme remains effective up to CV=5.42 (or $CV^2 = 30$) beyond which it reduces essentially to the scheme without using Bayesian decision analysis.

6 Concluding Remarks

Using prior/posterior distributions and Bayesian analysis, we proposed a new LS scheme which can estimate, even with out–of–date state information, the workload of other nodes, and select the best candidate receiver of each unguaranteed task. The probability of dynamic failure as a result of using out–of–date information is thus reduced significantly. Moreover, as the simulation results indicate, the ability of making Bayesian decisions based on imperfect state information makes this scheme insensitive to communication delays. The proposed scheme is also shown to be robust to the variation of tunable parameters used in adaptive LS.

In a companion paper [27], using the continuous time Markov chain embedded in the corresponding Markov process, we developed an analytic model that describes the state evolution of a node with Poisson arrivals for several LS schemes. Specifically, the state of a node is defined as the CET on that node, and is modeled as an $M^{[k]}/D/1$ queue with bulk arrivals. The task arrival rate at a node is a function of the node's CET and the transfer/location policies used.

References

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

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

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

[4] T. P. Yum and H.-C. Lin, "Adaptive load balancing for parallel queues with traffic constraints," *IEEE Trans. on Communications*, vol. COM–32, no. 12, pp. 1339–1342, December 1984.

[5] Y. T. Wang and R. J. T. Morris, "Load sharing in distributed systems," *IEEE Trans. on Computers*, vol. C–34, no. 3, pp. 204–217, March 1985.

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

[7] T. C. K. Chou and J. A. Abraham, "Distributed control of computer systems," *IEEE Trans. on Computers*, vol. C–35, no. 6, , June 1986.

[8] C.-Y. H. Hsu and J. W.-S. Lin, "Dynamic load balancing algorithms in homogeneous distributed systems," *IEEE Proc. 6th International Conf. on Distributed Computing Systems*, pp. 216–223, 1986.

[9] J. F. Kurose and R. Chipalkatti, "Load sharing in soft real-time distributed computer systems," *IEEE Trans. on Computers*, vol. C–36, no. 8, pp. 993–999, August 1987.

[10] A. Weinrib and S. Shenker, "Greed is not enough: Adaptive load sharing in large heterogeneous systems," *IEEE INFOCOM'88–The Conference on Computer Communications Proceedings*, pp. 986–994, 1988.

[11] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Trans. on Computers*, vol. C–38, no. 8, pp. 1110–1141, August 1989.

[12] L. M. Ni and K. Hwang, "Optimal load balancing in a multiple processor system with many job classes," *IEEE Trans. on Software Engineering*, vol. SE–11, no. 5, pp. 491–496, 1985.

[13] J. A. Stankovic, "An application of Bayesian decision theory to decentralized control of job scheduling," *IEEE Trans. on Computers*, vol. C–34, no. 2, pp. 117–130, February 1985.

[14] K. G. Shin and Y.-C. Chang, "Load sharing in distributed real-time systems with state change broadcasts," *IEEE Trans. on Computers*, vol. C–38, no. 8, pp. 1124–1142, August 1989.

[15] J. A. Stankovic, K. Ramamritham, and S. Chang, "Evaluation of a flexible task scheduling algorithm for distributed hard real-systems," *IEEE Trans. on Computers*, vol. C–34, no. 12, pp. 1130–1141, December 1985.

[16] R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Analysis of the effect of delays on load sharing," *IEEE Trans. on Computers*, vol. C–38, no. 11, pp. 1513–1525, November 1989.

[17] R. Mirchandaney, D. Towsley, and J. A. Stankovic, "Adaptive load sharing in heterogeneous systems," *IEEE Proc. 9th Int'l Conf. on Distributed Computing Systems*, pp. 298–306, 1989.

[18] T. L. Casavant and J. G. Kuhl, "Analysis of three dynamic distributed load-balancing strategies with varying global information requirements," *IEEE Proc. 7th International Conf. on Distributed Computing Systems*, pp. 185–192, 1987.

[19] K. G. Shin and Y.-C. Chang, "A coordinated location policy for load sharing in hypercube multicomputers," submitted for publication, 1990.

[20] J. O. Berger, *Statistical Decision Theory and Bayesian Analysis*, Springer-Verlag, 1986.

[21] M. H. DeGroot, *Optimal Statistical Decisions*, Mcgraw-Hill, Inc., 1970.

[22] S. Pulidas, D. Towsley, and J. A. Stankovic, "Embedding gradient estimators in load balancing algorithms," *IEEE Proc. 8th Int'l Conf. on Distributed Computing Systems*, pp. 482–490, 1988.

[23] P. Ramanathan, D. D. Kandlur, and K. G. Shin, "Hardware assisted software clock synchronization for homogeneous distributed systems," *IEEE Trans. on Computers*, vol. C–39, no. 4, pp. 514–524, 1990.

[24] D.-T. Peng and K. G. Shin, "Static allocation of periodic tasks with precedence constraints in distributed real-time systems," *IEEE Proc. 9th International Conf. on Distributed Computing Systems*, pp. 190–198, 1989.

[25] R. Alonso and L. L. Cova, "Sharing jobs among independently owned processors," *IEEE Proc. 8th International Conf. on Distributed Computing Systems*, pp. 282–288, 1988.

[26] J. Hong, X. Tan, and D. Towsley, "A performance analysis of minimum laxity and earliest deadline scheduling in a real-time system," *IEEE Trans. on Computers*, vol. C-38, no. 12, pp. 1736–1744, December 1989.

[27] K. G. Shin and C.-J. Hou, "Analytic models of adaptive load sharing schemes in distributed real-time systems," to appear in *IEEE Trans. on Parallel and Distributed Systems*, 1993.