A Coordinated Location Policy for Load Sharing in Hypercube-Connected Multicomputers

Kang G. Shin, *Fellow, IEEE*, and Yi-Chieh Chang, *Member, IEEE*

***Abstract*—Uneven task arrivals in a hypercube-connected multicomputer may temporarily overload some nodes while leaving others underloaded. This problem can be solved or alleviated by load sharing (LS); that is, some of the tasks arriving at overloaded nodes, called *overflow* tasks, are transferred to underloaded nodes. One important issue in LS is to locate underloaded nodes to which the overflow tasks can be transferred. This is termed the *location policy*. Any efficient location policy should distribute the overflow tasks to the entire system instead of 'dumping' them on a few underloaded nodes. To reduce the overhead for collecting state information and transferring tasks, each node is required to maintain the state information of only those nodes in its proximity, called a *buddy set*. Several location policies—random probing, random selection, preferred lists, and bidding algorithm—are analyzed and compared for hypercube-connected multicomputer systems. Under the random-selection and preferred-list policies, an overloaded node can select, without probing other nodes, an underloaded node within its buddy set, while under the random probing policy and the bidding algorithm the overloaded node needs to probe other nodes before transferring the overflow task. *Task collision*(s) is said to occur if two or more overflow tasks are transferred (almost) simultaneously to the same underloaded node. The performances of these location policies are analyzed and compared in terms of the average number of task collisions. Our analysis shows that use of preferred lists allows the overflow tasks to be shared more evenly throughout the entire hypercube than the other two location policies.**

***Index Terms*—Load sharing, hypercube-connected multicomputers, location policy, buddy sets, preferred lists, random probing and selection, bidding algorithm.**

I. Introduction

The hypercube topology has drawn considerable attention as a multicomputer architecture due mainly to its regularity, potential for fault-tolerance, and scalability over a useful range [1], [2], [3]. Thus, a large number of autonomous computers or computing nodes can be connected in hypercube structure to exploit these features. Various research issues in hypercube multicomputers have been addressed, such as fault-tolerant routing and broadcasting [4], [5] and subcube allocation [6]. However, very little on workload distribution/redistribution to meet certain requirements in hypercube multicomputers has been reported in the open literature. The hypercube multicomputer considered in this paper is different from most commercial hypercubes in that it is a stand-alone machine, not a backend or attached machine. In other words, it is a general multicomputer in which nodes are connected with the binary hypercube topology. So, each node in this system is independent of others and tasks (user programs) arrive directly[1] at different nodes, while in most commercial hypercubes only a host computer/node receives external tasks and distributes them to processing nodes. In such a hypercube-connected multicomputer, bursty task arrivals may temporarily overload some nodes while leaving others underloaded. One way to alleviate this problem is load sharing (LS); those tasks arriving at overloaded nodes, called "overflow tasks," are transferred to underloaded nodes in order to meet certain requirements, e.g., task deadlines.

An important issue associated with LS is to locate underloaded nodes efficiently so that each overloaded node can offload its overflow tasks with minimal delay. A *location policy* is responsible for resolving this issue. Many location policies have been proposed and analyzed, but in general, the location policies that collect more state information are shown to outperform those without collecting any state information. There are two different approaches to collecting state information: 1) on a regular basis and 2) only when a node needs to transfer some of its tasks, or when a node is ready to receive tasks. In this paper, we will call the first approach *state-collection* (SC) [7] and the second approach *state probing* (SP) [8], [9], [10], [11]. In the SC method, each node collects and maintains the state information of other nodes, regardless whether the information will be used or not, so that an overloaded node can transfer its tasks without incurring any probing delay. Hence, a fixed amount of state-collection overhead will be induced to the system even when no task needs to be transferred. In the SP method, on the other hand, the overloaded node needs to probe other nodes to find a potential receiver before transferring its tasks, thus incurring delays to the tasks to be transferred. But no state-collection overhead will incur when there are no tasks to be transferred.

The SP location policies have been studied extensively. Two commonly-used methods are probing [8] and *bidding* [9], [10], [12], [13]. Under the probing method [8], an overloaded node selects a node (candidate receiver) and checks whether or not the node can share its load. If it can, the overloaded node will transfer an overflow task to that node; otherwise, it will probe another node. The communication overhead for probing nodes was shown to depend only on the number of probes re-

Manuscript received Dec. 14, 1992; revised Sept. 1, 1993.
K.G. Shin is with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109-2122; e-mail: kgshin@eecs.umich.edu.
Y.-C. Chang is with the Electrical and Computer Engineering Department, University of Texas, El Paso.
IEEECS Log Number C95033.

1. Instead of going through a host node/computer.

gardless of system size. However, probing nodes is inefficient, because not only it will introduce an additional delay in completing the tasks to be transferred, but also the overloaded node may not be able to locate a receiver within a prespecified number of probes if there are only a few underloaded nodes in the system. In case a bidding algorithm [9], [10], [12], [13] is used, an overloaded node broadcasts a request for bids. Three to four rounds of messages have to be generated and exchanged before an overflow task is actually transferred. If an underloaded node's bid is accepted by more than one overloaded node and if the underloaded node can accept only one overflow task, then the rest of the overloaded nodes must start another round of bidding. This bidding process usually causes excessive communication delays to the tasks to be transferred. To reduce the number of messages in the bidding algorithm, Ni et al. proposed a *drafting algorithm* by allowing underloaded nodes to initiate the drafting process [11]. This algorithm is shown to reduce the number of messages exchanged to about one third of that of a bidding algorithm. However, when an underloaded node is chosen by more than one overloaded node, it may not be able to accept tasks from all overloaded nodes and complete them in time.

In the SC location policy, each node needs to collect and maintain the state information of other nodes, so that when a node becomes overloaded, it can transfer its overflow tasks without probing other nodes, thus eliminating the probing delay in the execution of overflow tasks. Although the SC location policy can minimize the task-transfer delay, it will incur a fixed amount of state-collection overhead. Thus, how to reduce/minimize the state-collection overhead in the SC location policy is an important design issue. We considered this problem first in [7]. The state-collection overhead was reduced by requiring each node to collect and maintain the state information of only a small set of nodes in its physical proximity, called a *buddy set*. This overhead was reduced further by using thresholds to determine the load state. For example, three thresholds, denoted as TH_u, TH_f, and TH_v, were used to define the load state (queue length, QL) of a node. A node is said to be *underloaded* (U–state) if $QL \leq TH_u$, *medium–loaded* (M–state) if $TH_u < QL \leq TH_f$, *fully loaded* (F–state) if $TH_f < QL \leq TH_v$, and *overloaded* (V–state) if $QL > TH_v$. Whenever a node becomes fully loaded (underloaded) due to the arrival and/or transfer (completion) of tasks, it will multicast its change of state to all the other nodes in its buddy set. Every node that receives this information will update its state information by eliminating the fully–loaded node from, or adding the underloaded node to, its ordered list (called a *preferred list*) of available receivers. An overloaded node can then select, without probing other nodes, the first available node from its preferred list.

The main goal of this paper is to design, and analyze the performance of, the SC and SP location policies with respect to their ability of solving the *coordination* and *congestion* problems. The coordination problem arises when more than one overloaded nodes attempt to (almost) simultaneously transfer overflow tasks to one underloaded node, and the congestion problem occurs when many overloaded nodes are located in a physical proximity. Specifically, we will develop a systematic method of constructing preferred lists for the SC method and prove that this method minimizes the probability of more than one overloaded node transferring overflow tasks to an underloaded node and distributes the overflow tasks in a buddy set over many different buddy sets, rather than overloading the nodes in its own buddy set. That is, this method solves both the coordination and congestion problems. Since we have already shown in [7] that the use of preferred lists works well in handling task deadlines, we would like to show that the new method of constructing the preferred lists can also be used for improving average system response time. Based on modeling and simulations, the SC location policy with the preferred lists is found to outperform the SP location policies, such as bidding and drafting algorithms, when the system is medium–loaded. But the SP location policies perform slightly better than the SC location policy when the system is lightly loaded. The difference between SC and SP location policies is negligible when the system is heavily loaded.

The rest of this paper is organized as follows. In Section II, we propose to solve the coordination and congestion problems by systematically constructing the preferred lists. Using the number of task collisions as a yardstick, the performance of the SC method with the proposed solutions to the coordination and congestion problems is evaluated in Secton III. In Section IV, the proposed SC location policy is compared with random selection, probing, bidding, and drafting policies. The paper concludes with Section V.

II. Construction of Preferred Lists and Buddy Sets

A. Advantages and Problems of the SC Method

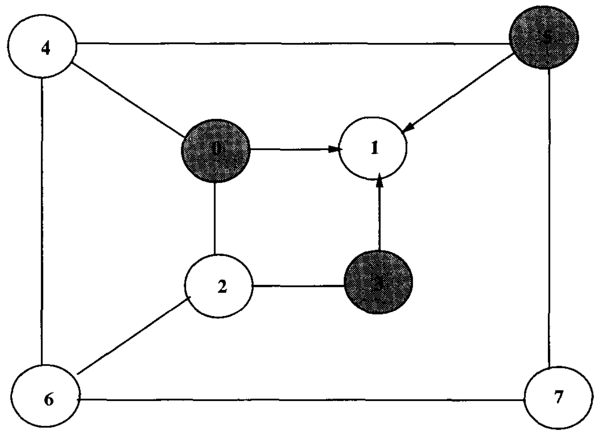
The fundamental difference between the SC and SP location policies lies in that the state information is collected on a regular basis in the SC method while in the SP method it is collected only when a node needs to transfer some of its tasks or when a node is ready to receive tasks. So, the advantage of the SC method is to reduce the task-transfer delay by making the state information available on a regular basis with a fixed state-collection overhead. On the other hand, the advantage of the SP method is to eliminate the state-collection overhead when no task needs to be transferred, but adds a probing delay to the execution of the tasks to be transferred. Since most of the SP location policies are well-known and studied extensively elsewhere, we will focus on the design and analysis of a SC location policy.

Since a fixed amount of state-collection overhead is imposed by the SC method, it is essential to design an SC location policy which minimizes this overhead. It is also necessary to coordinate task transfers to avoid the effect of 'dumping' tasks on the same underloaded node, as stated in the Introduction. Thus, we must design an SC location policy to 1) minimize the overhead for collecting/maintaining state information and 2) resolve the coordination and congestion problems. Our solution is to design an SC location policy with the features of state-change broadcasting, buddy sets, and preferred lists. Note that the details on how to set thresholds for state-change broadcasting and how to design buddy sets were treated in [7].

Any good location policy must be the result of resolving the following two issues. First, when the number of overloaded nodes is less than, or equal to, the number of underloaded nodes, no more than one overloaded node should be allowed to select the same underloaded node as a receiver. Simultaneous selection of an underloaded node by multiple overloaded nodes results from lack of coordination among them (hence the *coordination* problem [14]). Specifically, *n task collisions* are said to occur if $n + 1$ overflow tasks are sent (almost) simultaneously to an underloaded node.[2] Second, to minimize the task-transfer delay, the buddy set of a node is composed of those nodes in its physical proximity (e.g., those nodes one or two hops away). The overloaded nodes in a buddy set should be able to transfer their overflow tasks to the nodes in different buddy sets such that those tasks arriving at overloaded nodes within a *hot region*—a region where the number of overloaded nodes is greater than that of underloaded nodes—can be shared by the entire system, not just by those nodes in the same buddy set. (Formation of hot region(s) with an excessive number of overflow tasks causes the *congestion* problem.)

Despite their importance, neither the coordination problem nor the congestion problem was addressed in [7]. These problems can, of course, be resolved easily if every overloaded node knows the receivers of other overloaded nodes. It is, however, practically impossible to equip every overloaded node with this knowledge, because each overloaded node is required to communicate with all the other overloaded nodes before transferring an overflow task to an underloaded node. This will not only introduce excessive communication traffic, but also a significant delay in completing the tasks to be transferred, thus offsetting any benefit to be gained from LS. One must therefore establish a rule for each overloaded node to select a receiver among possibly multiple underloaded nodes while minimizing the probability of more than one overloaded node simultaneously transferring overflow tasks to the same underloaded node. For example, consider Fig. 1, where nodes 0, 3, and 5 are overloaded and all other nodes are underloaded. Unless properly coordinated, these three overloaded nodes may choose node 1 as the receiver, thus overloading node 1 and leaving the other nodes underloaded. Similarly, the congestion problem is illustrated in Fig. 2. Suppose nodes 0, 1, 2, 4, and 8 are in the same buddy set and three of them are overloaded (shaded circles), then they should transfer overflow tasks to the nodes outside of the buddy set. This problem can be resolved if the overloaded nodes select receivers using their preferred lists. As shown in Fig. 3, the overflow tasks within a hot region can be shared by the underloaded nodes outside of the region, thus spreading the overflow tasks over the entire system.

The congestion problem can be solved by designating each node in an n-cube as the kth preferred node of one and only one other node. In such a case, since each node is designated as the most preferred node of one and only one other node, the probability of an underloaded node being selected for task transfer by more than one overloaded node becomes very small. Moreover, to enable the entire system to share overflow tasks, the most preferred receiver of each node in a buddy set must also belong to a different buddy set. Since an overloaded node is most likely to transfer a task to its most preferred receiver, the overloaded nodes in a buddy set will spread their overflow tasks over many different buddy sets, thus solving the congestion problem. Note that the preferred list used in [7] is only one of a class of equivalent ways to construct the preferred lists. In this paper, we will treat this problem formally and derive the best way for constructing the preferred lists to resolve the coordination and congestion problems. For notational convenience, we will henceforth call this method the *preferred-list SC* (PLSC) location policy.



(a). Two collisions

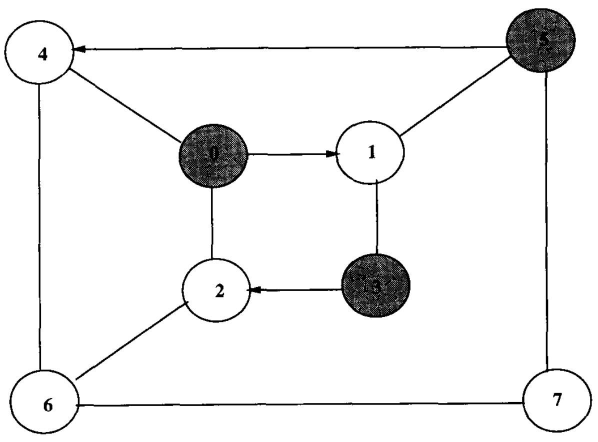


Fig. 1. Coordination problem.

2. This is to reflect the severity of the coordination problem. For example, sending three tasks to a node would be severer than sending two tasks to the node.

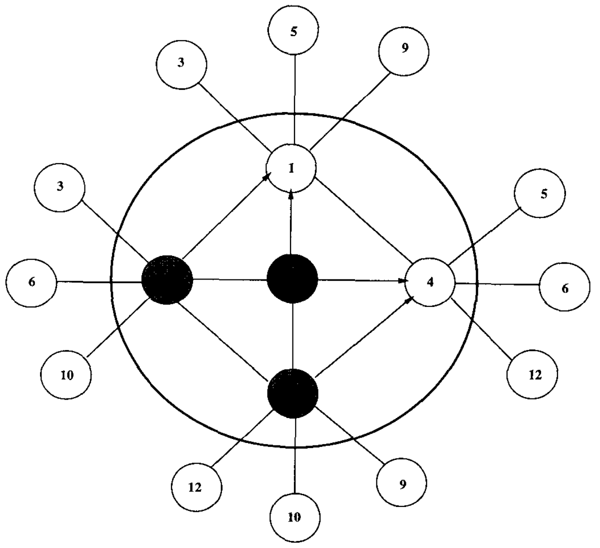


Fig. 2. Congestion problem.

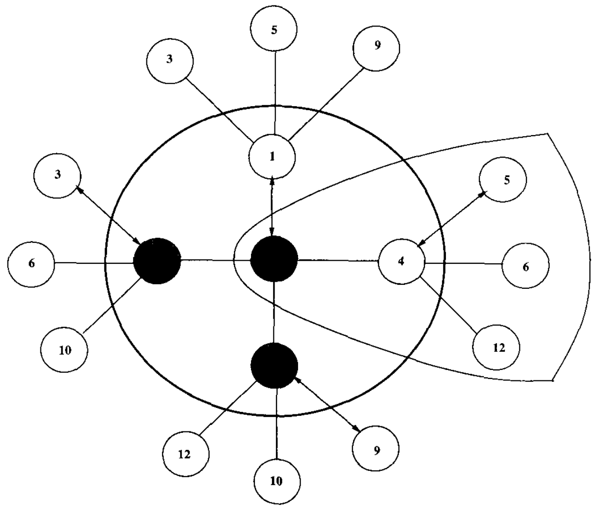


Fig. 3. Solution to the congestion problem.

B. Construction of a Preferred List

DEFINITION 1. *An n-dimensional binary hypercube, Q_n, is defined recursively as follows:*

1) *Q_0 is a trivial graph with one node, and*
2) *$Q_n = K_2 \times Q_{n-1}$, where K_2 is a complete graph with two nodes and $\times$ is the product operation on two graphs* [15].

The address of a Q_2 can be represented by a sequence of binary digits and two $*$s, where $*$ represents either 0 or 1. For example, the address $b_{n-1} \ldots b_2{**}$ represents a Q_2 formed by the four nodes, $b_{n-1} \ldots b_2 00$, $b_{n-1} \ldots b_2 01$, $b_{n-1} \ldots b_2 10$, $b_{n-1} \ldots b_2 11$. Let the address of node i (denoted by N_i) be $i_{n-1}i_{n-2} \cdots i_0$, I_k be the unit vector in which all but kth bit (which is set to 1) are 0, and $\oplus$ denote the bitwise EXCLUSIVE–OR operation. For example, every node, N_i, of a Q_n has n nodes within one hop, and the addresses of these n nodes can be obtained from $N_i \oplus I_k$ for $k = 0, \ldots, n-1$.

Since the cost of transferring a task increases with the distance between the sender and receiver, it is natural for the sender N_x to explore the nodes within one hop first, then the nodes within two hops, and so on. For convenience, the nodes within one hop of N_x are said to be in the *first component group* of N_x's preferred list, the nodes within two hops are in the *second component group*, and in general, the nodes within m hops are in the mth *component group* of N_x's preferred list.

The nodes in all component groups of N_i's preferred list are ordered as defined below.

DEFINITION 2. *Let $i_{n-1}i_{n-2} \cdots i_0$ be the address of N_i, then*

1) The nodes in the first component group are ordered as $\{(i_n\text{-}1\text{in}_{-2} \cdots i_0) \oplus I_j\}_{j=0}^{j=n-1}$.
2) *The nodes in the second component group are ordered as $\{(i_{n-1}i_{n-2} \cdots i_0) \oplus I_j \oplus I_k\}$ ($j = 1, \ldots, n-2, 0$ and $j+1 \le k \le n-1$).*
3) *The nodes in the third component group are ordered as $\{(i_{n-1}i_{n-2} \cdots i_0) \oplus I_j \oplus I_k \oplus I_{\,}\}$ ($j = 1, \ldots, n-3, 0, j+1 \le k \le n-2$, and $k+1 \le \,, \le n-1$).*
4) *In general, the nodes in the kth ($k \le n$) component group are ordered as $\{(i_{n-1}i_{n-2} \cdots i_0) \oplus I_{j_1} \oplus I_{j_2} \cdots \oplus I_{j_k}\}$ ($j_1 = 1, \ldots, n-k, 0, j_1+1 \le j_2 \le n-k+1, \cdots,$ and $j_{k-1}+1 \le j_k \le n-1$).*

Order of preference	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
N_0	1	2	4	8	6	10	12	3	5	9	14	13	11	7	15
N_1	0	3	5	9	7	11	13	2	4	8	15	12	10	6	14
N_2	3	0	6	10	4	8	14	1	7	11	12	15	9	5	13
N_3	2	1	7	11	5	9	15	0	6	10	13	14	8	4	12
N_4	5	6	0	12	2	14	8	7	1	13	10	9	15	3	11
N_5	4	7	1	13	3	15	9	6	0	12	11	8	14	2	10
N_6	7	4	2	14	0	12	10	5	3	15	8	11	13	1	9
N_7	6	5	3	15	1	13	11	4	2	14	9	10	12	0	8
N_8	9	10	12	0	14	2	4	11	13	1	6	5	3	15	7
N_9	8	11	13	1	15	3	5	10	12	0	7	4	2	14	6
N_{10}	11	8	14	2	12	0	6	9	15	3	4	7	1	13	5
N_{11}	10	9	15	3	13	1	7	8	14	2	5	6	0	12	4
N_{12}	13	14	8	4	10	6	0	15	9	5	2	1	7	11	3
N_{13}	12	15	9	5	11	7	1	14	8	4	3	0	6	10	2
N_{14}	15	12	10	6	8	4	2	13	11	7	0	3	5	9	1
N_{15}	14	13	11	7	9	5	3	12	10	6	1	2	4	8	0

Fig. 4. Preferred lists in a 4-cube system.

As an example, the preferred lists of all nodes in a four-cube are presented in Fig. 4. Once each node's preferred list is constructed, its buddy set can be formed by *any* required number of nodes counting from the top of its preferred list. (Note that the issue of determining the size of a buddy set has already been addressed in [7].) An overloaded node can then select an underloaded node from its buddy set based on the order of preference determined above. This selection can easily be implemented with a pointer to the first underloaded node in the node's preferred list. If, albeit rare, an overloaded node cannot find any underloaded node from its buddy set, all of its tasks will be executed locally.

In what follows, we show that the above preferred lists can solve both the coordination and congestion problems.

THEOREM 1. *Each node in an n-cube will be selected as the kth preferred node by one and only one other node for $k = 1, \ldots, N$ ($N = 2^n$).* u

The proof of this theorem follows from Definition 2. Since the nodes in the first component group of each node are determined by $\{(i_{n-1}i_{n-2} \cdots i_0) \oplus I_j\}_{j=0}^{j=n-1}$, the result of EXCLUSIVE–ORing a node's address with I_j is unique for $j = 0, \ldots, n-1$. Moreover, the ordering of the nodes in the second and higher component groups is essentially derived from that of the first component group nodes. So, a node will be selected as the kth preferred node by one and only one other node for $k = 1, \ldots, N$.

Since each node in a Q_n will be selected as the most preferred node by one and only one other node, the probability of an underloaded node being selected by more than one overloaded node is very small, thereby solving the coordination problem.

Before presenting Theorem 2, it is necessary to consider one special feature of hypercube structure, i.e., there does not exist any cycle which is composed of an odd number of nodes. For example, let the sequence of nodes $N_0, N_1, \ldots, N_k$ be part of a cycle, where N_i and N_{i+1} are adjacent to each other for $i = 0, \ldots, k-1$. Using the $\oplus$ operation for $0 \le i \le k-1$, this sequence of nodes can be represented as follows: $N_{i+1} = N_i \oplus I_p$, and $N_{i+2} = N_{i+1} \oplus I_q = N_i \oplus I_p \oplus I_q$ for $0 \le p, q \le n-1$. Then it is easy to show that N_{i+1} is one hop away from N_0, and N_{i+2} is either two hops away from N_0 if $p \ne q$, or is equal to N_0 if $p = q$. Repeating this procedure, $N_{i+3} = N_0 \oplus I_p \oplus I_q \oplus I_r$ is either one hop away from N_0 if any two of p, q, r are equal, or three hops away from N_0 if p, q, r are all distinct integers. Generally, one can show that N_{i+j} will be an odd (even) number of hops away from N_0 if j is odd (even). In order to form a cycle, the start and end nodes must be one hop away from each other, so j can only be an odd number, thus making the total number of nodes in a cycle even.

THEOREM 2. *The most preferred node of each node in a buddy set must come from a different buddy set if the buddy set size is no greater than $\binom{n}{2}$, where n is the dimension of the hypercube.*

PROOF. Since the numbers of nodes in the first and second component groups of a node's preferred list are $\binom{n}{1}$ and $\binom{n}{2}$, respectively, the buddy set of size $\le \binom{n}{2}$ will consist of only the nodes in the first or second component group. This theorem is proved in the following two steps.

Step 1. Any two nodes in the first or second component group cannot be adjacent to each other, otherwise one can form a cycle with an odd number of nodes, thus violating the feature of hypercube structure mentioned above. Since the most preferred node of a node is within one hop, a node in the first or second component group cannot select another node in the same component group as its most preferred node.

Step 2. Each node in the first component group will be adjacent to some other nodes in the second component group, but these adjacent nodes will not select each other as their most preferred nodes if the buddy set size is not greater than $\binom{n}{2}$ for the following reason. As stated earlier, the most preferred node of a first component group node will be in the first component group of the most preferred node. According to Definition 2, the first component group of the most preferred node is the last component group to be considered in generating the second component group for each node. Since there are $\binom{n}{1}$ nodes in the first component group, if the buddy set size is not greater than $\binom{n}{2}$, then only the first $\binom{n}{2} - \binom{n}{1}$ nodes in the second component group will be included in the buddy set. Thus, none of the first component group nodes in a given node's preferred list will select any node in the second component group which is included in its buddy set as the most preferred node. u

If the system load is well-balanced, the probability of a node staying in V-state is much smaller than in U-state because of the assumed system stability. So, an overloaded node is most likely to transfer a task to its most preferred node; the overloaded nodes in a buddy set will spread their overflow tasks over many different buddy sets instead of overloading the nodes in the buddy set, thus solving the congestion problem. As concluded in [7], buddy sets with 10 to 15 nodes perform well in most cases, so the buddy set size of $\binom{n}{2}$ in Theorem 2 will not alter the usefulness of preferred lists in practice for any Q_n, $n > 5$.

From Definition 2, the first component group nodes are ordered as $\{(i_{n-1}i_{n-2} \cdots i_0) \oplus I_j\}_{j=0}^{j=n-1}$. One can generate another set of preferred lists by changing the incrementing sequence of index j. (There are $n!$ ways of ordering the nodes in the first component group.) However, the resulting preferred lists will also retain the properties of Theorems 1 and 2, making them indistinguishable from the ones already generated according to Definition 2.

III. Derivation of Number of Task Collisions in PLSC

We now want to analyze and derive the performance of the PLSC location policy in terms of task collisions. A good location policy should be able to transfer overflow tasks from overloaded nodes to underloaded nodes in such a way that each underloaded node may not exceed its capacity (e.g., in meeting deadlines) due to transferred tasks; otherwise, some of the underloaded nodes may become overloaded and need to retransfer some of their received tasks, thus possibly increasing the network traffic and the task-transfer delay without improving system performance.

So, we will develop a model using the number of task collisions to analyze the goodness of a location policy. Specifically, if $n + 1$ overflow tasks are sent simultaneously to an underloaded node, n task collisions will result. This quantification of task collision is to reflect the severity of the coordination problem. We want to devise a location policy so that each overloaded node may find an underloaded receiver without any conflict with other overloaded nodes. Although acceptance of more than one overflow task at a time may not necessarily make an underloaded node overloaded, the average response time in such a case will be larger than the case in which each overloaded node can find an underloaded node that is not chosen by any other overloaded node. For example, if an underloaded node can complete only one overflow task in time, then all but one transferred task will miss their deadlines or must be retransferred. In such a case, it is the best location policy that results in no task collision at all.

The performance of a location policy strongly depends on the number of task collisions, especially if every underloaded node is required not to overload itself at any time by receiving transferred tasks from other nodes. To meet this requirement, one has to use a location policy that results in no task collision.

Let the *capacity* of an underloaded node be defined as the maximum number of tasks that a node can receive without overloading itself. If the number of underloaded nodes is always greater than, or equal to, the number of overloaded nodes, then the total number of overflow tasks is less than the combined capacities of all underloaded nodes. One can show that the necessary and sufficient condition to guarantee each underloaded node not to exceed its capacity by receiving transferred tasks is to avoid task collisions. Since each underloaded node can accept at least one task, it will not exceed its capacity when there is no task collision (each underloaded node will receive at most one task at a time). As soon as an underloaded node receives a transferred task, it will update and multicast its load state to the nodes in its buddy set. If there are task collisions, some of the underloaded nodes may receive more transferred tasks than their capacity, thus needing to retransfer some of the received tasks. So, to guarantee each underloaded node not to exceed its capacity is to avoid task collisions.

In the rest of this section, we will derive the number of task collisions for the proposed location policy as a function of the total number of nodes and the number of overloaded nodes.

To simplify the analysis, we assume that only U– or V–state nodes exist in the system. We will show later that this analysis can be easily extended to the case with F–state nodes. The conditions that result in no task collision are stated in the following two lemmas.

LEMMA 1. *If at most two nodes in a Q_2 are overloaded, there will be no task collision in the Q_2.*

PROOF. If there is only one overloaded node in a Q_2, this node will transfer an overflow task to its most preferred node which is underloaded. If there are two overloaded nodes in the Q_2, these two nodes will both select their most or second preferred nodes to be the receivers. In either case, no task collision will occur according to Theorem 1. u

Note that without using a preferred list, there is a 50% chance that two overloaded nodes will simultaneously transfer tasks to the same underloaded node in a Q_2, thus resulting in one task collision. Let $N = 2^n$ and k be the number of overloaded nodes in a Q_2. From Lemma 1, the number of ways these k overloaded nodes can be distributed in the system without resulting in any task collision is:

$$\begin{aligned}&\sum_{i=0}^{\lceil k/2\rceil}\frac{(k-i)!}{i!(k-2i)!}\binom{N/4}{k-i}\times\binom{4}{1}^{k-2i}\times\binom{4}{2}^{i}\\&=\sum_{i=0}^{\lceil k/2\rceil}\binom{N/4}{N/4-k+i}\binom{k-i}{i}\binom{4}{1}^{k-2i}\binom{4}{2}^{i}\\&=\sum_{i=0}^{\lceil k/2\rceil}\binom{N/4}{N/4-k+i,\ i,\ k-2i}\binom{4}{1}^{k-2i}\binom{4}{2}^{i}\end{aligned}\tag{1}$$

where $\binom{a}{b}=\frac{a!}{(a-b)!b!}$ and $\binom{a}{b,\ c,\ d}=\frac{a!}{b!c!d!}$.

For convenience, let $\Lambda(N, k)$ =

$$\Lambda(N,k)=\sum_{i=0}^{\lceil k/2\rceil}\binom{N/4}{N/4-k+i,\ i,\ k-2i}\binom{4}{1}^{k-2i}\binom{4}{2}^{i}.$$

Equation (1) can be explained as follows. Let x and y be the numbers of Q_2s in an n–cube that have one or two overloaded nodes, respectively. Then, $x + y \le 2^{n-2}$ ($= N/4$) and $2x + y = k$ when $k \le 2^{n-1}$ for $x = 0, \ldots, \lceil k/2\rceil$, $y = k - 2x$. The number of ways of distributing k overloaded nodes in the system without causing any task collision is the same as that of choosing $x + y$ Q_2s out of the total number of Q_2s ($N/4$) in a Q_n, thus yielding the $\binom{N/4}{k-i}$ term in (1). Since there are many possible combinations of x and y, we need to sum them. Furthermore, the number of ways of assigning the overloaded nodes to the selected Q_2s is obtained from the binomial formula, thus giving the $\binom{4}{1}^{k-2i}\binom{4}{2}^{i}$ term in (1).

LEMMA 2. *If all four nodes in a Q_2 are overloaded but none of the nodes in its most preferred Q_2 is overloaded, no task collision will occur to this Q_2.*

PROOF. Even if all four nodes are overloaded in a Q_2, each of them will find a receiver from its most preferred Q_2. Since none of the nodes in the latter Q_2 is overloaded, no task collision will occur. u

From Lemma 2, the number of ways k overloaded nodes can be distributed over N nodes without any task collision is

$$\begin{aligned}&\sum_{\ell=1}^{\lfloor k/4\rfloor}\binom{N/4}{\ell}\times\left[\sum_{i=1}^{\lceil k'/2\rceil}\binom{N'/4}{k'-i}\binom{k'-i}{i}\binom{4}{1}^{k'-2i}\binom{4}{2}^{i}\right]\\&=\sum_{\ell=1}^{\lfloor k/4\rfloor}\binom{N/4}{\ell}\times\left[\sum_{i=0}^{\lceil k'/2\rceil}\binom{N'/4}{N/4-k'+i,i,k'-2i}\binom{4}{1}^{k'-2i}\binom{4}{2}^{i}\right]\end{aligned}\tag{2}$$

where ℓ is the number of groups each with four overloaded nodes, $N' = N - 8\ell$, and $k' = k - 4\ell$. The first term $\binom{N/4}{\ell}$ in (2) is the number of ways of selecting overloaded Q_2s out of the total $N/4$ Q_2s. The rest of overloaded nodes, $k' = k - 4\ell$, can be distributed among the rest of nodes, $N' = N - 8\ell$, as shown in (1).

Combining (1) and (2), the probability, P_0, of no task collision given N and k, can be calculated by:

$$P_0 = \frac{\Lambda(N,k) + \sum_{\ell=1}^{\lfloor k/4 \rfloor} \binom{N/4}{\ell} \Lambda(N',k')}{\binom{N}{k}}, \qquad (3)$$

where $N' = N - 8\ell$, $k' = k - 4\ell$.

THEOREM 3. *At least three nodes must be overloaded to result in task collisions, and these three nodes must include a node, and both its first and second preferred nodes; otherwise, no task collision will occur.*

PROOF. If the number of overloaded nodes is less than three, or if there are three overloaded nodes distributed over at least two different Q_2s, then no collision will occur according to Lemma 1. If all three overloaded nodes reside in the same Q_2, one can show that these three nodes must include a node and both its first and second preferred nodes, and in such a case, a task collision will always occur.

Since there is no duplication of nodes in a preferred list, a node and its first and second preferred nodes must all be different. Since there are only four different ways to choose three out of the four nodes in a Q_2, each of these choices must include a node and its first and second preferred nodes.

The last part of this proof is to show that if a node and its most and second preferred nodes are overloaded, a task collision will always occur. According to Definition 2, the most and second preferred nodes of N_i are $i_{n-1}i_{n-2} \ldots i_0 \oplus I_0$ and $i_{n-1}\, i_{n-2} \ldots i_0 \oplus I_1$. Since all these nodes are overloaded, each of them will try to transfer a task to an underloaded node in its preferred list. It is easy to show that N_i will transfer an overflow task to node $i_{n-1}i_{n-2} \ldots i_0 \oplus I_2$, while node $i_{n-1}i_{n-2} \ldots i_0 \oplus I_0$ (N_i's most preferred node) will transfer an overflow task to node $i_{n-1}i_{n-2} \ldots i_0 \oplus I_0 \oplus I_1$, and node $i_{n-1}i_{n-2} \ldots i_0 \oplus I_1$ (N_i's second preferred node) will transfer an overflow task to node $i_{n-1}i_{n-2} \ldots i_0 \oplus I_1 \oplus I_0$. Since $I_0 \oplus I_1 = I_1 \oplus I_0$, the most and second preferred nodes of N_i, will simultaneously transfer tasks to the same node $i_{n-1}i_{n-2} \ldots i_0 \oplus I_0 \oplus I_1$, thus resulting in a task collision. u

COROLLARY 1. *If the number of overloaded nodes is greater than three and includes the pattern described in Theorem 3, task collisions will always occur; otherwise, no collision will occur.*

Task collision will occur with the following patterns of overloaded nodes, or combinations thereof:

Pattern 1: three overloaded nodes in a Q_2
==> one task collision.

Pattern 2: four overloaded nodes in a Q_2 whose most preferred Q_2 has one overloaded node
==> one task collision.

Pattern 3: four overloaded nodes in a Q_2 whose most preferred Q_2 has two overloaded nodes
==> two task collisions.

Given N and k, the probability of resulting in a task collision is given by

$$P_i = \frac{\binom{N/4}{1}\binom{4}{3}\Lambda(N',k') + \binom{N/4}{2}\binom{4}{1}\Lambda(N'',k'')}{\binom{N}{k}} \qquad (4)$$

where $N' = N - 4$, $N'' = N - 8$, $k' = k - 3$, and $k'' = k - 5$. The first (second) term in (4) is the number of ways to select one Q_2 out of the total $N/4$ Q_2s and distribute three (four) overloaded nodes in this Q_2 times the number of ways to distribute the rest of overloaded nodes without any task collision. Since three, five, and six overloaded nodes are required in patterns 1 to 3, respectively, the assignment to each of these patterns will consume a corresponding number of overloaded nodes (three, five, or six) out of the total number of overloaded nodes in the above equation.

When two or more task collisions occur, all combinations of these patterns need to be considered. For example, ℓ collisions can result from any combination of a groups of pattern 1, b groups of pattern 2, and c groups of pattern 3, such that $a + b + 2c = \ell$, and $a, b, c \in \{0, 1, \ldots, \ell\}$. Given N and the number, ℓ, of collisions, the following algorithm determines the total number of combinations of these patterns.

```
For $t_{rej}$ = 1 to $\ell$ do
   for $i$ = 0 to $\lfloor t_{rej}/2 \rfloor$ do
      for $j$ = $t_{rej}$ - 2$i$ to 0 do
         $Pattern_1$ <— $j$;
         $Pattern_2$ <— $t_{rej}$ - 2$i$ - $j$;
         $Pattern_3$ <— $i$;
         $N_{over}$ <— $Pattern_1$ × 3 + $Pattern_2$ × 5 +
         $Pattern_3$ × 6;
         $j$ <— $j$ - 1;
      end_do
      $i$ <— $i$ + 1
   end_do
end_do
```

Although many combinations of the above three patterns can result in the same number of task collisions, they may require a different number of overloaded nodes which was given as N_{over} in the above algorithm. Once combinations of these patterns are determined, the probability of resulting in ℓ task collisions, denoted by P_ℓ, can be calculated by:

$$P_\ell = \frac{\sum_{\text{all } a,b,c} \binom{N/4}{a}\binom{N/4-a}{b}\binom{N/4-a-b}{c}\binom{a+b+c}{a,b,c}\Lambda(N',k')}{\binom{N}{k}} \qquad (5)$$

where $\ell = 1, \ldots, t_{rej}$, $N' = N - 4\ell$, and $k' = k - N_{over}$.

It should be noted that in the above analysis only U– and V– state nodes are considered. Equations (4) to (5) can be easily modified to derive task collisions in the presence of F– state nodes. Let f and k be the number of nodes in F– and V–

state, respectively. One can add f to k as the total number of V–state nodes and apply (5). The actual number of task collisions will be $[k/(f+k)]^2$ times Equation (5), because a task collision will occur only when two V–state nodes simultaneously send tasks to the same U–state node.

IV. Comparative Performance Evaluation

The performance of the PLSC location policy is compared against the SP location policies, such as the probing, bidding, and drafting algorithms. In order to show the advantages of PLSC location policy, a similar approach without using the preferred list, called *random selection*, is also considered. The random selection[3] policy is exactly the same as the proposed location policy except that it does not use preferred lists. Since each node collects, via state–change multicasts, the state information of other nodes in its buddy set, a V–state node can randomly select one of the U–state nodes and transfer an overflow task to that node. Note that this is a typical location policy when a node's LS is restricted to its neighboring nodes. Moreover, the performance achieved from this method can be used for comparison with the approach of using the preferred lists. Under the probing policy, on the other hand, a V–state node will randomly probe the nodes in its buddy set to find a receiver. The V–state node will transfer an overflow task to the node it probed if it happens to be underloaded; otherwise, it will repeat the probing process with another node, up to a total of five probes. The reason for using up to five probes is based on the finding in [8]. The buddy set size is chosen to be 10, since the performance improvement by increasing the buddy set size beyond 10 was shown to be insignificant [7].

In the bidding algorithm, a V–state node will broadcast a request-for-bid message to all other nodes, and U–state nodes will reply with their bids. If every U–state node replies with the same bid, the V–state node will randomly accept one of the bids received—which is similar to the random selection policy. However, if U–state nodes have different load states (so different bids), the U–state node with the least workload (highest bid) will be selected by all V–state nodes, so some of the transferred tasks have to be retransferred if the total number of overflow tasks received exceeds the capacity of this U–state node. In the drafting algorithm, two more messages will be exchanged between the U–state and V–state nodes to avoid "dumping" overflow tasks to the same U–state node. That is, the U–state node can inform the rest of V–state nodes of its refusal (in being their receiver) when it moves into state F, thus avoiding the need of task retransfer.

The coordination problem is analyzed when V–state nodes are randomly distributed in the system. It is necessary to introduce the following notation:

- f: total number of fully loaded nodes in the system
- k_s : total number of overloaded nodes in the system
- k_b : average number of overloaded nodes in a buddy set
- σ: size of a buddy set
- $(\overline{c_t})c_t$: (average) number of task collisions
- $P_i(k)$: probability of i task collisions when there are k overloaded nodes
- $P_0^{(i)}$: probability for the ith overloaded node to find an underloaded node within five probes.

A. Probing Policy

Under this policy, an overloaded node randomly probes up to five nodes and transfers an overflow task to the first underloaded node found during the probing process [8]. Since the number of messages exchanged in the probing policy is independent of the physical location of the nodes involved, this policy works best when the entire system is probed.

Our goal is to guarantee each V–state node to find a U–state node when the number of overloaded nodes is not greater than that of underloaded nodes in the system. To meet this goal, whenever a U–state node accepts an overflow task, it will refuse, even if it is still underloaded, to accept any more overflow tasks unless it is instructed otherwise. If a V–state node cannot find a U–state node which has not yet received any overflow task within five probes, it can either process this task locally or transfer this task to one of the U–state nodes which had already accepted an overflow task. In either case, a task collision is considered to have occurred in our model. The probability that each of k_s overloaded nodes can find an underloaded node is given by:

$$P_0^{(1)} = \sum_{i=1}^{5}\left[\prod_{j=1}^{i-1}\frac{f+k_s-j}{N-j}\right]\frac{N-f-k_s}{N-i}$$

$$P_0^{(m)} = \sum_{i=1}^{5}\left[\prod_{j=1}^{i-1}\frac{f+k_s-j+m-1}{N-j}\right]\frac{N-f-k_s-m+1}{N-i}$$

for $m = 1, \ldots, k_s$. Then the probability, $P_{c_t}(k_s)$, of $c_t = m (1 \le m \le k_s)$ can be calculated as:

$$P_0(k_s) = \prod_{i=1}^{k_s} P_0^{(i)} \qquad (6)$$

$$\begin{aligned} P_1(k_s) &= \sum_{i=1}^{k_s}\left[\prod_{j=1, j\neq i}^{k_s}\left(1-P_0^{(i)}\right)P_0^{(j)}\right] \\ &\vdots \\ P_m(k_s) &= \sum_{x_1=1}^{k_s}\cdots\sum_{x_m=x_{m-1}}^{k_s}\left[\prod_{y=1, y\neq x_1,\ldots,y\neq x_m}^{k_s}\left(1-P_0^{(x_1)}\right)\cdots\left(1-P_0^{(x_m)}\right)P_0^{(y)}\right]. \end{aligned} \qquad (7)$$

Although $\overline{c_t} = \sum_{i=1}^{k_s} iP_i(k_s)$ can be calculated with (6), the nested summations in these equations require a prohibitive amount of computation. The required computation can be reduced significantly if there is one representative, approximate value (e.g., average value) for all $P_0^{(i)}$s. Equation (6) can then be simplified as

3. Among underloaded nodes.

$$P_i(k_s)=\binom{k_s}{i}\left(1-P_0^{(j)}\right)^i\left(P_0^{(j)}\right)^{k_s-i} \quad (8)$$

for $i = 1, ..., k_s$. To get a better approximate, j can be set to $k_s/2$, or $P_0^{(j)}$ can be replaced by the average value of all these terms. Comparison between the approximate and simulation (exact) results for 6– and 8– cubes is summarized in Table I. The approximation with the average value is shown to be close to the simulation results except when the number of overloaded nodes reaches $N/2$. In such a case, the median–value approximation works better. One explanation of this result is that when the number of overloaded approaches $N/2$, the last few overloaded nodes are almost impossible to find underloaded nodes, because most of the underloaded nodes are marked as unavailable after they were probed by the overloaded nodes. Thus, using the average value to represent the collision probability for every underloaded node will increase the collision probability. Note that even when the number of overloaded nodes is approaching $N/2$, for most overloaded nodes it is still possible to probe another underloaded node unless most of the underloaded nodes have already been probed and labeled as unavailable. So, the median-value works better than the average value when the number of overloaded nodes approaches $N/2$.

Although (6) is derived in a sequential manner, it gives the same result as when all overloaded nodes execute the probing procedure in parallel. Suppose an underloaded node receives more than one probe at the same time, then this node should reply to only one probe when it is in U–state, whereas it must reply to all other probe requests when it is in F– or V– state; otherwise, this node may receive more than one overflow task and become overloaded. So, although different nodes run the probing procedure concurrently in practice, the probability of probe collisions remains the same as the case when the probing is done sequentially by the V–state nodes.

B. Random Selection

Under this policy, an overloaded node randomly selects one of the underloaded nodes within its buddy set. A collision will occur only when the buddy sets of two overloaded nodes overlap. Since this method is considered only to show the advantages of the PLSC location policy, the candidate receiver nodes are assumed to be in the same buddy set. So, the first step is to determine the intersection of two buddy sets. According to [7], buddy sets of 10 to 15 nodes yield good results, and thus the intersection of any two nodes' buddy sets can be approximated under the assumption that each node's buddy set is formed with only its immediate neighbors.

THEOREM 4. *The number of ways the buddy sets of any two nodes overlap is $\binom{n}{2}$, and there are only two nodes in the intersection of the two overlapping buddy sets.*

PROOF. If the buddy set of a node is composed of only its immediate neighbors, then the nodes in a node i_0's buddy set can be determined by $i_0 \oplus I_p$ $(0 \le p \le n-1)$ according to Definition 2. The buddy sets of nodes $i_0 \oplus I_j \oplus I_k$ $(0 \le j \le n-1$ and $j+1 \le k \le n-1)$ will overlap with node i_0's buddy set when $p = j$ or $p = k$, because $(i_0 \oplus I_j \oplus I_k) \oplus I_k = i_0 \oplus I_j$. So, there are a total of $\binom{n}{2}$ ways two buddy sets overlap, and there are only two nodes in their intersection, i.e., when $p = j$ and $p = k$. u

TABLE I
TASK COLLISIONS UNDER THE RANDOM PROBING POLICY

Overloaded nodes \ Eval. methods	simulation	approximation (median)	approximation (average)
2	0.000	0.0000	0.0000
6	0.000	6.573e-05	6.573e-05
10	0.007	0.0038	0.0038
14	0.074	0.0419	0.0419
18	0.293	0.2271	0.2297
22	0.997	0.7773	0.8709
26	2.532	1.7381	2.5484
28	3.710	2.6857	4.0582
30	5.288	4.4459	6.3214
32	7.252	6.9511	9.5599

$N = 64$

Overloaded nodes \ Eval. methods	simulation	approximation (median)	approximation (average)
10	0.000	3.131e-06	3.131e-06
20	0.001	0.0004	0.0004
40	0.027	0.0318	0.0318
70	1.070	0.8793	1.0298
80	2.354	1.6096	2.3334
90	4.653	3.2155	4.7855
100	8.323	6.5668	9.1233
110	13.803	11.8384	16.3212
120	21.625	20.1099	27.7301
125	26.348	25.4312	35.5508

$N = 256$

THEOREM 5. *The number of ways the buddy sets of k nodes overlap is $2(n-k+1)$ for $k = 3, ..., n$, and there is only one node in the intersection of these k buddy sets.*

PROOF. From Theorem 4, if the buddy sets of k nodes overlap, these nodes must contain a node i_0 and the nodes which are two hops away from node i_0. One can choose the first $k-1$ nodes as i_0, $i_0 \oplus I_p \oplus I_a$, $i_0 \oplus I_p \oplus I_b$, ..., and $i_0 \oplus I_p \oplus I_x$, where $a, b, ..., x$, and p are distinct integers between 0 and $n-1$ (both 0 and $n-1$ inclusive). Then these $k-1$ nodes will have a common node, $i_0 \oplus I_p$, in their buddy sets. There are $n-k+1$ other nodes, $i_0 \oplus I_p \oplus I_k$, $0 \le k \le n-1$, $k \ne p$,

$a, b, \ldots, x$, whose buddy sets contain the same node, $i_0 \oplus I_p$. So, there are $n - k + 1$ ways the buddy sets of k nodes intersect at node $i_0 \oplus I_p$. Similarly, one can show that there are $n - k + 1$ nodes, $i_0 \oplus I_q \oplus I_k$, $0 \leq k \leq n - 1$, $k \neq q, a, b, \ldots, x$, whose buddy sets intersect, at node $i_0 \oplus I_q$, those of nodes i_0, $i_0 \oplus I_q \oplus I_a$, $i_0 \oplus I_q \oplus I_b$, $i_0 \ldots$, and $i_0 \oplus I_q \oplus I_x$, where $p \neq q$. So, there are $2\,(n - k + 1)$ ways the buddy sets of k nodes overlap. u

Once the intersection of buddy sets is determined, one can calculate the average number of collisions when an underloaded node is selected each time in a buddy set as:

$$k_b = \frac{\sigma k_s}{N}$$

$$\overline{c_t} = \sum_{i=2}^{\sigma} \left[\binom{k}{i} \frac{2c_i}{(\sigma - k_b)^i} \left(\frac{\binom{\sigma - 2}{k_b}}{\binom{\sigma}{k_b}} \right)^{i-1} \right], \tag{9}$$

where $c_2 = \binom{n}{2} / \mathrm{N}$, $c_j = c_{j-1} \times \frac{n-j+1}{N-j+2}$, $3 \leq j \leq \sigma$. Note that if buddy set size is greater than the hypercube's dimension, a node's buddy set must contain nodes which are two or more hops away. Even in such a case, the above equation can be used to get approximate results, or the coefficients c_is need to be adjusted to get the exact value of c_t. Again, it should be noted that in the above analysis only U–state and V–state nodes are considered. One can follow the same procedure described at the end of Section III to calculate the task collisions in the presence of F–state nodes. Table II shows the numbers of task collisions obtained from the simulation and calculated by using (9).

C. Bidding and Drafting Algorithms

In the bidding algorithm, an overloaded node will request bids from all underloaded nodes, and it will choose a node with the least workload. Since no task can be transferred before a bid is accepted and acknowledged, no task collisions will result in this method. However, if there are more than one overloaded node in the system, a large number of request/reply messages will be generated in order to pair each overloaded node with a different underloaded node. So, we will use the total number of messages generated in the bidding algorithm to compare it with the SC and SP policies.

Let U_n and V_n be the number of underloaded and overloaded nodes at time t, and m be the total number of messages generated in the bidding process. To simplify the analysis, we restrict that the overloaded nodes can transfer only one task at a time. Thus, if an overloaded node with v tasks to be transferred, it needs to get v acceptable bids in v different rounds. Similarly, an underloaded node will accept one task in each round, and each underloaded node may have a different capacity before becoming fully loaded. Let T_v be the total number of overflow tasks in the overloaded nodes and T_u be the combined capacity of all underloaded nodes. Since LS works best when the system is lightly to medium-loaded, without loss of generality, we can assume $T_u > T_v$.

TABLE II
TASK COLLISIONS UNDER THE RANDOM SELECTION POLICY

Eval. methods / Overloaded nodes	simulation	approximation	difference (%)
2	0.021	0.0161	– 23.2
4	0.100	0.0988	– 1.1
6	0.248	0.2527	1.9
10	0.797	0.7934	– 0.4
14	1.651	1.6821	1.8
18	2.919	2.9700	1.7
22	4.677	4.7178	0.9
26	7.034	6.9956	– 0.5
30	9.944	9.8844	0.6
32	11.605	11.5857	– 0.2

$N = 64$, buddy set=10.

Eval. methods / Overloaded nodes	simulation	approximation	difference (%)
10	0.184	0.1809	– 0.3
20	0.793	0.7849	– 1.0
40	3.489	3.4081	– 2.3
70	11.469	11.5345	0.6
90	20.503	20.3685	– 0.6
100	25.947	26.0039	0.4
110	32.705	32.5518	– 0.5
120	40.113	40.0947	– 0.1
125	44.314	44.266	– 0.1

$N = 256$, buddy set=10.

The total number of rounds necessary for the bidding process is then T_v. The total number of request/reply messages is calculated as follows. In the first round, $U_n + V_n$ broadcast messages will be generated and at the end of the first round U_n and V_n may remain unchanged or reduced by one. Similarly, in the second round, $U_n + V_n$, or $U_n + V_n - 1$, or $U_n + V_n - 2$ broadcast messages will be generated and at the end of the second round, U_n and V_n may remain unchanged or reduced by one. In the last round, only two messages (in the best case) will be generated and T_v will become zero. So, we have the following equations. When $T_v = V_n$,

$$\frac{V_n(U_n + V_n + 2)}{2} \leq m \leq V_n(U_n + V_n). \tag{10}$$

When $T_u > T_v > V_n$,

$$m \geq \frac{V_n(U_n + V_n + 2)}{2} + \frac{(V_n + U_n + 2)(T_v - V_n)}{2}, \tag{11}$$

or

$$m \leq \frac{V_n(U_n + V_n + 2)}{2} + (V_n + U_n)(T_v - V_n). \tag{12}$$

In (11) and (12), the actual value of m depends on the distribution of the overflow tasks. When all the overflow tasks reside in one overloaded node, m is equal to an upper bound of (12). If the overflow tasks are evenly distributed among the overloaded nodes, m is close to a lower bound of (11).

In the drafting algorithm, underloaded nodes will initiate the drafting process and the procedure is the same as the bidding algorithm. Thus, the total number of messages generated in the drafting process can also be derived from (10) to (12). Note that the actual number of m derived from (10) to (12) depends on the actual distribution of the overflow tasks and the capacity of underloaded nodes. Thus, although m can be derived from these equations for both bidding and drafting algorithms, the actual value of m in the drafting algorithm will be much smaller than that of the bidding algorithm, as discussed in [11].

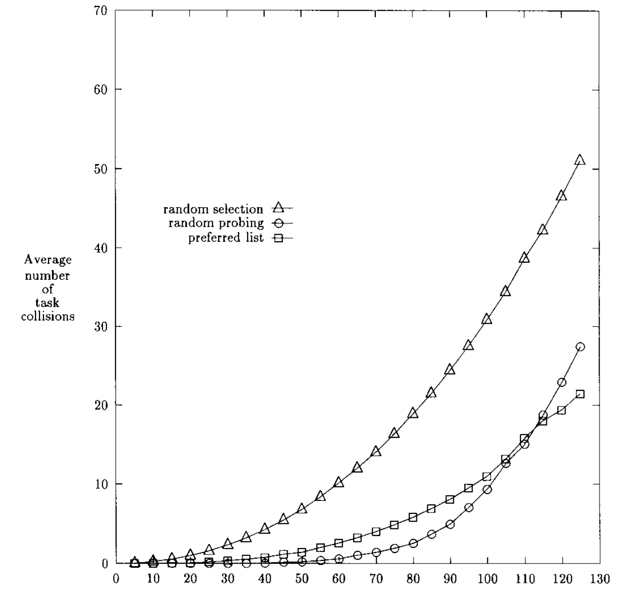


Fig. 5. Comparison of different location policies for $N = 256$.

D. Comparison of Performance of Different Location Policies

The performances of different location policies are compared in terms of the number of task collisions, number of rounds to offload overflow tasks, and communication overhead. A Q_8 is used for this comparison.

The numbers of task collisions under the preferred list, random selection, and probing policies are plotted in Figs. 5–7. Generally, the number of task collisions increases with the number of overloaded nodes in all of the three location policies. When the number of overloaded nodes is less than 0.15 N, there is no significant difference in the number of collisions among the three. However, when the number of overloaded nodes is close to 0.5 N, the number of collisions of the proposed location policy with preferred lists is about 30% of that of the random selection policy and 60% of that of the probing policy. When there is no F–state node, the probing policy performs as well as the preferred-list policy except when k approaches $N/2$. However, when the number of F–state nodes increases, the task collisions resulted from probing increase much faster than the preferred-list policy. As shown in Fig. 7, the probing results in more collisions than the preferred-list policy in all cases.

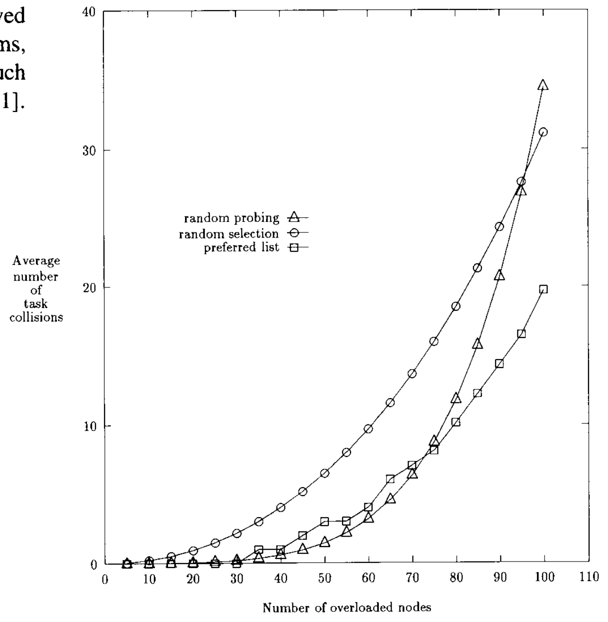


Fig. 6. Task collisions in three location policies (fully loaded nodes = 50).

It is important to observe that use of preferred lists requires no extra state information as compared to the random selection policy, but the number of task collisions resulting from the latter policy is more than three times that of the proposed policy. This result indicates the importance of coordinating overloaded nodes to locate underloaded nodes, which has been overlooked in existing local LS methods.

Task collisions under the probing policy are another interesting issue. The number of collisions is found to increase as the number of probes decreases. When only two (instead of five) probes are used, this policy results in even more collisions than the random selection policy. The number of collisions can be reduced by increasing the number of probes, which, in turn, increases the probing delay. (Note that two messages, one request and one reply, need to be exchanged for each probe.) Since an overloaded node cannot transfer an overflow task before locating an underloaded node, the probing delay will prolong the completion of the tasks to be transferred. It is shown in [8] that the delay resulting from using more than five probes outweighs the benefit that might be gained by transferring a task from an overloaded node to an underloaded node.

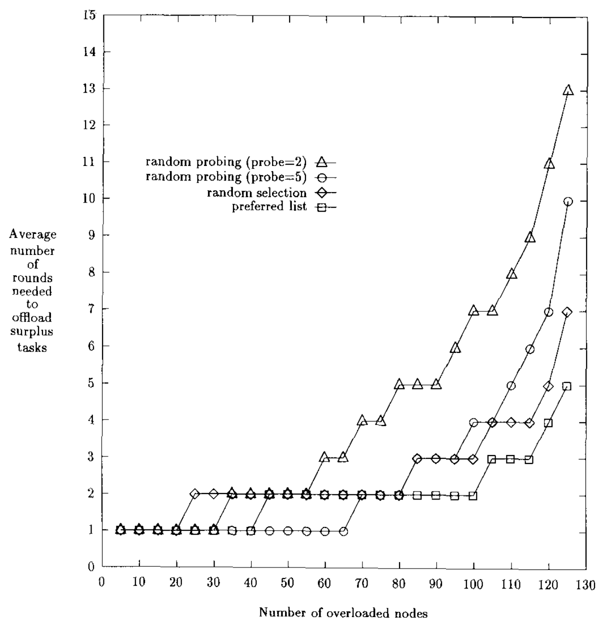


Fig. 7. Comparison of the number of rounds needed to offload the surplus tasks in three location policies.

TABLE III
THE NUMBER OF TASK COLLISIONS IN OVERLOADED SUBCUBES

$N = 64$ / Strategies	Average number of task collisions		
random selection	2.99	2.72	9.92
random probing	1.93	1.61	11.61
preferred lists	0.00	0.00	0.00
address of overloaded nodes	0 – 15	32 – 47	0 – 15, 32 – 47

$N = 256$ / Strategies	Average number of task collisions		
random selection	14.8	13.28	58.36
random probing	11.57	10.13	35.41
preferred lists	0.00	0.00	0.0
address of overloaded nodes	0 – 63	128 – 191	0 – 63, 128 – 191

The advantage of using preferred lists becomes more pronounced as the system gets congested. To see this tendency more clearly, suppose overloaded nodes are concentrated in an area forming a hot region. Table III shows the case when every node in a subcube is overloaded, where one to two Q_4s and one to four Q_6s are considered in 6- and 8- cubes, respectively. An interesting result can be found in Table III: Both the random selection and probing policies result in more collisions than the case when overloaded nodes are randomly distributed throughout the system. This indicates that both the random selection and probing policies cannot handle the congestion problem effectively. Unsurprisingly, the location policy with preferred lists has resulted in no collision at all in these cases, because if all the nodes in a subcube are overloaded, they will transfer their overflow tasks to the underloaded nodes in another subcube, thus eliminating the possibility of task collision.

TABLE IV
THE NUMBER OF TASK COLLISIONS IN OVERLOADED BUDDY SETS

$N = 64$ / Strategies	Average number of task collisions		
random selection	3.65	3.32	8.67
random probing	1.54	1.48	7.31
preferred lists	2.00	2.00	4.0
overloaded buddy set(s) of	node 0	node 10	node 1 and 12

$N = 256$ / Strategies	Average number of task collisions		
random selection	2.5	9.11	20.08
random probing	1.89	6.47	14.11
preferred lists	2.00	4.00	8.0
overloaded buddy set(s) of	node 0	node 1 and 63	node 0, 63, 128, 191

In Table IV, overloaded nodes are assumed to be concentrated in one to four buddy sets ("overloaded buddy sets"), where the address of the node whose buddy set is overloaded is given at the bottom row. Two collisions will always result if every node in a buddy set is overloaded. The first collision occurs, because the node with the overloaded buddy set will not be able to find any underloaded node to transfer an overflow task. A second collision occurs due to the nodes in the overloaded buddy set as proved in Theorem 3. The location policy with preferred lists always results in less task collisions, except for one overloaded buddy set under the random probing policy which results in slightly less collisions. Note that in case the system is congested, (9) needs to be adjusted to calculate the number of task collisions under the random probing and selection policies, because these equations are derived under the assumption that the overloaded nodes are randomly distributed throughout the system.

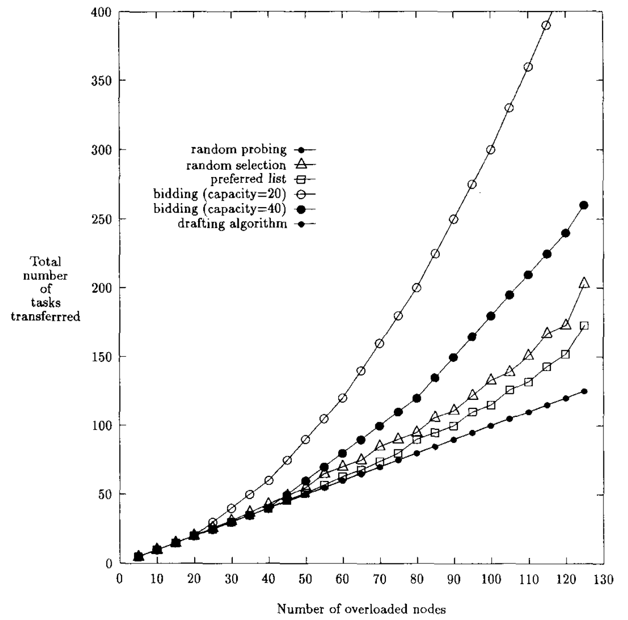


Fig. 8. Comparison of the number of tasks transferred in different location policies.

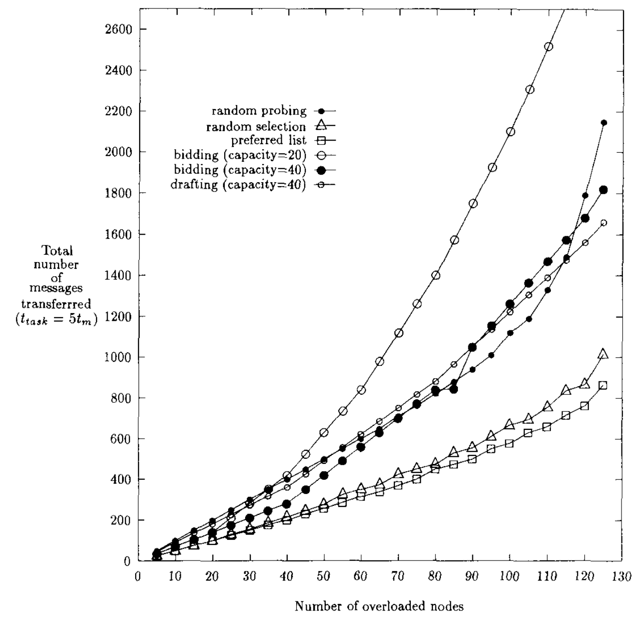


Fig. 9. Comparison of the number of messages exchanged in different location policies.

The number of rounds used to offload the overflow tasks is plotted in Fig. 7. It is clear that the preferred-list policy requires the least number of rounds when $k > 40$. The number of tasks actually transferred is plotted in Fig. 8 as a function of k. It should be noted that in both the probing and drafting algorithms, there is no retransfer of "collided" tasks, so the number of actually transferred tasks is equal to the number of overloaded nodes, assuming that each overloaded node has only one overflow task. Under all other policies the number of transferred tasks is always greater than the number of overflow tasks, but the preferred-list policy still performs best.

Note, however, that it is not fair to consider only the number of transferred tasks as the performance measure of a location policy, because under the probing and drafting policies many control messages need to be exchanged before transferring each overflow task. So, it is desirable to compare the performance of different location policies based on a total number of messages exchanged as shown in Fig. 9. The time to transfer a task is assumed to be two, five, and 10 times of the time of exchanging a control message between U–state and V–state nodes. It is easy to see from Fig. 9 that the preferred-list policy results in the least number of messages. When the time to transfer a task becomes much longer than that of exchanging messages, the gap between the preferred list, probing and drafting policies decreases. It is found that as long as the time to transfer a task is not greater than 20 times of that needed to exchange messages, the preferred-list policy will result in less communication delay than all the other policies.

In a recent paper by Shivaratri and Krueger [16], two adaptive probing methods were proposed to reduce the number of probes in the SP method by keeping the probe history of other nodes. These methods can reduce the number of probes when the load state of the nodes in the system doesn't change frequently. When the number of overloaded nodes approaches 50% of the total number of nodes, these methods will perform very similar to the random probing, because the past probe history may not reflect the actual load state of the nodes at all.

V. Conclusion

We proposed and analyzed a new PLSC location policy for load sharing in hypercube-connected multicomputers based on state–change multicasts in each buddy set. The coordination and congestion problems are solved by systematically generating and then equipping preferred lists with each node. The proposed location policy is shown to minimize the number of task collisions, and its superiority to other location policies becomes more pronounced when one or more hot regions are formed in the system.

One potential application of the LS with the proposed location policy is the distribution and/or redistribution of tasks to meet timing constraints. Since each underloaded node will accept at most one task at a time from an overloaded node in our task collision model, a transferred task can be completed before its deadline with a high probability. However, the proposed algorithm which generates preferred lists only works on hypercube topology, it is desirable to generalize this algorithm for other interconnection topologies. For example, on a bus connected distributed system, such as the Ethernet, a binary address code can be assigned to each node. Then, the same algorithm can be used to generate the preferred lists and buddy sets to resolve the coordination and congestion problems. Generalizing this algorithm to other topologies is a worthy problem and warrants further investigation.

Acknowledgments

The work reported in this paper was supported in part by the Office of Naval Research under Grant N00014-92-J-1080, and the NSF under Grant MIP-9203895. Any opinions, findings, and recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the funding agencies.

References

[1] L.N. Bhuyan and D.P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Computers*, vol. 33, no. 4, pp. 323–333, Apr. 1984.
[2] C.L. Seitz, "The cosmic cube," *Comm. ACM*, vol. 28, no. 1, pp. 22–33, 1985.
[3] B. Becker and H.U. Simon, "How robust is the n-cube?" *Proc. 27th Ann. Symp. on Foundations of Computer Science*, pp. 283–291, 1986.
[4] M.-S. Chen and K.G. Shin, "Adaptive fault–tolerant routing in hypercube multicomputers," *IEEE Trans. Computers*, vol. 39, no. 12, pp. 1,406–1,416, Dec. 1990.
[5] P. Ramanathan and K.G. Shin, "Reliable broadcast in hypercube multicomputers," *IEEE Trans. Computers*, vol. 37, no. 12, pp. 1,654–1,657, Dec. 1988.
[6] M.-S. Chen and K.G. Shin, "Processor allocation in an n-cube multiprocessor using gray codes," *IEEE Trans. Computers*, vol. 36, no. 12, pp. 1,396–1,407, Dec. 1987.
[7] K.G. Shin and Y.-C. Chang, "Load sharing in distributed real-time systems with state change broadcasts," *IEEE Trans. Computers*, vol. 38, no. 8, pp. 1,124–1,142, Aug. 1989.
[8] D.L. Eager, E.D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Trans. Software Engineering*, vol. 12, no. 5, pp. 662–675, May 1986.
[9] R.G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Computers*, vol. 29, no. 12, pp. 1,104–1,113, Dec. 1980.
[10] K. Hwang, W.J. Croft, G. H. Goble, B.W. Wah, F.A. Briggs, W.R. Simmons, and C.L. Coates, "A unix-based local computer network with load balancing," *Computer*, vol. 15, no. 4, pp. 55–64, Apr. 1982.
[11] L.M. Ni, C.W. Xu, and T.B. Gendreau, "A distributed drafting algorithm for load balancing," *IEEE Trans. Software Engineering*, vol. 11, no. 10, pp. 1,153-1,161, Oct. 1985.
[12] D.J. Farber, J. Feldman, F.R. Heinrich, M.D. Hopwood, K.C. Larson, D.C. Loomis, and L.A. Rowe, "The distributed computing system," *IEEE COMPCON Spring*, pp. 31–34, 1973.
[13] W. Zhao, K. Ramamritham, and J.A. Stankovic, "Scheduling tasks with resource requirements in hard real-time systems," *IEEE Trans. Software Engineering*, vol. 13, no. 5, May 1987.
[14] R.M. Bryant and R.A. Finkel, "A stable distributed scheduling algorithm," *IEEE Proc. Second Int'l Conf. on Dist. Comp. Systems*, pp. 314–323, 1981.
[15] F. Harary, *Graph Theory*. Boston, Mass.:Addison-Wesley, 1969.
[16] N.G. Shivaratri and P. Krueger, "Two adaptive location policies for global scheduling algorithms," *IEEE Proc. Fifth Int'l Conf. on Dist. Comp. Systems*, pp. 502–509, 1990.

Kang G. Shin received the BS degree in Electronics Engineering from Seoul National University, Seoul, Korea, in 1970 and both the MS and PhD degrees in Electrical Engineering from Cornell University, Ithaca, New York, in 1976 and 1978, respectively. He is professor and director of the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor.

From 1978 to 1982 he was on the faculty of Rensselaer Polytechnic Institute, Troy, New York. He has held visiting positions at the U.S. Airforce Flight Dynamics Laboratory, AT&T Bell Laboratories, Computer Science Division within the Department of Electrical Engineering and Computer Science at UC Berkeley, and International Computer Science Institute, Berkeley, CA. He also chaired the Computer Science and Engineering Division, EECS Department, The University of Michigan for three years beginning in January 1991.

He has authored/coauthored over 300 technical papers (more than 140 of these in archival journals) and several book chapters in the areas of distributed real-time computing and control, fault-tolerant computing, computer architecture, robotics and automation, and intelligent manufacturing. In 1987, he received the Outstanding IEEE Transactions on Automatic Control Paper Award for a paper on robot trajectory planning. In 1989, he received the Research Excellence Award from The University of Michigan. In 1985, he founded the Real-Time Computing Laboratory, where he and his colleagues are currently building a 19-node hexagonal mesh multicomputer, called HARTS, to validate various architectures and analytic results in the area of distributed real-time computing.

He has also been applying the basic research results of real-time computing to intelligent transportation systems and manufacturing-related applications ranging from the control of robots and machine tools to the development of open architectures for manufacturing equipment and processes. Recently, he has initiated research on the open-architecture Information Base for machine tool controllers.

He is an IEEE fellow, was the program chairman of the 1986 IEEE Real-Time Systems Symposium (RTSS), the general chairman of the 1987 RTSS, the guest editor of the 1987 August special issue of *IEEE Transactions on Computers* on Real-Time Systems, a program co-chair for the 1992 *International Conference on Parallel Processing*, and served on numerous technical program committees. He also chaired the IEEE Technical Committee on Real-Time Systems during 1991-93, is a Distinguished Visitor of the Computer Society of the IEEE, an Editor of *IEEE Trans. on Parallel and Distributed Computing*, and an Area Editor of *International Journal of Time-Critical Computing Systems*.

Yi-Chieh Chang (S' 84) received the BS and MS degrees in electrical engineering from National Taiwan University, Taipei, Republic of China, in 1979 and 1984, respectively. He received his PhD degree in electrical engineering and computer science from the University of Michigan, Ann Arbor in 1991. He is currently an assistant professor in the Electrical and Computer Engineering Department at the University of Texas at El Paso. He has published more than 20 technical papers in the areas of reconfigurable fault-tolerant array processors, parallel computer architecture, and distributed real-time systems. His research interests include computer architecture, VLSI systems, parallel processing, and distributed real-time systems.