A Semi Distributed Task Allocation Strategy for Large Hypercube Supercomputers *

Ishfaq Ahmad
School of Computer and Information Science,
Syracuse University, Syracuse, NY 13244

Arif Ghafoor
Department of Electrical and Computer Engineering
Syracuse University, Syracuse, NY 13244

* *Best Student Paper Award - Systems*

Abstract

This paper presents a semi distributed approach for task scheduling in large parallel and distributed systems which is different from the conventional centralized and fully distributed approaches. The proposed strategy partitions the system into independent regions (spheres) centered at some control points. The central points called schedulers, optimally schedule tasks within their spheres and maintain state information with low overhead. We consider Hypercube systems for evaluation and using its algebraic characteristics, show that identification of spheres and their scheduling points is an NP-complete problem. An efficient solution for this problem is presented by making an exclusive use of a combinatorial structure known as Hadamard Matrix. Performance of the proposed strategy was evaluated and compared with an efficient fully distributed strategy, through simulation. In addition to yielding high performance in terms of response time, better resource utilization and throughput, the proposed strategy is shown to incur small overhead in terms of network traffic.

1. Introduction

As a result of evolutionary advancements in computation and communication technology, one can foresee future supercomputers consisting of thousands of processors [8]. In addition to providing enhanced availability, reconfigurability and resource sharing, these massively parallel systems can theoretically multiply the computational power of a single processor by a large factor. The key advantage of such systems is that they allow concurrent execution of tasks which can be independent programs or partitioned modules of a single program. From a performance perspective, task response time, throughput and better resource utilization are critical measures that need to be optimized while keeping the control overhead within a reasonable value. These performance measures pose many challenges, one of which is an efficient scheduling of tasks on the computing nodes. Inefficient scheduling can lead to a load imbalance on various nodes which can significantly increase the response times of tasks scheduled at heavily loaded nodes. Dynamic task scheduling has gained considerable attention [5], [6], [7], [11], [23] as a technique to overcome this problem since it takes into account the time dependent fluctuations in the load patterns across the system. Most of the existing dynamic scheduling strategies are, in essence, based on two extreme solutions, that is centralized [7], [20] and fully distributed models [1], [6], [11], [15], [22].

In this paper we propose a new approach that is semi-distributed in nature. With this approach, we introduce the notion of sphere of locality where each sphere, with a central control, is a group of nodes. Accordingly, a scheduling algorithm is presented that is executed by only a set of nodes, called schedulers. Each sphere is centered at a scheduler for that sphere and each center is at equal distance from all other, except one, centers. Tasks are optimally scheduled within the sphere or transported to other spheres, if required. We also propose an information building algorithm that keeps the accurate information about load distribution within a sphere. Both scheduling and information building algorithms have low overheads. The number of schedulers which need to execute the scheduling and information building algorithm is relatively small - incurring a low control overhead. At the same time the schedulers are sufficiently enough to effectively manage load within their spheres. We show that, in general, optimal determination of the centers or schedulers is an NP-complete problem. However, for Hypercube networks, we propose an efficient semi distributed design based on a combinatorial structure known as Hadamard Matrix. The technique also applies to other topologies which are *distance regular* [13]. Through simulation study, we show that, for large Hypercube systems, the proposed strategy yields better performance in terms of response time, throughput and resource utilization. We compare the results with an efficient fully distributed scheduling algorithm. The impact of communication delay and message exchange overhead is also evaluated.

The rest of the paper is organized as follows. In the next section we present a brief overview of existing scheduling and load balancing strategies. We discuss their limitations for large systems. In Section 3, we give an algebraic characterization of Hypercube systems and state the problem of constructing a set of spheres. The system model using the semi distributed structure is also presented in that section. In Section 4, we describe the simulation environment. Results and comparisons are given in Section 5. Section 6 concludes this paper.

2. Related Work and Motivation for a New Approach

The task scheduling problem has been extensively reported in the literature. Essentially, scheduling techniques can be classified into two categories. In the first category, a set of communicating tasks is considered for scheduling on the system nodes before run time. For a set of tasks with a priori knowledge, the scheduling problem is better described as the assignment or the mapping problem which has been commonly solved with graph theoretic models such as network flow [4] and heuristics [11]. These types of techniques have also been termed as static scheduling techniques.

The second class of scheduling, frequently based on queueing theory, assumes a continuous arrival of tasks which may be inde-

CH2916-5/90/0000/0898/$01.00 © IEEE

pendent processing modules or a set of related modules with or without communication requirements. These types of strategies, do not assume a priori knowledge about the tasks and are also known as dynamic load balancing strategies. The main objective in such strategies is to balance the load across the system for a faster response time and a better throughput. Most of the existing dynamic load balancing techniques employ centralized model [7], [20] or fully distributed model [1], [6], [9], [15], [23]. In a centralized model, a dedicated node gathers the global information about the state of the system and schedules the tasks to individual nodes. On the other hand, in a fully distributed model, each node executes a scheduling algorithm by exchanging state information with other nodes. Fully distributed scheduling strategies employing heuristics [9], [11] have been shown to yield good performance. In some strategies, the load transfer can be initiated by the under loaded nodes[15], [16]. Simulation methodologies [1], [6], [21], [22], [23] have gained wide attention as the mean for predicting the performance of such strategies. A number of load balancing algorithms are compared in [27] using a simulation model that takes into account the traces of actual job statistics. Topology dependent models have been reported in [26]. A hierarchical scheme has also been proposed [25] but this scheme has many drawbacks such as high overhead and underutilization of the system resources. A more detailed classification of scheduling techniques can be found in [2]. Most of these schemes are efficient and yield a near optimal performance for a system size ranging from two nodes [4] to a few tens of nodes [27].

Not much attention has been paid to designing strategies for large systems consisting of thousands of nodes. For these systems, centralized and fully distributed schemes are not very suitable due to the following reasons:

- With fully distributed schemes, optimal scheduling decisions are hard to obtain [1], [6]. This is due to a great degree of randomness involved in a rapidly changing environment. Since, in a system comprising a number of nodes, each node makes autonomous decisions, no node can envision the exact state of the whole system. Thus, a node has to make either a probabilistic decision or i has to make some kind of guess about the state of the system.
- The second problem with fully distributed models is that communication delays can render a scheduling decision wrong. For instance, a task after going through communication set up, queueing and transmission delays over multiple links may find out that the destination node that was originally conceived of as the best choice has become the most heavily loaded due to arrivals from some other nodes [1]. These scenarios result in occasional wrong decisions and system can enter into the state of instability or task thrashing [1], [11] when tasks keep on migrating without getting settled at one node.
- Fully distributed algorithms use small amount of information about the state of the system. Since gathering large amount of state information may decrease the accuracy [6], it becomes more appropriate to collect small amount of more accurate information. Small systems can yield good performance with limited information [6], [9] but this may not be true for large systems. If the most heavily and the most lightly loaded sections of a large network (with a diameter of the order of logarithm of the number of processors) are a large distance apart, a fully distributed algorithm, with limited amount of information may have to be executed for a number of iterations to balance the load among these sections. Reduced amount of information results in a smaller range of scheduling options and hence a narrow scope of sending a task to a suitable node.
- Despite the fact that fully distributed algorithms incur less overhead due to message exchange, this overhead linearly increases with the system size [2], [27]. This may result in a proportional increase in the average response time and extra communication traffic which can impede regular communication activity.
- With a few exceptions [11], most researchers have ignored the overhead of the scheduling algorithm which can delay the processing of a task. This overhead is also a linearly increasing function of the system size.
- Saturation effect in large distributed systems is a widely cited phenomenon [11], [27] which deteriorates the performance as the system grow in size. An incremental increase in system resources may result in a decrease in throughput [11]. It is known that for a distributed algorithm the response time does not improve with an increase beyond a few tens of nodes [27].
- Centralized algorithms do have the potential of yielding optimal performance [7], [20], [27], but with a large system, the global information collection becomes a formidable task [22].
- The storage requirement for maintaining the state information also becomes prohibitively high with a centralized model of a large system.
- In a large system with a single centralized control a huge number of processors, the controlling node can become the bottleneck and hence can lower the throughput [2].
- Centralized models are also highly vulnerable to failures. The failure of any software or hardware component at the control point can cause malfunctioning of the whole system.

It has been shown [27], that neither fully distributed nor centralized strategies always yield optimal performance. It is, therefore, expected that there exists a trade-off between centralized and fully distributed scheduling mechanisms. The aim of this paper is to present a new approach that employs a semi-distributed scheme exploiting the advantages of both centralized and fully distributed models.

3. A Semi-Distributed Model for Scheduling

The proposed scheme is a two level scheduling scheme. At the first level, tasks can migrate between different regions of the system providing a distributed environment among *spheres*. At the second level, scheduling takes place within individual *spheres* where the scheduler of each *sphere* acts as a centralized controller for its own *sphere*. We choose Hypercube systems for our experimentation, firstly because these systems are commercially available and secondly because algebraic characterization of Hypercube topology can be readily exploited for our analysis.

First, we briefly present some algebraic characteristics for the Hypercube system and introduce the notion of *sphere of locality*. Next, we explain the methodology of selecting a set of nodes which act as central control points for scheduling their respective *spheres*. We show that an optimal design of a semi-distributed control is an NP-complete problem. Subsequently, we propose an efficient solution, for finding the set of schedulers, based on a combinatorial structure called *Hadamard design*.

3.1. Mathematical Preliminaries

In this section, we present some definitions and terminology which are used to characterize the binary n-cube topology and its property of *symmetry*. We denote this topology as Q_n and represent it by an undirected graph, $\Lambda = < U, E >$ where U represents the set of nodes and E is the set of edges (communication links) joining the nodes. Let 0, 1, 2, ..., $(N-1)$ denote the set of

$N = 2^n$ nodes. This model is an abstraction of inter-processor communication patterns, which is equally applicable to large multicomputer networks with 2^n nodes. These results can be extended to a large class of parallel systems which have distance regular network topologies [14] (distance regularity is described below). In that case, we can assume that there exist a virtual topology for the network which is isomorphic to a Hypercube topology. If the number of nodes is not a power of 2^n, pseudo nodes can be added to make the topology appear like a Hypercube [14].

The degree of each node represents the number of edges incident on it. The degree is assumed to be constant and is equal to n for a binary n-cube. A *path* in Q_n is a sequence of connected nodes and the length of the shortest path between nodes i and j is called the *graphical distance* and is represented as L_{ij}. The path in the network having such a distance is termed as the *shortest path*. Let

$k = Max[L_{ij} | \forall i,j, 0 \leq i,j, \leq N-1]$.

k is called the *diameter* of the network Λ. For Q_n, $k = n$.

Definition: Any two nodes which are at distance n, the diameter of the network, apart are called an *antipodal pair.* The codewords of an *antipodal pair* are complement of each other.

Definition: Given a set of nodes C, its *graphical covering radius* r in the graph Λ is defined as:

$r = Max_{i \in U}(Min_{j \in C}(L_{ij}))$

Definition: A graph of diameter k is said to be distance-regular if

$$\forall (i,j,l) \in [0 \ldots n]^3, \quad \forall (x,y) \in U,$$
$$L_{xy} = l \Leftrightarrow |\{z \in U, L_{xz} = i, L_{yz} = j\}| = p_{ij}^l$$

Where $| y |$ represents the cardinality of some set y and p_{ij}^l are constants whose values are determined by the characteristics of the graph. The parameters p_{ij}^l for Q_n are given as

$$p_{ij}^l = \begin{cases} \binom{l}{\frac{i-j+l}{2}} \binom{n-l}{\frac{i+j-l}{2}}, & \text{if } i+j+l \text{ is even} \\ 0, & \text{if } i+j+l \text{ is odd} \end{cases}$$

Distance-regularity is an important property in terms of describing *sphere of locality* which in turn defines the range of a *scheduler* and gives an estimate of the number of test messages generated for gathering state information.

Definition: Let v_i^j be the number of nodes which are at a graphical distance i from a node j. This number is a constant for $\forall j \in U$ in Q_n and is called the i-th valency. It is given as $v_i^j = p_{ij}^0 = \binom{n}{i}$ This value is consistent with the expression for p_{ij}^l. Let a be a binary codeword of length n and $w(a)$ be the weight (the number of 1's in a) of this codeword. Then for Q_n

$$\underline{0} = \text{all 0's}, \qquad v_i^0 = \{ a \mid w(a) = i \},$$
$$v_i^j = \{ l = a + j \mid \forall a \in v_i^0 \}$$

Definition: A Hadamard matrix M is a j by j matrix with ± 1 entries, such that $MM^T = jI$, where I is the identity matrix and M^T is the transpose of M. The *complementary Hadamard matrix*, denoted as M^C, is obtained by multiplying all the entries of M by -1. Hadamard matrices can exist only if j is a multiple of 4. If we replace 1 by 0, and -1 by 1, the matrix is said to be in 0-1 *notation.* We will refer to this matrix as *Hadamard matrix M*, and use the 0-1 notation in the rest of this paper. Figure 1 shows an untruncated 8×8 Hadamard matrix and its complement, using 0-1 notation.

It is known that Hadamard matrices of order up to 428 exist. Furthermore, it has been conjectured that a Hadamard matrix of order n exists if n is 1, 2 or a multiple of 4. Various methods of generating Hadamard Matrices include Sylvester's method, Paley's construction and the use of Symmetric Balanced Incomplete Block Designs (SBIBD) [18].

3.2. Hypercube Partitioning

In this section, we discuss the criteria for partitioning of Hypercube and show that the selection of a subset of nodes, referred as schedulers in Q_n, for the purpose of carrying out scheduling can be modeled as a problem which is NP-hard. Assuming that, somehow, the network is partitioned into spheres, we then propose a semi distributed scheduling mechanism. This is achieved by dedicating one scheduler for each sphere and making it responsible for (a) assigning tasks to individual nodes of the sphere, (b) maintaining the load status of the sphere and nodes, and (c) transferring load to other spheres, if required. A scheduler is responsible for optimally assigning tasks within its sphere depending upon the *range* of the scheduler which is, in turn, determined by the size of the sphere. The load status of each node of the sphere and the accumulative load of the sphere is known to the scheduler. Tasks can also migrate between spheres depending upon the degree of imbalance in sphere loads. To carry out load exchange at inter sphere level, a scheduler needs to communicate with the schedulers of other spheres for exchanging load status of the spheres. The exchange and maintenance of load information, among spheres, is carried out with an efficient method which is fast and has a low overhead. The details of the scheduling algorithm and information maintenance scheme are described in section 3.4.4.

Let C be the desired set of scheduling nodes. There can be various possible options to devise a semi distributed scheduling strategy based on this set, but the performance of such a strategy depends on the "graphical locations" of the scheduler nodes (distances between them) of C in Q_n and the range of scheduling used by these nodes. The *range of scheduling* quantifies the graphical distance within which a scheduler assigns tasks to the nodes of the sphere.

Definition: Let the sphere assigned to a node $j \in C$, be denoted by $S_i(j)$, where i is the radius of this sphere. The number of nodes in $S_i(j)$ is the total number of nodes lying at graphical distances 0 through i, from j. Since the number of nodes at the graphical distance i is given by valency v_i^j, the total size of the sphere is given as $|S_i(j)| = \sum_{k=0}^{i} v_k^j$. It should be noted that, in a centralized scheme, i must be equal to the diameter (k) of the network and in a fully distributed scheduling, using a local information among neighbors, i is equal to 1.

Definition: A uniform set C, of centers, is the maximal set of nodes in Λ, such that the graphical distance among these centers is at least δ and $|S_i(j)|$ is constant (uniform) $\forall j \in C$, where i is the covering radius of C.

We need a δ -uniform set C (for some δ to be determined) with graphically symmetric spheres, in order to design a symmetric scheduling algorithm. The size of $|C|$ depends on the selection of δ. Intuitively, larger δ yields smaller $|C|$ but larger size of the sphere. In addition, a number of other considerations for scheduling are given below.

(1) Since a scheduler needs to migrate tasks to all the nodes in the sphere, the diameter of the sphere should be as small as possible.
(2) $|C|$ should be small, so that the global overhead of scheduling

algorithm in terms of message exchange, maintenance of information and storage requirements is small.

(3) The size of the sphere should be small, firstly, because j needs to send/receive $|S_i(j)|$ messages and, secondly, because information storage and maintenance requirements increase with the increase in sphere size. However, we can notice that reducing $|C|$ increases the sphere size. We now describe the complexity of selecting a δ-uniform set (C).

Theorem 1: For a given value of $\delta > 2$,

(a) Finding $|C|$ is NP-hard.

(b) Determining the minimum sphere size is also NP-hard.

Proof: For the proof of (a), see [24]. Finding the minimum sphere size, $|S_f(j)| \forall j \in C$, requires us to determine the minimum value of f, which is equal to the covering radius of the set C. Finding the covering radius of a code C in Q_n(and for any sub space of Q_n) is NP-hard problem [17]. Since finding the minimum sphere size is equivalent to finding the covering radius, the complexity of the whole problem will not be less than NP-hard.

3.3. Identification of Scheduler Sites

For Hypercube systems, we present an efficient solution for determining the set C. The solution is "efficient" in the sense that the size of the selected set C is considerably small and it is of the order of logarithm of the size of the $Q_n$network . This results in a considerably small storage requirement. At the same time, the whole network is uniformly covered and each sphere is symmetric and equal in size. The set C of scheduler nodes is selected from the code generated by the rows of Hadamard matrix M and its complement M^C. The set C is also called *Hadamard* code. Note, $|C| = 2n$. Following are the main reasons for this selection (we might as well select other codes such as Hamming code or BCH codes, but these codes have certain limitations as described below).

1. A Hadamard code is a code with rate approaching zero, asymptotically [24] where rate of a code C with length n is defined as $n \xrightarrow{\lim} \infty (\log_2 |C|/n)$[18]. This results in the size of a Hadamard code being considerably smaller than the size of a Hamming code in a Q_n. In fact, the size of Hadamard code is proportional to the *logarithm* of the size of the network Q_n. On the other hand, the rate of a Hamming code is 1, which results in a large size of the code and hence the set $|C|$.

2. The range of values of n for which a Hadamard code exists, considerably exceeds the range of n for which a Hamming code exists. For example, it is conjectured that a Hadamard matrix exists for all values on n which are multiple of 4. The smallest order for which a Hadamard matrix has not been constructed is 428 [18]. On the other hand an extended Hamming code only exists if n is a power of 2. Similarly, a BCH code exists only for limited values of n.

3. The covering radius of C is known [18] for many values of n which are even powers of two.

Due to the above mentioned advantages, we use Hadamard codes to construct the set C. Since, an untruncated Hadamard code exists only when n is a multiple of 4, selection of the set C can be made by modifying this untruncated code in various ways, to form the remainder of values. These modifications are described below.

Case a: Q_n with $n \bmod 4 = 1$. For this case, consider the set C obtained from untruncated Hadamard matrices M and M^C(in 0–1 notation) of size $n-1$. By appending an all 0's and an all 1's column, to M and M^C respectively, at any fixed position, say at extreme left, we generate the modified set C for the network under consideration.

Case b: $Q_n$with $n \bmod 4 = 2$. This case is treated the same way as the Case (a), except we consider the set C obtained from untruncated Hadamard matrices (in 0–1 notation) of size n–2 and append two columns 0 and 1 to M and 1 and 0 to M^C. However, the all 0's row in M is augmented with bits 00 rather than with bits 01. Similarly, the all 1's row in M^C is augmented with bits 11 rather than with bits 10.

Case c: $Q_n$with $n \bmod 4 = 3$. For this case, the set C consists of the rows of the truncated matrices M and M^C in 0–1 notation.

A truncated Hadamard matrix (the one without all 1's column) using Symmetric Balanced Incomplete Block Design (SBIBD) [18] can be easily generated, since most of the available SBIBD's are cyclic by construction. For this purpose, all the blocks (which corresponds to all the elements of the set C, besides codewords with all 0's and all 1's) can be generated by taking $n-1$ cyclic shifts of a single generator codeword. Such generators, for different values of $n-1$ can be found in a straight forward manner, using the so called *difference set approach* [18]. Table 1 illustrates the generator codewords for various values of n-1. The set of scheduler nodes for Q_7 can be obtained by first constructing codewords for Q_7. The generator codeword for Q_7 is 0010111. The additional 6 codewords are generated by taking 6 left cyclic shifts of this generator.

The complete set C then consists of rows with all 0's and all 1's plus the following 7 codewords and their complements C = {0010111, 0101110, 1011100, 0111001, 1110010, 1100101, 1001011, 1101000, 1010001, 0100011, 1000110, 0001101, 0011010, 0110100}. For Q_8, the set C can be produced by restoring the truncated matrices from the above set, which is the same as shown earlier in Figure 1 where each row of the matrix represents the binary address of the 16 schedulers.

The set, consisting of codewords, given in Figure 1, can also be used to generate the set C for the Q_9 network, by appending an all 0's and all 1's column (say at extreme left position), of matrix M and M^C, respectively. Also, the same set can be used to generate the set C for Q_{10}, as described in the procedure of Case (b).

3.4. Distributed System Model

A distributed system can be modeled as a collection of processing nodes connected by a communication network. As mentioned above, we assume that a virtual network isomorphic to a Hypercube exists. The nodes contain general purpose resources such as memories, processors, databases etc., and provide an execution environment for the tasks entering into the network. The network is assumed to be homogeneous where all nodes are identical. The models of network, processing node and scheduler are briefly described as follows.

3.4.1. Network and Sphere Model

The network of a distributed system provides high speed communication channels for direct communication between any pair of nodes. The network traffic can be viewed at two levels. At the higher level, task migration takes place between different spheres and at the lower level, tasks are transported from the schedulers to the nodes of their respective spheres. Both kinds of task movements incur communication delay which is dependent on the task transfer rate of the network. Similarly, at the higher level, message passing takes places between schedulers for exchanging the information about the accumulative loads of spheres. At the second lev-

el, a node only need to send a message to its scheduler when it finishes a task.

3.4.2 Node Model

A node in is assumed to be an independent processing element with its own local memory and operating system. We assume that tasks entering in the network are independent program modules. These independent task are capable of being serviced at any node (except for the transfer cost due if task is migrated to another node) irrespective of where they are submitted. Tasks scheduled at a node are entered in the execution queue which is served on FCFS principle.

3.4.3. The Scheduler

Tasks generated at the nodes of a sphere or migrated from other spheres are received at the scheduler. The Scheduler, which is a processing element along with some memory for keeping load information, is responsible for deciding where a task should be assigned. We assume that the set of schedulers, selected through the Hadamard code, are embedded on the actual system topology. In other words, the nodes that are designated as schedulers perform their regular task execution as well as they carry out the rule of the scheduler. Alternatively, we can assume that those nodes are augmented by special purpose processing elements that perform the scheduling.

3.4.4. Information storage and maintenance

As discussed above, a scheduler keeps two types of information. First, it maintains the accumulative load of the sphere which is simply the total number of tasks being serviced in that sphere at any point of time. This load index is adjusted every time a task enters a sphere or finishes execution. The other type of information is a linked list that points to the nodes of sphere according to their loads, and a load entry for each node of the sphere. The list is maintained in a non decreasing order with the first element of the list pointing to the most lightly loaded node. When the scheduler assigns a task to the node pointed to by the first list entry, the load entry of that node is updated. The node is removed from the head of the list and re-inserted in the list according to its new value of load. This update operation is done by using some fast recursive algorithms e.g. Binary search. When a node finishes a task, it informs its scheduler which adjusts the position of the node in the linked list according to its new value of load. In the proposed semi-distributed structure, it is possible that a node is shared by more than one scheduler. For example, in the case of Q_8 network, a node is either shared by one scheduler or it is shared by four schedulers. In case a node is a shared, the scheduler that assigned the task informs the other schedulers so that the load entry of that node could remain consistent. A shared node has to inform all of its schedulers whenever it receives or finishes a task. Therefore, load the load entries of the shared nodes in different spheres remain consistent.

3.5. The scheduling Algorithm

Tasks are generated at all nodes of the network and the nodes send them to their respective schedulers (in case a node is a shared among schedulers, one scheduler is randomly chosen). In our proposed semi distributed scheme, we assume that the operating system routes a newly generated task to a scheduler.

One parameter associated with the scheduling algorithm is the load threshold which could be set as zero or one depending upon the communication rate of the channels. When a task arrives, the scheduler first checks the load entry pointed by the first element of the linked list which shows the load of the most lightly loaded node in the sphere. If the load entry of that node is less than or equal to the threshold, then the scheduling algorithm stops here, the linked list is updated and the tasks is sent to the most lightly loaded node. Suppose the load threshold is set to one. Then if an idle node is available in the sphere, it is obviously the best possible choice. Even if the most lightly loaded node already has one task in its local queue, there exist chances that by the time the task migration from the scheduler to that node takes place, that node can become idle. If the task is sent to another sphere, it may not find a better node in the new sphere after going through sphere to sphere delay. The scheduler considers sending a task to another sphere only if the load of the most lightly loaded node in its sphere is greater than the load threshold. In that case, the scheduler sends queries to other schedulers. The other schedulers respond by sending the the values of the cumulative loads of their respective spheres. One of the remote spheres with cumulative load less than that of local sphere is randomly selected. If there is no sphere with cumulative load less than that of local sphere, the task is sent to the most lightly loaded node in the local sphere, irrespective of its load. The reason for selecting a sphere randomly is to avoid the most lightly loaded sphere becoming a victim of task dumping from other spheres. Choice of the load threshold should be made according to the communication rate of the channels. For example it would be more beneficial to set load threshold more than 1 if the task transfer rate is very low.

4. Simulation Methodology

The proposed scheme was simulated to analyze its performance on a Q_8 network having 256 nodes. For this network, the set C of scheduler includes 16 nodes whose addresses are shown in Figure 1. The covering radius for this network is 2 [18]. The valencies v_0^j, v_1^j and v_2^j, for all j, have values 1, 8 and 28 respectively, corresponding to total volume of the sphere $|S_f(j)|$ equal to 37. Figure 2 shows one scheduler (with binary address '00000000') and other nodes in its sphere. The addresses of the nodes for v_0^j, v_1^j and $v_2^j, \forall j$, can be obtained by using the expressions given in Section 3.1. The number of spheres being shared by a node as a function of covering distance (f) is given in Table 2. It is obvious from Table 2 that the minimum f is 2, the covering radius of C. For a Q_8 network, a node lies in one sphere or four spheres.

The simulation was written in 'C' on an Encore Multimax containing 16 CPU's and 128 megabytes of memory. In each simulation run, 20,000 tasks were generated. The results were obtained under steady-state conditions and each data point was obtained by taking the average of a large number of simulation runs with different random number streams. All results were obtained with a 95 percent confidence interval, with the size of the interval varying up to 5 percent of the mean value. A number of parameters were varied, to observe the sensitivity of the proposed strategy. The parameters varied included the system load, channel communication rate, transfer limit and the load threshold. Task generation was modeled as a Poisson process with average arrival rate λ tasks/unit-time, identical for all the nodes. The execution and communication times of tasks were exponentially distributed with a mean of $1/\mu_s$ time-units/task and $1/\mu_c$ time-units/task, respectively.

5. Performance Evaluation and Comparison

The main performance measures, selected for analysis, were the mean response time of a task, standard deviation of node utilization, throughput, and the average number of messages generated. A number of simulation runs were carried out to study the effect of

various parameters on these performance metrics.

For comparison we selected two additional schemes. The first scheme was the no load balancing scheme that used the same workload and executed tasks on the nodes they originally arrived without making any task migration. The other scheme was a fully distributed scheme that was proposed in other studies [9], [27]. In this fully distributed scheme, every node executes the scheduling algorithm. Tasks can migrate between nodes depending upon the decision taken by the algorithm. Upon arrival of a task, a node gets the load status from its immediate neighbors (the load status is the number of tasks scheduled at that node) and if the local load is greater than the most lightly loaded neighbor, the task is sent to that neighbor; otherwise it is executed locally. The information exchange between the neighbors can be done periodically or instantaneously. The periodic information exchange incurs less overhead but presents a less accurate information as compared to the instantaneous exchange. For comparison we selected instantaneous information exchange version, that is, whenever a task arrives at a node from the external world or from a neighbor, the load information from the neighbor is obtained. If the task is to be transferred to another node, the neighbor with the lowest load is selected. A task is allowed to make many migrations until it finds a suitable node or the number of migrations made exceed a predefined transfer limit. Due to heavy task migrations, a node has to maintain a communication queue for each neighbor. The scheme has been studied through simulation and has shown to exhibit high performance with a wide range of parameters. It is also known as the *best choice scheme* [9]. The time to execute the scheduling algorithm is assumed to be zero, both for fully and semi-distributed schemes.

5.1. The Response Time Comparison

The curves of mean response times versus system load with all three strategies are shown in Figure 3. Both fully and semi distributed schemes yield a substantial improvement in response time over the no load balancing scheme at all loading conditions. It should be mentioned that the parameters we selected in simulation, were those that produced the best achievable performance for the fully distributed scheme. The task transfer limit, for instance, was chosen as 12 which provided the best results through simulation. The fully distributed schemes, including the one under discussion, are critically dependent on the channel speed of the network. To make fair comparisons, we considered a fairly fast network with a communication rate of 20 task/time-unit compared to service rate of 1 task/time-unit. The performance of the fully distributed strategy does not improve if the channel rate is further increased. The load threshold for the semi distributed strategy was set to zero.

The average response time of the proposed semi distributed scheme is shown to be superior to the fully distributed scheme, in Figure 3. It is to be noted that with utilization ratio below 0.6, the response time curve with semi-distributed scheme is rather smooth. This is due to the fact that under low loading conditions, a scheduler is usually able to find a node whose load index is less than or equal to the load threshold which is set to zero in this case. In other words, the scheduler always makes use of an idle node. At load levels slightly higher, the inter-sphere task migrations occur. At a very high load, the tasks migrate frequently between spheres. The load is balanced between heavily and lightly loaded regions of the network. This retains the improvement in response time even under heavy loading conditions.

Since there is no random task migration activity within a sphere, the communication links can be designed to be dedicated links operating in a circuit switched mode. The task transfer mechanism from scheduler to nodes can be deterministic and can be designed as one to many broadcast, in a pipelined fashion [12]. Although, for the sake of comparison with the fully distributed scheme, we considered the task transfer time, from the scheduler to other nodes, in the simulation model, we believe that if this is done in the pipelined fashion, the task transfer delay would be negligible and the performance of the semi distributed scheme would be further improved.

At the higher level, the migration of tasks incurs delays between the schedulers. In the proposed semi-distributed design, all schedulers are at equal distance from each other except for the anti-podal pair. The schedulers of the anti-podal pair are located at a distance equal to diameter of the network (These are the scheduler nodes whose Hadamard codes are complementary). Therefore, with a $|C| = 2n$, each scheduler is at equal distance from $2n-2$ schedulers (excluding itself and its complement). We also conjecture that if the task generation rate at nodes is not uniform, the semi distributed strategy would yield even better performance since the load can migrate between all parts of the network which is not possible with fully distributed strategies, within short time.

5.2. Load Distribution

In addition to providing a fast job turn around time, load balancing provides a better resource utilization by trying to keep all nodes equally busy. Standard deviation of total utilizations of nodes is a measure of goodness of a load balancing strategy giving an estimate of *smoothness* of load distribution. Figure 4 shows standard deviation curves for all three schemes with the parameters same as those for Figure 3. The curve for the no load balancing scheme presents the variations in utilization and actually indicates the imbalance in work load assigned to the system. Low standard deviation, for both load balancing schemes, indicates a more uniform distribution of load. However the semi-distributed scheme results in a better load balancing as indicated by low values of standard deviation. This is due to the scheduling algorithm executed by scheduler since the nodes with identical loads are handled on an equal priority basis.

If a node is assigned a task, its position in the linked list is adjusted and the next node in the list is considered for the next task. At high load, occasional spikes of high loading are smoothed out by sending load to other spheres. High variations in utilizations at low loading conditions are due to the fact that no inter-sphere task migration takes places and some spheres may be get more task than others.

5.3. Throughput

Figure 5 shows the enhancement in throughput gained with the semi-distributed scheme at medium to high system load. The throughputs at low loading conditions, for all three schemes, essentially remain the same. The lower value of throughput with the no load balancing scheme can be attributed to the increased queue lengths at individual nodes acting as bottlenecks for tasks.

The fully distributed scheme equalizes the queue lengths, thereby yielding a subtle improvement in the throughput but at the cost of task accumulations in the communication queues which can get considerably large at heavy loads. From Figure 5, we observe that the throughput for fully distributed scheme starts converging to the curve for the no load balancing scheme. Semi distributed scheme yields further improvement and does not suffer from this problem.

5.4. Sensitivity To Communication Delay

Fully distributed load balancing strategies suffer because of the communication delays. This is due to channel contentions caused by the high task transfer rates. High transmission delay not only slows task migration but also results in an increase in queuing delay in the communication queues. With slow communication, the state of the system can change by the time a task reaches its destination [1], and the task may have to be redirected to another node. This may result in thrashing of tasks [1][5].

The impact of communication delay, on the task response time depends upon the ratio of mean communication delay to mean service time. It is known that if this ratio is greater than one, then the response time with load balancing may even exceed the response time obtained with no load balancing [1], [19].

Figure 6(a) and Figure 6(b) shows the percentage improvement in average response times gained with fully and semi distributed schemes over the no load balancing scheme for varying task transfer rates, at low and high loads respectively. The load threshold was set to one for task transfer rates between 2 task/time-unit to 10 tasks/time-unit and to zero for the range between 12 tasks/time-unit to 20 tasks/time-unit. The transfer limit for the fully distributed scheme was also varied from 7 to 12. The results indicate that even at a task transfer rate of 2 tasks/time-unit (and mean communication to service time ratio equal to half), the response time improvement with semi distributed scheme is almost two times the improvement gained with the fully distributed scheme. At low load, the response time improvement yielded by semi distributed strategy is nearly independent of the transfer rate beyond 6 tasks/time-unit. This is because most of the time the scheduler is able to find a node with zero or one task in its sphere. This was confirmed because less than 5 percent of the total tested tasks made any migration out side the spheres, even with load threshold set to zero. Figure 6(a) also indicates that at low load, the response time with fully distributed strategy changes rapidly between transfer rates of 2 task/time-unit to 10 tasks/time-unit. This clearly indicates that fully distributed scheme is more susceptible to channel delays.

As depicted in Figure 6(b), at high loading conditions, the semi distributed strategy exhibits some dependency on the channel rate due to the delays incurred during inter-sphere communications. This dependency factor diminishes if the transfer rate is increased beyond 8 tasks/time-unit. In contrast, the response time obtained with the fully distributed strategy shows steady improvement up to a transfer rate of 16–20 tasks/time-units, after which there is negligible improvement. The gain in performance with semi distributed strategy over the fully distributed strategy keeps on increasing with transfer rate up to 20 tasks/time-unit.

5.5. The Message Overhead

One of the goals of the proposed strategy was to reduce the communication traffic due to the control overhead. The messages carrying control information can delay normal task migrations [22]. In the proposed strategy, the number of messages are small. A node belonging to one sphere has to send only one message to its scheduler when it finishes a task. If a node is shared among more than one spheres, then the scheduler assigning the task to that node has to send messages to the spheres that share that node. Similarly, upon finishing a task, the node has to inform all of its schedulers. At the sphere level, a scheduler only communicates with other schedulers only when it considers a task migration. As mentioned above, the number of schedulers is only of the order of $\log N$. Therefore, the number of messages is also small. On the other hand, in the fully distributed environment, a node has to send and receive messages from all of its neighbors, every time a task is to be scheduled. If a task makes many migrations, each time a node has to exchange those messages. Figure 7 shows the average number of messages per task (this was calculated by dividing the total number of messages by the total number of tasks in a simulation run).

At low load, the overhead for the fully distributed is smaller since the nodes do not send any messages due to their local loads being very low, and tasks get scheduled without requiring any transfer. However as the load increases, the fully distributed strategy starts inducing high overhead which becomes almost three times the overhead incurred by the semi distributed strategy. Although, the message delay was assumed to be zero, it is hoped that, if considered, the average delay due to these messages will be less for semi-distributed scheme as compared to the fully distributed scheme. This is because the fully distributed model generates more messages than the semi distributed scheme.

6. Conclusions

In this paper, we have proposed a semi distributed approach for task scheduling in large parallel and distributed systems and have presented the concept of uniform covering of spheres with centralized control points. We have presented an efficient scheme for partitioning Hypercube systems for task scheduling. The approach is applicable to distance regular topologies. We have evaluated the performance of the proposed scheme through simulation and, with a comparison with fully distributed scheme, have shown that the new approach yields a better performance in terms of response time, throughput and load distribution. Due to an efficient scheduling and information collection method, the new approach has exhibited a low overhead in terms of number of messages generated. The proposed scheme was found to be less susceptible to channel delay.

Acknowledgements

We gratefully acknowledge the constructive comments from David Andrews. We are also thankful to NPAC (North East Parallel Architecture Center) for allowing us to use Encore Multimax.

References

[1] Ishfaq Ahmad and Arif Ghafoor, "Performance Evaluation of Decentralized Load Balancing Methodologies for Distributed Systems," paper submitted for publication.

[2] Katherine M. Baumgartner, Ralph Kling and Benjamin Wah, " A Global Load Balancing Strategy for a Distributed System," in *Proc. of IEEE Conf. on Distributed Computing Systems*, Hong Kong, 1988 pp. 93–102.

[3] E. Bannai and T. Ito. *Algebraic Combinatorics and Association Schemes.* Benjamin-Cummings (1984).

[4] Shahid H. Bokhari, "Dual Processor Scheduling with Dynamic Reassignments," *IEEE Trans. on Software Eng.* vol. SE-5, pp. 341–349, July 1979.

[5] Raymong M. Bryant and Raphael A. Finkel, "A Stable Distributed Scheduling Algorithm," in *Proc. of 2nd Intl. Conf. on Distributed Computing Systems, April 1981, pp. 314–323.*

[6] Thomos L. Casavant and John G. Kuhl, "Analysis of Three Dynamic Load-Balancing Strategies with Varying Global Information Requirements," in *Proc. of 7-th Intl. Conf. on Distributed Computing Systems*, West Germany, April 1987, pp. 185–192.

[7] Yaun-Chien Chow and Walter H. Kohler,"Models for Dynamic Load balancing in Homogeneous Multiple Processor Systems," *IEEE Trans. on Computers*, vol. c-36, no. 6, pp. 667–679, May, 1982.

[8] DARPA, "Strategic Computing: New Generation Computing technology," *Defence Advance Research Project Agency*, Arlington, Va, Oct. 1983

[9] Derek L. Eager, Edward D. Lazowska and John Zahorjan,"Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. on Software Eng.* ,vol. SE-12, pp. 662-675, May 1986.

[10] Kemal Efe, "Heuristic Models of Task Assignment Scheduling in Distributed Systems," *IEEE Computer*, June 1982, pp. 50-56.

[11] Ahmed K. Ezzat, R. Daniel Bergerson and John L. Pokoski, "Task Allocation Heuristics for Distributed Computing Systems, "in *Proc. of Intl. Conf. on Distributed Computing Systems.* pp. 337-346.

[12] G. C. Fox, S. W. Otto nd A. J. G. Hey, " Matrix algorithms on a hypercube I: Matrix multiplication," *Parallel Computing,*, 4 (1987), pp. 17-31.

[13] Arif Ghafoor and P. Bruce Berra, " An Efficient Communication Structure for Distributed Commit Protocols," IEEE *Jour. on Selected Areas of Communications*, vol. 7, no. 3, pp. 375-389, April. 1989.

[14] Arif Ghafoor, S.A Sheikh, and P. Sole, "Distance-Transitive Graphs for Fault-Tolerant Multiprocessor Systems", *Proc. of Intl. Conf. on Parallel Processing*, August 1989, Illinois, pp. I. 176- I. 180.

[15] Anna Ha'c and Xiaowei Jin, "Dynamic Load Balancing in Distributed System Using a Decentralized Algorithm," in *Proc. of 7-th Intl. Conf. on Distributed Computing Systems, West Germany, April 1987, pp. 170-178.*

[16] Frank C. H. Lin and Robert M. Keller, "Gradient Model: A demand Driven Load Balancing Scheme," in *Proc. of 6-th Intl Conf on Distributed Computing Systems*, Aug 1986, pp. 329-336.

[17] A. M. McLoughlin, "The Complexity of Computing the Covering Radius of a Code," *IEEE Trans. on Inform.* Theory, col IT-30, pp. 800-804, Nov., 1984.

[18] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes, vols. I and II*, New York: North Holland, 1977.

[19] R. Mirchandancy, D. Towsly and J. A. Stankovic, "Analysis of Effect of Delays on Load Sharing," *IEEE Trans. on Computers*, vol. 38. no. 11, pp. 1513-1525, Nov. 1989.

[20] Lionel M. Ni and Kai Hwang, "Optimal Load Balancing in a Multiple Processor System with Many Job Classes," *IEEE Trans. on Software Eng.*, vol. SE-11 pp. 491-496, May 1985.

[21] Krithi Ramamritham, John A. Stankovic and Wei Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. on Computers*, vol. 38, no. 8. pp. 1110-1123, Aug. 1988.

[22] Kang G. Shin and Y. -C. Chang, "Load Sharing in Distributed Real-Time Systems with State-Change Broadcasts," *IEEE Trans. on Computers*, vol. 38, no. 8. pp. 1124-1142, Aug. 1988.

[23] John A. Stankovic, "Simulation of Three Adaptive Decentralized Controlled Job Scheduling Algorithms, "*Computer Networks*, June 1984, pp. 199-217.

[24] L. J. Stochmeyer and V. V. Vazirani, "NP-Completeness of some Generalization of the Maximum Matching problems," *Information Proc. Letters*, vol. 15, pp 14-19, 1982.

[25] A. M. Van Tilborg and L. D. Wittie, "Wave Scheduling - Decentralized Scheduling of Task Forces in Multicomputers," *IEEE Trans. on Computers*, vol. C-33, no. 9, pp. 835-844, Sept. 1984.

[26] Benjamin W. Wah and Jei-Young Juang, "An Efficient Protocol for Load Balancing in CSMA/CD Networks," in proc. *8-th Conf. on Loacl Networks*, Oct. 1983, pp. 55-61.

[27] Songnian Zhou, "Performance Studies of Dynamic Load Balancing in Distributed Systems," Ph.D. Dissertation, Univ. of California, Berkeley, Sept. 1987.

$$M = \begin{matrix} 0&0&0&0&0&0&0&0 \\ 0&0&0&1&0&1&1&1 \\ 0&0&1&0&1&1&1&0 \\ 0&1&0&1&1&1&0&0 \\ 0&0&1&1&1&0&0&1 \\ 0&1&1&1&0&0&1&0 \\ 0&1&1&0&0&1&0&1 \\ 0&1&0&0&1&0&1&1 \end{matrix}$$

$$M^C = \begin{matrix} 1&1&1&1&1&1&1&1 \\ 1&1&1&0&1&0&0&0 \\ 1&1&0&1&0&0&0&1 \\ 1&0&1&0&0&0&1&1 \\ 1&1&0&0&0&1&1&0 \\ 1&0&0&0&1&1&0&1 \\ 1&0&0&1&1&0&1&0 \\ 1&0&1&1&0&1&0&0 \end{matrix}$$

Figure 1. An 8×8 Hadarmard Matrix in 0-1 notation and its complement

Table I.
Generator Codes for different lengths

Length = n -1	Generator Codewords
7	0 0 1 0 1 1 1
11	1 0 1 1 1 0 0 0 1 0 1
15	1 1 1 1 0 1 0 1 1 0 0 1 0 0 0
19	1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0 1

Table II.
Distribution of nodes shared by different spheres as a function of the covering radius

Distance of a node from C	f = 2	3	4	5	6	7	8
0	0	0	14	14	14	14	15
1	1	8	8	15	15	16	16
2	4	4	12	12	16	16	16

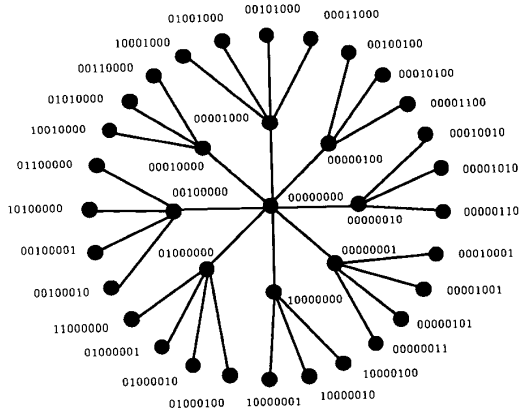

Figure 2. The Scheduler 00000000 and it corresponding sphere in Q_8 Network

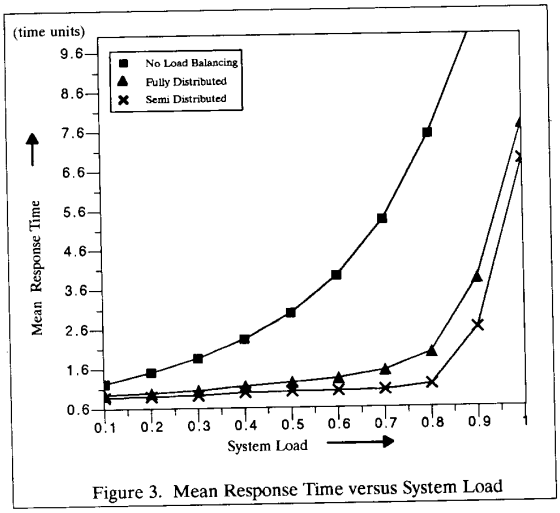

Figure 3. Mean Response Time versus System Load

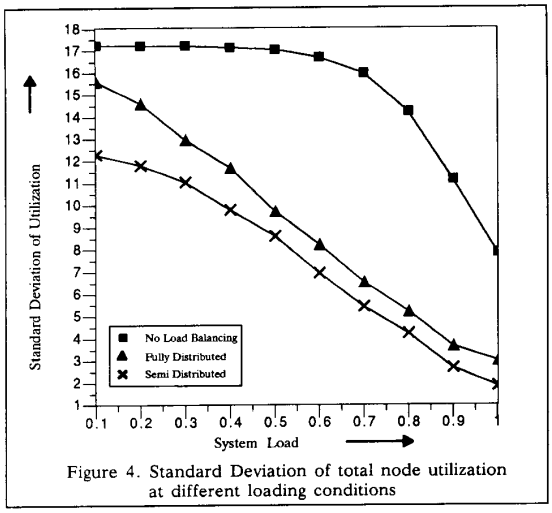

Figure 4. Standard Deviation of total node utilization at different loading conditions

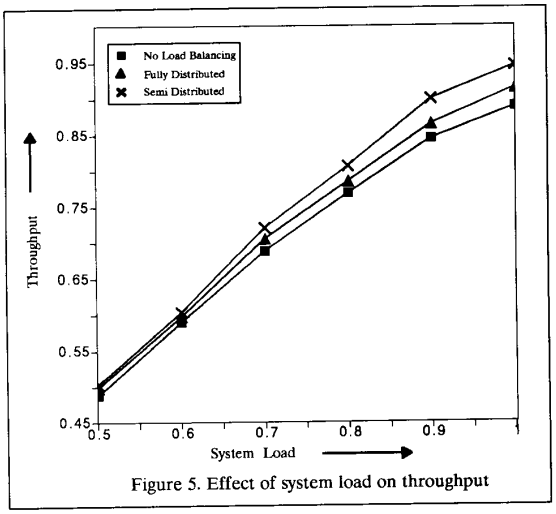

Figure 5. Effect of system load on throughput

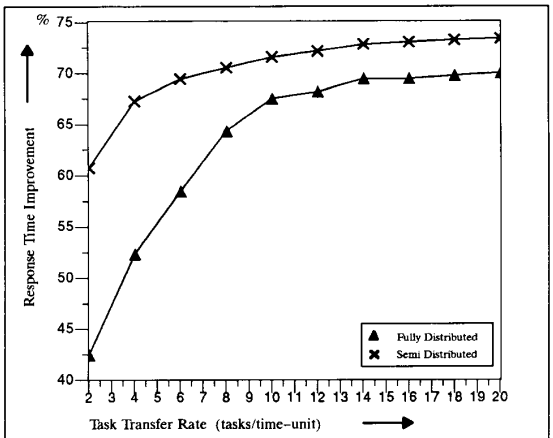

Figure 6 (a). Percentage improvement over no load balancing versus channel communication rate at low load (System Load = 0.6)

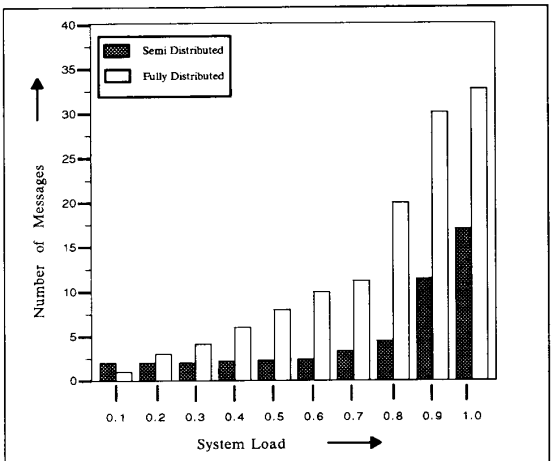

Figure 7. Average number of messages per task versus system load

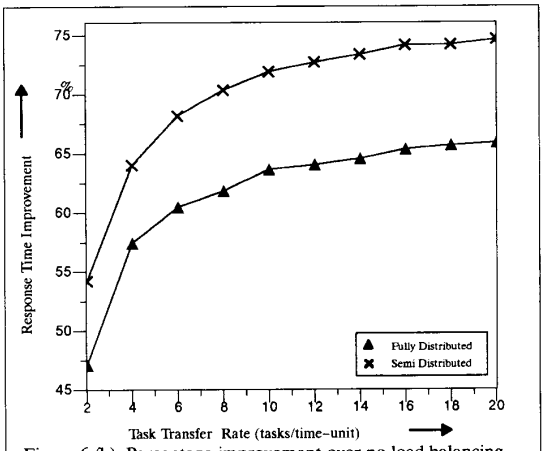

Figure 6 (b). Percentage improvement over no load balancing versus channel communication rate at high load (System Load = 0.9)