A Semi Distributed Load Balancing Scheme for Large Multicomputer Systems

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

Abstract

In this paper, we propose a semi distributed approach, for load balancing in large parallel and distributed systems. The proposed scheme is a two level hierarchical scheme which partitions the interconnection structure of a multiprocessor system into independent symmetric regions. We consider interconnection structures belonging to the classical infinite families of distance transitive graphs. The proposed scheme uses the partitioning property of these graphs by employing Hadamard matrices. The performance of systems consisting of these interconnection structures with varying sizes is evaluated through simulation and compared with a fully distributed scheme. Simulation results indicate that the performance of the proposed scheme improves with the increase in system size.

1. Introduction

When designing a distributed system, the problem of load balancing on the computing nodes of the system becomes an important issue. The problem becomes more challenging with a system consisting of hundred or thousands of nodes due to the overhead resulting from collection of state information, communication delays, saturation effect, high probability of node failures etc. Dynamic load balancing techniques have gained wide attention, in recent years, for providing improved performance in comparison with static load balancing techniques [6] since the unpredictable fluctuations in the load patterns across the system need to be balanced dynamically. Several dynamic load balancing techniques [2], [3], [4], [5], [8], [11], [16] have been proposed in the literature, but most of these techniques employ centralized [13] or fully distributed models [4], [5], [14], [15] which do not prove efficient for very large systems. It is, therefore, expected that there exists a trade–off between centralized and fully distributed load balancing mechanisms. The aim of this paper is to present a new approach exploiting the advantages of both centralized and fully distributed models.

For designing and developing very large multiprocessor system, both parallel and distributed, we propose a new scheme for task scheduling and load balancing. Our approach, which is semi–distributed in nature, partitions a very large system into independent and symmetric regions with control points centered at each region or sphere. The center of each sphere is, generally, at equal distance from all other centers. Load balancing and exchange of state information is carried out by the set of nodes at control points. The work load submitted to the system, which is characterized by arrival of tasks, is optimally balanced within and among these sphere. The control points, called schedulers, execute efficient scheduling and information building algorithms which have low overheads. Through simulation study, we show that for large systems, the proposed strategy yields a better performance than an efficient fully distributed scheme. The proposed scheme also incurs less overhead resulting from message exchanges.

The proposed scheme is a two level load balancing scheme. At the first level, load is shared 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*. The design of such a scheme involves the following steps:
(1) Formulate a network partitioning strategy for creating symmetric spheres, (2) Identify the control nodes (schedulers) for controlling their individual spheres, (3) Design an algorithm for performing optimal task scheduling and load balancing within the sphere as well as between spheres and (4) Develop efficient means for collecting state information at inter sphere and intra sphere level that should result in less message traffic.

For designing and developing large systems which can meet these design objectives, we propose an approach using a combinatorial structure. This approach is believed to be the first attempt towards developing such a methodology. We consider a class of graphs known as *Distance–Transitive (DT) Graphs.* The reason for analyzing DT graphs is not only they possess many useful properties, but most of the previously proposed topologies such as hypercube and bisectional graphs, are indeed distance–transitive. We show that distance–transitivity is a highly desirable property since these graphs are shown to be node–symmetric which helps in designing parallel and distributed systems with semi–distributed control. These DT networks are shown to possess a remarkable partitioning property based on a combinatorial structure known as Hadamard matrix. In this paper, we focus on a class of DT graphs, belonging to algebraic structures known as the Hamming [1],[9] Schemes.

We first provide a few definitions from graph theory. An interconnection network is represented by an undirected graph, $\Lambda = < U, E >$ where U represents the set of nodes and E represents the set of edges joining the nodes. Each node represents a combination of a processing unit and a message switching element through which the processor is connected to the network. The degree of each node represents the number of edges incident on it. It is assumed to be constant and is denoted as Δ. A *path* is a sequence of connected nodes. The length of the shortest path between nodes i and j is called the *graphical distance* and is represented as L_{ij}. Given $U = N$, let $k = Max[L_{ij} | \forall i, j, 0 \leq i, j, \leq N-1]$ where k is called the *diameter* of the network.

Definition: Let $G(\Delta)$ be the automorphism group of Δ. Δ is said to be *distance-transitive*, if for each quartet of vertices $u, v, x, y, \in \Delta$ such that $L_{u,v} = L_{x,y}$, there is some automorphism g in $G(\Delta)$ satisfying $g(u) = x$ and $g(v) = y$.

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. If we multiply, all the entries of M with -1, we obtain the *complementary Hadamard matrix*, denoted as M^C. If we replace 1 by 0, and -1 by 1, the matrix is said to be in *0–1 notation.* Hadamard matrices exist for order up to 268 and only if the order is a multiple of 4. Figure 1 shows an untruncated 7×7 Hadamard matrix, using 0–1 notation. The truncated matrix is obtained by discarding the first row and first column.

2. Interconnection Structures based on Hamming Schemes

These scheme describe algebraic structure of a well-known multiprocessor architecture, namely the Hypercube and two other networks, explained below.

The binary n–cube Network Q_n

Q_n, a binary n–cube network (Hypercube), which consists of 2^n nodes, is represented as a binary vector where two nodes with binary codewords x and y are connected if the Hamming distance $H_{xy} = 1$. For these graphs, $L_{xy} = H_{xy}$. For Q_n, $k = n$.

The Bisectional Network B_n

A Bisectional Network is analogous to a Hypercube and possesses some of the similar algebraic characteristic. A node x in a Bisectional network with constant degree n is connected to a neighbor y if $H_{xy} = n - 1$ [9]. We will denote a Bisectional network as B_n. A B_n network has 2^{n-1} nodes which is equal to the number of nodes in a Q_{n-1} network. For B_n, $k = (n - 1)/2$ and $L_{xy} = Min(H_{xy}, H^+_{xy})$ where $H^+_{xy} = n - H_{xy}$.

The binary odd network O_k

The odd graph O_k has for vertex set the binary words of length $2k - 1$ and Hamming weight $k - 1$ where two vertices are connected if and only if the Hamming distance between them is $2k - 2$. The O_k graphs are selected due to their higher density than various other interconnection networks: they have degree k, diameter $k - 1$ and $\binom{2k-1}{k-1}$ nodes.

3. The Partitioning Scheme

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: Let v^j_i be the number of nodes which are at a graphical distance i from a node j. This number is a constant for $\forall j \in U$ and is called the i–th valency.

Let C be the desired set of scheduling nodes. There are 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^j_i, the total size of the sphere is given as $|S_i(j)| = \sum_{k=0}^{i} v^j_k$

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.

The reason for selecting Hadamard code is firstly because this is a code with rate approaching zero, asymptotically [12]. This results in the size of a Hadamard code being considerably smaller than the size of a Hamming code, even for large values of n. Roughly the size of Hadamard code is proportional to the *logarithm* of the size of the network. The second reason for selecting Hadamard Code is that the range of values of n for which a Hadamard code exists, is larger than the range of n for which a Hamming code exists.

Thirdly, if n is an even power of two the covering radius of C is known [1]. Finally, Hadamard code can easily be extended. 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 the following ways.

Q_n with n mod 4 = 1. First we 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.

Q_n with $n \bmod 4 = 2$. This case is treated the same way as the above case, except that the set C is obtained from untruncated Hadamard matrices (in 0-1 notation) of size n-2. We, then 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.

Q_n with $n \bmod 4 = 3$. The set C, for this case 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) can also be generated by using Symmetric Balanced Incomplete Block Design (SBIBD) [9]. 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 by using the *difference set approach* [12]. Table I illustrates the generator codewords for various values of n-1. The set of C nodes for Q_7 can be obtained by first constructing codewords for Q_7. The generator codeword for Q_7 is 0010111. Hadamard matrix M is formed by taking these 7 codewords plus a row with all 0's. The complete set C, for Q_7, therefore consists of matrix M and its complement M^C. The set C for other Q_n's are generated by the methods described above. Similarly the generator codeword for O_6 is 10111000101. The additional 10 codewords generated by taking 10 left cyclic shifts of this generator, constitute the set C for O_6. The set C for Bisectional networks can be generated using the Symmetric Balanced Incomplete Block Design techniques except that we do not take the M^C matrix as the addresses of schedulers. Therefore in a B_n network the set C is half the size of that for Q_n.

4. Semi Distributed System Model

Figure 2 shows a logical view of the semi distributed model with the interconnection between spheres, and with each sphere consisting of a number of nodes. The system consists N identical processing nodes connected by a communication network. The work load entering in the network consists of independent program modules or tasks. 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 the proposed semi distributed scheme, we assume that the operating system routes a newly generated task to a scheduler. Therefore tasks are assumed to originate at schedulers.

Task migration takes place from schedulers to nodes and among schedulers. In addition, message passing takes places between schedulers for exchanging the information about the accumulative loads of spheres.

4.1. State Information exchange

The state information maintained by a schedulers is the accumulative load of its sphere which in turn is the total number of tasks being serviced in that sphere at that of time. This load index is adjusted every time a task enters a sphere or finishes execution. In addition, a linked list, maintained in a non decreasing order, points to the nodes of sphere according to their loads. The first element of the list points to the most lightly loaded node. The list is adjusted whenever a task is scheduled at a node or a task finishes its execution. It is possible that a node is shared by more than one scheduler. In that case, the scheduler that assigned the task to shared node informs the other schedulers to update their linked lists and load entries. Similarly, a node has to inform all of its schedulers whenever it finishes a task.

4.2. The Load Balancing Algorithm

As mentioned earlier, at the second level, load balancing is done optimally within the spheres by schedulers. A task is always scheduled at a node with the lowest load. At the higher level, load balancing is achieved by migrating tasks between spheres so that the accumulative load between spheres is equalized. Whenever a scheduler receives a task from the outside world or from another scheduler, it executes the scheduling algorithm. Associated with the scheduling algorithm are two parameters, namely threshold-1 and threshold-2. Threshold-1 is the load of the most lightly loaded node within the sphere. Threshold-2 is the difference between the accumulative load of the local sphere and the accumulative load of the remote sphere. The load balancing algorithm executed by the schedulers is described below.

Step 1. Check the load entry pointed by the first element of the linked list (the load of the most lightly loaded node in the sphere).
Step 2. If the load entry of that node is less than or equal to the threshold-1, then go to step 3. Otherwise go to step 6.
Step 3. Schedule the task at the most lightly loaded node pointed by the linked list, within the local sphere. Go to step 4.
Step 4. Updated the linked list.
Step 5. Update the accumulative load of the sphere. Stop.
Step 6. Check the accumulative load of other spheres.
Step 7. If the difference between the accumulative load of the local sphere and the accumulative load of the most lightly loaded remote sphere is less than threshold-2, send the task to that remote sphere. Otherwise go to step 3. Stop.

Threshold-1 determines whether the task should be scheduled in the local sphere or the scheduler should consider remote spheres for transferring task. Suppose the load threshold is set to one. Then if an idle node is available in the sphere, then that node is obviously the best possible choice. Even if the most lightly loaded node already contains one task in its local queue, the probability of that node becoming idle, during the time task migrates from the scheduler to that node, is high. 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. Threshold-2 determines if there is enough difference between the accumulative load of the local sphere and that of the remote sphere. One of the remote spheres, which meet the threshold criteria, is randomly selected. 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 thresholds should be made according to the system load and the task transfer rate. Values for threshold-1 were varied between 1 and 2 and for threshold-2 was varied from 1 to 4.

5. Simulation Results

For the performance analysis of the proposed scheme, we have simulated Q_7 ,Q_8 , Q_9 , Q_9 , B_7 , B_9 andO_6 networks. The topological characteristics and semi distributed structure for these networks are given in Table II which shows the number of nodes N, the degree d of each node, the distance between schedulers δ , cardinality of the set C, the covering radius r, valencies v_i^j and the size of sphere $S_i(j)$. For comparison we have selected the no load balancing scheme and a fully decentralized scheme. The fully distributed was proposed in an other study [5]. In this scheme the control is fully decentralized and every node executes the scheduling algorithm. Tasks can migrate between nodes depending upon the decision taken by the algorithm at each individual node. When a task arrives at a node, that node gets the load status from its immediate neighbors. The load status of a node is the number of tasks scheduled at that node. If the local load is less than the load of the most lightly loaded neighbor, the task is executed locally. Otherwise the task is considered for migration. In that case, 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.

For simulation, task load was modeled as a Poisson process with average arrival rate λ tasks/unit-time, and was identical for all the nodes. The execution and communication times of tasks were also assumed to be exponentially distributed with a mean of $1/\mu_s$ time-units/task and $1/\mu_c$ time-units/task, respectively. Extensive simulation was carried out to analyze and compare the performance of the above mentioned networks. The performance measures selected were mean response time of a task and average number of control messages generated per task. In each simulation run, 20,000 to 100,000 tasks were generated depending upon the size of the network. The steady-state results are presented with 95 percent confidence interval, with the size of the interval varying up to 5 percent of the mean value.

5.1. The Response Time Performance

The mean response time of a task is the major performance criteria. To analyze the impact of system load, which is described as the ratio of task arrival rate to task service rate, on mean response time, the system load was varied from 0.1 to 1.0. Figure 3. shows the curves of mean response times versus system load with all three strategies for Q_{10} network. Both fully and semi distributed schemes yield a significant improvement in response time over the no load balancing scheme at all loading conditions. The parameters selected in simulation, were those that produced the best achievable performance for the both schemes. The task transfer limit, for instance, was chosen as 12 which provided the best results through simulation. Task transfer rate was selected as 20 task/time-unit compared to service rate of 1 task/time-unit. Threshold-1 and threshold-2 were set for 1 and 3 for the semi distributed scheme.

The average response time of the proposed semi distributed scheme is superior to the fully distributed scheme as shown in Figure 3. It is to be noted that for utilization ratio below 0.8, the response time curve with semi-distributed scheme is rather smooth and is almost equal to 1.0, which is the average service time of a task. It implies that with the semi distributed scheme the load balancing is optimal and tasks get serviced, virtually, without any queuing delays. 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 one in this case. For Q_{10} , the sphere size is 176 and the probability of finding an idle node in a sphere is very high. In other words, the scheduler always makes use of an idle node in its own sphere. The only delay incurred before a task gets executed is the communication delay resulting from task transfer from a scheduler to a node. At slightly higher load levels, the inter-sphere task migrations occur. At a very high load, the tasks migrations among spheres take place more frequently and the load is balanced between heavily and lightly loaded regions of the network.

When load balancing among spheres takes place, extra delays are incurred due to migrations of tasks between the schedulers. In the proposed semi-distributed design, all schedulers are at equal distance from each other except for hypercube where a scheduler's anti-podal node (the scheduler whose binary address is the complement of this scheduler) is also present. The schedulers of the anti-podal pair are located at a distance equal to diameter of the network. Therefore, with a $|C| = 2n$, each scheduler is at equal distance from $2n$ -2 schedulers (all except itself and its complement). This results in an inter sphere load migration in a symmetric manner. For B_9 and O_6 , all schedulers are at equal distance from each other. As shown in Figure 4 and Figure 5, the response times curves obtained for the three schemes for B_9 and O_6 , exhibit similar patterns, with semi distributed scheme outperforming the fully distributed scheme.

5.2. The Performance Comparison of Different Networks

In order to analyze the impact of network size, the number of schedulers and the sphere size on the performance of semi distributed and fully distributed schemes, we compare the average response times yielded by B_7 , Q_7 , Q_8 , B_9 , O_6 , Q_9 and Q_{10}, networks. Table III shows the percentage improvement in average response time yielded by semi and fully distributed schemes over the no load balancing scheme, for these networks. Three different loading conditions, low, medium and high were chosen corresponding to system loads of 0.6, 0.8 and 0.9 respectively. The task transfer rate for all these results was chosen as 20 tasks/unit-time and transfer limit for fully distributed scheme was varied between 8 and 12. The most interesting observation is the increase in performance of semi distributed scheme with the increase in system size. On the other hand the performance of fully distributed scheme drops as as the system size increases. These observations are valid for all loading conditions. We, therefore reach to the conclusion that the proposed semi distributed scheme is more suitable for large systems. This is obvious from the difference in response time improvements of both schemes which is the maximum for Q_{10}. When comparing the performance Q_8 and B_9 which have identical number of nodes but different number of schedulers and sphere sizes, we note that B_9 outperforms Q_8 at medium load. This is due to larger sphere size which results in an increased probability of finding a suitable node within the local sphere. The performance of the semi distributed scheme was slightly better than the fully distributed scheme for Q_7 at medium load. However, the fully distributed scheme outperforms the semi distributed scheme for B_7 at high load. This is due to the high connectivity of the network which is more suitable for the fully distributed scheme for making task migrations.

5.3. The Cotrol Overhead Due to Message Exchange

The average number of messages generated per task were evaluated for all seven networks to compare the impact of size and other topological parameters on this overhead. A node belonging to one sphere has to send only one message to its scheduler when it finishes a task. The messages are generated in order to keep the load entries in different spheres consistent or to exchange state information among schedulers. If a node is shared among more than one spheres, then the scheduler assigning the task to that node needs to send messages to the spheres sharing that node. Similarly, upon finishing a task, a node has to inform all of its schedulers. At the sphere level, a scheduler communicates with other schedulers only when it considers a task migration. The number of messages for exchanging state information among schedulers is small since the number of schedulers is only of the order of $\log N$, whereas with fully distributed environment, the scheduling decision for every task is critically dependent on the precise information about neighboring nodes and a node has to get such information for each of its neighbors, every time a task is to be scheduled. Figures 6(a), Figure 6 (b) and Figure 6(c) show the average number of messages per task (this was calculated by dividing the total number of messages by the total number of tasks) for all seven networks at low, medium and high loading conditions, respectively.

At low load, the message overhead for the semi distributed scheme is much lower than that of fully distributed scheme except for O_6 network where the degree of each node is small, resulting in less number of messages for the fully distributed scheme. The overhead for the semi distributed scheme is higher due to node sharing and information exchange between schedulers. As the load increases from low to medium, the fully distributed strategy starts inducing high overhead which almost doubles the overhead incurred by the semi distributed strategy. This conclusion is valid for all networks except Q_7 where the sphere size if very small and with the semi distributed scheme, schedulers need to communicate with each other more frequently. The impact of sphere size is more apparent when the message overhead for Q_8 and B_9 is compared. At all loading conditions, the overhead for the semi distributed scheme with Q_8 is higher than incurred with B_9 . This is because the number of schedulers in B_9 is half the number of schedulers for Q_8 . This overhead is also less due to less sharing of nodes in different spheres (in B_9 a node is shared in either 1 or 3 spheres whereas in Q_8 , a node is shared in 1 or 4 spheres). At high load, the overhead with the semi distributed is higher but still less than that of fully distributed scheme.

6. Conclusions

In this paper, we have proposed an approach for load balancing in large parallel and distributed systems and have presented the concept of semi distributed control. The study was centered around a class of interconnection structures which are distance transitive. The use of Hadamard matrix results in an efficient scheme for partitioning these systems for load balancing. We have evaluated the performance of the proposed scheme through extensive simulation with three example graphs of various sizes. By comparing with a fully distributed scheme, we have shown that the the proposed scheme yields a better performance in terms of average response time and the average number of control messages generated. These results are validated for systems of various sizes.

References

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

[2] 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.*

[3] 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.

[4] Shyamal Chowdhury, "The Greedy Load Sharing Algorithm" *Journal of Parallel and Distributed Computing,* no. 9, May 1990, pp. 93–99.

[5] 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.

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

[7] 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.

[8] G. C. Fox, A. Kolawa and R. Williams, "The Implementation of Dynamic Load Balancer, "in *Proc. of SIAM Hypercube Multiprocessors Conf.*, 1987, pp. 114-121.

[9] Arif Ghafoor, Theodore Bashkow and Imran Ghafoor, "Bisectional Fault-Tolerant Communication Architecture for Supercomputer Systems," *IEEE Trans. on Computers*, vol. 38, no. 10. pp. 1425-1446, October 1989.

[10] 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.*

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

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

[13] 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.

[14] 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.

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

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

$$\begin{array}{ccccccc}
0 & 0 & 0 & 1 & 0 & 1 & 1 \\
0 & 0 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 0 & 1 & 1 & 0 & 0 \\
1 & 0 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1
\end{array}$$

Figure 1. A truncated Hadamard Matrix M of order 7.

Table 1. 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
The semi distributed structure of various interconnection networks

Network	N	d	δ	$\lvert C\rvert$	r	v^j_1	v^j_2	v^j_3	$\lvert S_i(j)\rvert$
B_7	64	7	3	8	1	7	-	-	9
Q_7	128	7	4	14	1	8	-	-	9
Q_8	256	8	4	16	2	8	28	-	37
B_9	256	9	4	8	2	9	36	-	46
O_6	462	6	5	11	3	6	30	75	112
Q_9	512	9	5	16	2	9	36	-	46
Q_{10}	1024	10	5	16	3	10	45	120	176

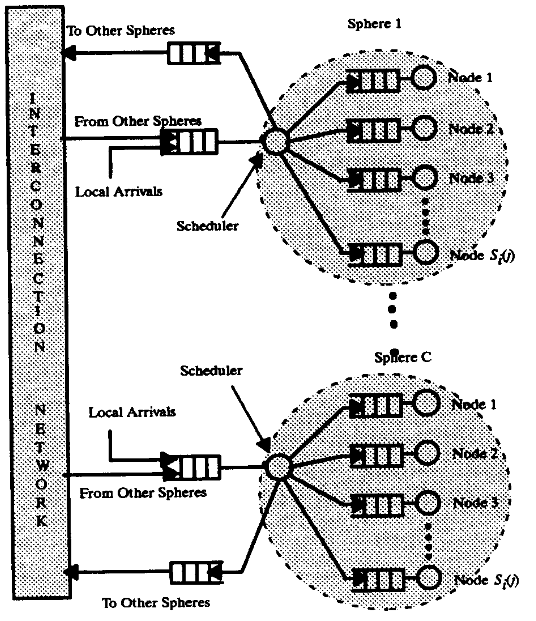


Figure 2. A logical View of Spheres and Semi Distributed Load Balancing Scheme.

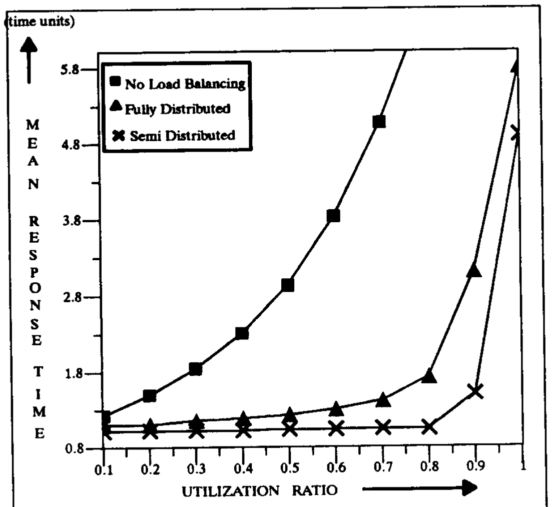


Figure 3. Mean response time versus system load with three schemes for the Q_{10} network.

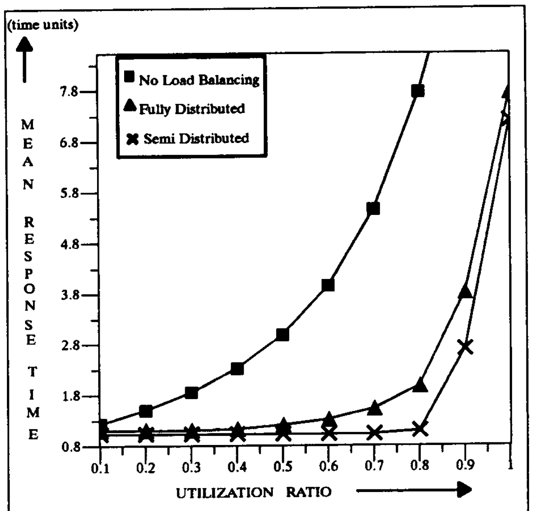


Figure 4. Mean response time versus system load with three schemes for the B_9 network.

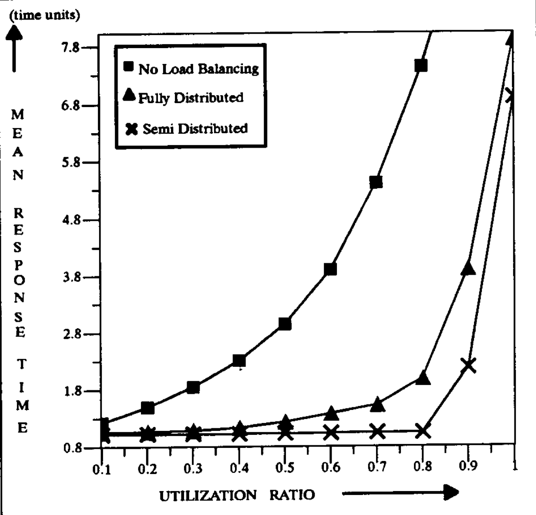
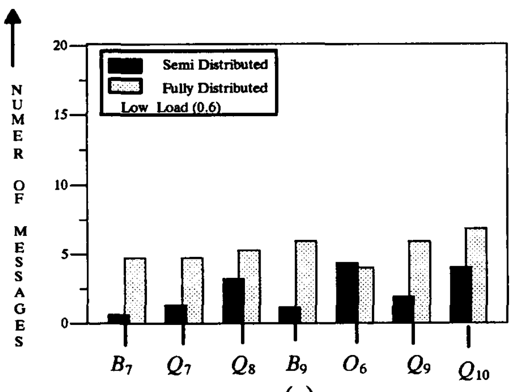


Figure 5. Mean response time versus system load with three schemes for the O_6 network.

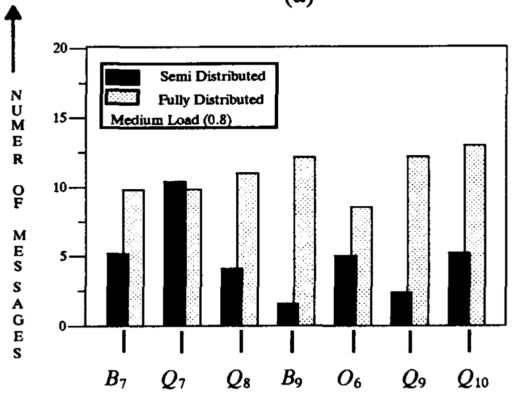
TABLE III

The percentage improvement (decrease) over no load balancing with semi and fully distributed schemes for various networks

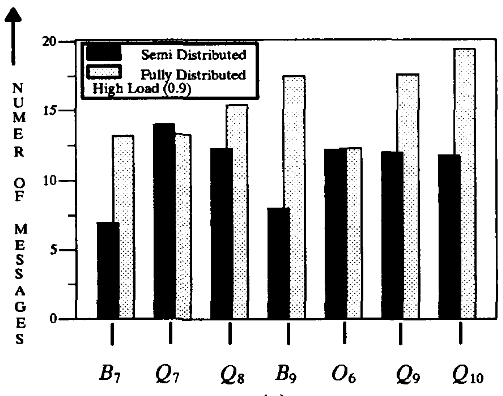
Load / Network	Low (0.6)		Medium (0.8)		High (0.9)	
	Semi Dist.	Fully Dist.	Semi Dist.	Fully Dist.	Semi Dist.	Fully Dist.
B_7	73.64	71.22	77.80	76.22	72.58	73.22
Q_7	73.50	71.04	77.71	77.80	72.52	71.94
Q_8	73.62	69.08	80.69	73.53	74.42	69.22
B_9	73.55	71.05	82.74	75.81	75.52	70.47
O_6	73.25	70.50	84.54	74.40	75.61	61.57
Q_9	73.19	71.67	84.93	74.52	79.16	68.10
Q_{10}	72.77	71.49	85.36	72.40	82.04	62.57



(a)



(b)



(c)

Figure 6. Average number of messages per task for various interconnection at works at low, medium and high loads.