7-1991

Fault-Tolerant Load Management for Real-Time Distributed Computer Systems

Arif Ghafoor

Ishfaq Ahmad

Syracuse University, School of Computer and Information Science

SU-CIS-91-24

Fault-Tolerant Load Management for Real-Time Distributed Computer Systems

Arif Ghafoor and Ishfaq Ahmad

July 1991

School of Computer and Information Science
Syracuse University
Suite 4-116, Center for Science and Technology
Syracuse, New York 13244-4100

Abstract

This paper presents a fault–tolerant scheme applicable to any decentralized load balancing algorithms used in soft real–time distributed systems. Using the theory of distance–transitive graphs for representing topologies of these systems, the proposed strategy partitions these systems into independent symmetric regions (spheres) centered at some control points. These central points, called fault–control points, provide a two–level task redundancy and efficiently re–distribute the load of failed nodes within their spheres. Using the algebraic characteristics of these topologies, it is shown that the identification of spheres and fault–control points is, in general, is an NP–complete problem. An efficient solution for this problem is presented by making an exclusive use of a combinatorial structure known as the Hadamard matrix.

Assuming a realistic failure–repair system environment, the performance of the proposed strategy has been evaluated and compared with no fault environment, through an extensive and detailed simulation. For our fault–tolerant strategy, we propose two measures of goodness, namely, the percentage of re–scheduled tasks which meet their deadlines and the overhead incurred for fault management. It is shown that using the proposed strategy, up to 80 % of the tasks can still meet their deadlines. The proposed strategy is general enough to be applicable to many networks, belonging to a number of families of distance transitive graphs. Through simulation, we have analyzed the sensitivity of this strategy to various system parameters and have shown that the performance degradation due to failures does not depend on these parameter. Also, the probability of a task being lost altogether due to multiple failures has been shown to be extremely low.

Key Words : Fault–tolerant Load Balancing, Multicomputer Systems, Network Partitioning, Distance–Transitive Graphs, Performance Evaluation, Task Scheduling.

1. Introduction

A major advantage of distributed computer systems is the availability of a large number of autonomous computers which can provide an enhanced processing speed, through coordination. A special class of distributed systems is multicomputers which consist of a large number computing nodes connected by a high bandwidth network [2]. If the resource allocation is carefully managed, a large–scale multicomputer can be much more cost effective and efficient as compared to expensive supercomputers. With the continuing advancements in microprocessors and high speed networks, one can envisage such a high performance system for a

wide range of applications [11]. An important use of multicomputers is to support real–time applications. For real–time systems, two important considerations need to be taken into account, namely; time and reliability.

Time is considered to the most crucial resource to manage in real–time systems since the occurrence of events must follow some timing constraints [32]. While for non–real–time systems, the main performance measures include average system response time, throughput and resource utilization, in real–time systems, each task has some pre–specified execution deadlines which must be met for safe system's operation. In other words, in real–time systems, the performance should be evaluated by successful execution of individual tasks within certain time period, rather than the average system behavior. Real–time systems are further classified as soft and hard [32]. In soft real–time systems, a task is considered lost if it does not meet its deadline [24] whereas in a hard real–time system, the failure to meet a deadline can be catastrophic [33].

The design of a large–scale multicomputer system for real time applications entails efficient and quick means of resource sharing such as load balancing on the computing nodes. Inefficient scheduling can lead to a load imbalance on various nodes which can significantly increase the waiting times of tasks scheduled at heavily loaded nodes which in turn hinders new tasks to meet their deadlines. An essential property of a dynamic scheduling strategy is to balance the load across the system by transferring load from heavily loaded nodes to lightly loaded nodes. Dynamic load balancing has been considered the inevitable solution for this problem because the time dependent fluctuations in the load patterns across the system need to be balanced dynamically [7], [10], [14], [16], [30], [38].

Considerable research has been done for task scheduling and load balancing or load sharing for both soft [23], [24], [31] and hard real–time systems [25], [33], [35]. Load balancing can be carried out with a centralized or a decentralized model. In a centralized model [10], a single node gathers the global information about the state of the system and assigns tasks to individual nodes. On the other hand, in a decentralized case, each node executes a scheduling algorithm by exchanging state information with other nodes [7], [10], [14], [30], [33].

The second consideration for a real–time system is the issue of reliability. Since, with the increase in the size of the system, the likelihood of failure of its components (processors and links) also increases, the problem of task scheduling becomes more complex. Moreover, in a large system, it is also very likely that computing resources become unavailable for a variety of other reasons such as some processors may have to be shut off or isolated from the rest of the system for maintenance purpose or due to failure of other attached I/O devices. In addition to load balancing, overhead incurred as a result of collection of state information, communi-

cation delays, etc., can cause sever performance degradation [15], [28], even for non–real–time systems. This problem is acute in real–time environment where not only task execution must be controlled to meet deadlines, but also the failure of nodes must be tolerated to maintain an acceptable performance of the system [8], [29]. Means must be provided to handle failures and unavailability of nodes so that the operation of the rest of the system remains uninterrupted and the performance is not adversely affected. Mechanisms must also be provided to redistributed the unfinished computational load from the unavailable nodes to the operational nodes. Moreover, there should be some way of redirecting new tasks arriving to the nodes which are failed and are unavailable.

The above two requirements entail a dynamic load re–distribution strategy. These strategies need to be designed carefully since the re–injection of computational load from the failed nodes to the rest of the operation system can make some nodes unstable. Therefore, simple load distribution strategies can not be applied for system under failure.

Designing load balancing strategies for real–time systems is an important issue and a few schemes have been proposed in the recent literature[8], [9], [35]. However, very little has been explored for large–scale systems, where the failure of component is also a critical issue. In this paper, we propose a fault–tolerant load redistribution strategy for large–scale multicomputers. The proposed scheme is general in the sense that it is applicable to any distributed load balancing scheme. Also the proposed strategy is based upon a more realistic assumption about the failure situations, than the one given in earlier work [8]. In previous studies failure have been assumed to be static in the sense that a given number of nodes are assumed to fail simultaneously without any subsequent recovery or repair. However, the static failure assumption does not hold for real life systems where failures are generally randomly occurring events. Furthermore, the failed components can be diagnosed and repaired off–line and can be re–integrated back into the system [29]. The second realistic assumption made in this paper is that the incoming tasks to the failed nodes cannot be simply ignored. Due to real–time constraints, some means are required to to handle tasks arriving at the failed nodes.

In summary, the proposed strategy has the following objectives.

(1) The main objective is to propose a fault–tolerant task scheduling strategy with decentralized control which has the flexibility to allow the system to continue to be operational in spite of failure of nodes. The failed nodes are assumed to be repaired subsequently.

(2) The proposed strategy is applicable to both real and non real–time systems. However, the emphasis in this paper is in real–time systems, with any decentralized scheduling scheme.

(3) To provide fault–tolerance for both real–time and non real–time environments, some kind of load replication scheme is required to keep redundant copies of tasks within the system

so that in case of a failure, the affected tasks can be re-scheduled. The system should be able o sustain a large number of failures. Therefore, a certain level of redundancy is required in case the nodes with redundant copies also fail. In this paper, we assume two-level redundancy, that is, two copies of every task are kept in the system.

(4) A system can consist of hundred or thousands of nodes which can fail or become unavailable due to some other reasons, from time to time. An efficient strategy is required to select a set of nodes which are responsible for the re-distribution of effected tasks.

(5) Re-scheduling of tasks should not cause instability in the system and tasks should be re-distributed to only those nodes which are lightly loaded. The communication delays and dumping of lightly loaded nodes with tasks from other nodes are generally two main factors that can cause "turbulence" in the system. This turbulence can cause the system to enter into the state of instability or "task thrashing" which can prevent new tasks at operational nodes to miss their deadlines.

(6) For a real-time system, re-scheduling of tasks should be quick and efficient so that most of the tasks, which would have been lost otherwise, can still meet their deadlines, without affecting the underlying normal decentralized load balancing algorithm.

(7) The total load entering into the system should not decrease, if some nodes in the system are out of order. The newly arriving tasks at those nodes should be redirected to the operational nodes. This re-direction of new load should also not cause any instability and performance degradation.

As mentioned above, one of the objectives is to smoothly re-balance the load of failed nodes with minimum impact on the normal load balancing. This can be achieved if the nodes re-scheduling the backup tasks, which we will refer to as Fault Control Points (FCPs), have some partial knowledge of the global load of the system. The selection of FCPs is a crucial factor to the fault-tolerant performance of the systems, especially in terms of re-scheduling and reducing chances of missing deadlines. In a centralized approach, a single FCP can manage redundant load for the whole system. However, the failure of the FCP itself can eliminate fault-tolerance capability, and therefore, this scheme is highly vulnerable to failure. On the other hand, if a fully distributed scheme is used where each node acts as an FCP, the global knowledge of the system load has to be acquired by each node which is a costly solution in terms of overhead and the generation of overhead traffic can seriously effect the system's performance [15]. If limited knowledge, such as only the load of immediate neighbors, is used, the scope of re-scheduling a task to a suitable destination becomes rather limited. Clearly, a semi-distributed scheme with a fewer number of FCPs, each having some partial knowledge of the global state of the system, would be an ideal choice.

It has been observed that, in a distributed system, there is always a high probability that some nodes are idle while some are overloaded [7]. As the system size increases, this probability also increases. The load re-distribution strategy should be able to exploit this phenomenon by redirecting the backup and newly arrived load to the best possible nodes. Accordingly, in this paper, we propose a semi-distributed load re-distribution strategy which is based on partition of the system's topology into multiple symmetric regions (spheres). Each sphere is a cluster of nodes and has a number of fault-tolerant control points (FCPs). The total number of FCPs is relatively small, resulting in a low overhead (in terms of communicating tasks among nodes and FCPs) to provide fault-tolerance. At the same time, the number of FCPs is large enough to effectively monitor the failures in the network and manage load re-distribution within their spheres. Load re-distribution is carried out within individual spheres where the scheduler of each sphere acts as a centralized controller for its own sphere. Each node, is assigned two types of FCPs, for storing two redundant copies of ever task present at the node. FCPs with the first-level redundancy will be termed as *primary FCPs* and the other as *secondary FCPs*. The complete description about the system model is presented in Section 4. We show that, in general, an optimal determination (we describe this determination in Section 3) of FCPs in a network is an NP-complete problem. However, for a class of interconnection structures, known as *distance transitive graphs (DT)* [20], we propose a remarkable network partitioning and FCPs identification scheme based on a combinatorial structure known as the *Hadamard matrix*.

Assuming a realistic failure-repair system environment, the performance of the proposed strategy has been evaluated and compared with no fault environment, through an extensive and detailed simulation. For our fault-tolerant strategy, we propose two measures of goodness, namely, the percentage of re-scheduled tasks which their deadlines and the overhead incurred for fault management. It is shown that using the proposed strategy, up to 80 % of the tasks can still meet their deadlines. The proposed strategy is general enough to be applicable to many networks, belonging to a number of families of distance transitive graphs. Through simulation, we have analyzed the sensitivity of this strategy to various system parameters and have shown that the performance degradation due to failures does not depend on these parameter. Also, the probability of a task being lost altogether due to multiple failures has been shown to be extremely low.

The proposed strategy is applicable to both large parallel and distributed systems. For example, a Hypercube topology can be extended beyond a parallel processing environment by assuming that the virtual communication network topology of a distributed system is isomorphic to the hypercube provided the number of nodes in the system is 2^n. For virtual topology, if the number of nodes in the system is not equal to 2^n, virtual nodes can be added to

complete the topology [19]. The same idea can also be extended to any distributed system having a number of nodes possessed by any *DT* graph discussed in this paper.

The remainder of the paper is organized as follows. Section 2 gives an algebraic characterization of distance transitive interconnection networks. In Section 3, we state the problem of determining the set of FCP and their assignments to nodes. The determination of spheres for load re-distribution by using Hadamard matrices is also discussed in the same section. Section 4 describes the characteristics and assumption about the system. It further discusses the task replication and re-scheduling strategies. Simulation results showing the evaluation of the proposed model are given in Section 5. Section 6 presents the concluding remarks.

2. Distance-Transitive Topologies for Large Distributed Systems

As mentioned earlier, the virtual communication topology of a parallel and distributed systems can be represented by a *DT* graph. In case, the number of nodes in the system is not equal to the nodes in a *DT* network, pseudo nodes can be added to complete the topology [19]. In this section, we describe some of the infinite families of *DT* graphs which can represent topologies of distributed systems. We also introduce the notion of *range of load redistribution* and present some definitions and terminology which are used to characterize these topologies and their property of *symmetry*. The reason for analyzing *DT* graphs that many of the existing and previously proposed interconnection networks, including the Hypercube topology, 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. We focus on a class of *DT* graphs which are governed by two algebraic structures known as the Hamming and the Johnson Association Schemes [5]. The graphs belonging to these schemes include the Hamming graph (the hypercube), the Sphere (Johnson) graph [5], and their derivatives. In order to define the *DT* topologies, we need the following definitions.

The network of a system can be represented 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 $U = \{ 0, 1, 2, ..., (N-1) \}$ be the set of N nodes in the system, including the pseudo nodes if required. The degree of each node, denoted by n, represents the number of edges incident on it. The degree is assumed to be constant. A *path* in Λ 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} .

Definition: Let a be a binary codeword. The *Hamming weight w(a)* of a is equal to the number of 1's in a.

Definition: *The Hamming Distance,* H_{xy}, between two binary codewords, $x = (x_1, x_2, \ldots x_n)$ and $y = (y_1, y_2, \ldots . y_n)$ of some length n, is defined as

$$H_{xy} = |\{i | x_i = y_i, 1 \leq i \leq n\}| \quad \text{where} \quad x, y \in \{0, 1\}.$$

In other words, Hamming distance, between two codewords, is the number of different bits in codewords.

Definition: Let $k = Max[L_{ij} | \forall i, j, 0 \leq i, j, \leq N-1]$. k is called the *diameter* of the network Λ.

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

Definition: A *distance–regular* graph is a weaker case of *distance–transitive* graph. A graph of diameter k is *distance–regular* if

$$\forall (i, j, l) \in [0 \ldots n]^3, \qquad \forall (x, y) \in U,$$

$$L_{xy} = l \Leftrightarrow \left| \{ z \in U, L_{xz} = i, L_{yz} = j \} \right| = p_{ij}^{l},$$

where $| y |$ represents the cardinality of some set y and p_{ij}^{l} are constants whose values are dependent on the characteristics of the graph.

Distance–regularity is an important property in terms of describing *range of load redistribution* which affects the number of messages generated for gathering state information for redistributing the load of failed nodes. This range is defined later in the section.

Definition: Let v_i^x be the number of nodes which are at a graphical distance i from a node x. This number is a constant for $\forall x \in U$ and is called the i–th valency. It is given as $v_i^x = p_{ii}^0$.

We can classify *DT* networks into two classes, namely, the antipodal networks and podal networks. These classes and some of the networks belonging to these classes are described below.

2.1 Antipodal Networks

In an antipodal network every node has an exactly one diametric node, that is node which is at distance k, the diameter. Such a pair is called *antipodal pair*. Many well known networks fall into this category, such as ring network with even number of nodes, the binary Hypercube etc. In this paper, we discuss two such networks which are briefly described below.

The Binary n–cube Network Q_n

The binary n-cube network (Hypercube), which we denote as Q_n, consists of 2^n nodes. Here each node is represented as a binary vector where two nodes with binary codewords x and y are connected if the Hamming distance $H_{xy} = 1$. Then for every node x in Q_n, the valency sequence is given as:

$v_i^x = \binom{n}{i}$ for $i = 0, 1, 2 \ldots n$

and $L_{xy} = H_{xy}$ and $k=n$. Q_n is a distance-regular graph.

The Binomial Network Γ_n [20]

This network has for its vertex set the binary codewords of length $2n-1$ and Hamming weights n and $n-1$. Two vertices are connected if and only if they are at a Hamming distance 1. Such a network has degree n, diameter $2n-1$, and $\binom{2n}{n}$ nodes.

We will denote such a network as Γ_n. It is known that for these networks. $L_{xy} = H_{xy}$ [20]. For every node x in this graph, the valency sequence is given as:

$v_i^x = \binom{n}{i}\binom{n-1}{i}$ for $i = 0, n-1, 1, n-2, 2, \ldots k, k, \ldots..0$, where $k=n-1$.

Note, that Γ_n is a sub-graph of Q_s, where $s=2n-1$. This graph belongs to the Johnson Association Scheme [5].

3.2 Podal Networks

For podal networks there do not exist any antipodal pairs. Some podal *DT* graphs are described below.

The Bisectional Network B_n [18]

A Bisectional Network is isomorphic to a folded Hypercube and is generated using all the binary codewords of length n with even weights. A node x in a Bisectional network is connected to a neighbor y if $H_{xy} = n-1$ [18]. We will denote a Bisectional network as B_n. The degree of a B_n network is n (odd) and it has 2^{n-1} nodes. For B_n, $k=(n-1)/2$ and $L_{xy} = Min(H_{xy}, H_{xy}^+)$ where $H_{xy}^+ = n - H_{xy}$.

For every node x in B_n, the valencies are given as

$v_i^x = \binom{n}{i}$ for $i = 0, 1, 2 \ldots n$.

The Binary Odd Network O_n [17]

The Odd graph is constructed by using binary codes with constant Hamming weight. This graph also belongs to the Johnson Association Scheme [5]. An Odd graph O_n has for its vertex

set the binary codewords of length $2n-1$ and Hamming weight $n-1$. Two vertices in O_n are connected if and only if the Hamming distance between them is $2n-2$. It has degree n, diameter $k = n-1$ and $\binom{2n-1}{n-1}$ nodes. Also, for O_n graphs, $L_{xy} = Min(H_{xy}, H^{+}_{xy})$ where $H^{+}_{xy} = 2n-1-H_{xy}$. The Odd graph O_3 is the celebrated Peterson graph [17].

For every node x in O_n[17], $v^x_i = \binom{n}{i}\binom{n-1}{i}$ for $i = 0, n-1, 1, n-2, 2, \ldots k$.

3. The Network Partitioning Strategy

In this section, we describe the criteria for partitioning both the podal and anti-podal *DT* topologies. We show that, in general, partitioning of the interconnection network and the selection of the set of fault-control points (FCPs), can be modeled as a problem which is NP-hard. Subsequently, we propose an efficient solution, for partitioning and finding the set of FCPs for the above mentioned networks, based on a combinatorial structure called *Hadamard matrix*. The proposed solution is "efficient" in the sense that the whole network is uniformly partitioned into spheres where each sphere is symmetric and is equal in size. In Section 3.3, we describe some properties of proposed use of Hadamard matrix for partition and in Section 5.3, we evaluate the efficiency and "goodness" of this scheme.

Based on this partitioning, we then propose a fault-tolerance strategy for an arbitrary distributed scheduling and load balancing mechanism, both for non real-time and real-time systems. In the proposed scheme, each sphere is assigned a subset of FCPs which are responsible for:

(a) maintaining redundant (backup) copies of the tasks in the network,
(b) monitoring failures in their spheres,
(c) in case of a failure, redistributing tasks of the failed node in individual spheres, and
(d) maintaining the load status of the nodes in the spheres.

We need to mention that for these responsibilities, the assignment of FCPs varies. For maintaining backups, the assignment of nodes to FCPs is independent of the physical distances in the network, which results in a rather simple rule of assignment (see Section 3.1). However, the average distance of a node to FCPs remain constant, as discussed in Section 6.3. On the other hand, for the above mentioned functionalities (b), (c) and (d), we use spheres centered at FCPs, as discussed in Section 3.2. In this paper, we do not address the issue of fault-diagnosis and failure identification. A sphere based fault-diagnosis scheme can be found in [18]. We assume the availability of some such scheme. An FCP is responsible for optimally assigning tasks within its sphere. The approach of load re-distribution, therefore, is semi-distributed in nature and is discussed in Section 4.

Let C be the desired set of FCPs. There can be various possible options to select C and to devise a fault–tolerance strategy based on this set. However, the performance of such a strategy depends on the "graphical locations" of the FCPs within the network and the *range of load redistribution* used by the FCPs. This range quantifies the graphical distance within which a FCP assigns tasks of a failed node to the nodes of its own sphere, where it is located at the center. The details of the backup copies, scheduling algorithm and information maintenance scheme are described in Section 4. In order to characterize spheres and to describe network partitioning, we need the following definitions.

Definition: Let the sphere assigned to a node $x \in C$ be denoted by $S_i(x)$, 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 node x. Since the number of nodes at the graphical distance i is given by valency v_i^x, the total size of the sphere is given as $|S_i(x)| = \sum_{j=0}^{i} v_j^x$. It should be noted that, in a centralized scheme where a single node acts as a FCP, i must be equal to the diameter (k) of the network.

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

We need a δ –uniform set C (for some δ to be determined) with graphically identical and symmetric spheres, in order to design a symmetric algorithm for FCPs. The size, $|C|$, depends on the selection of δ. Intuitively, larger δ yields smaller $|C|$, but also spheres with larger size. It can also be observed that reducing $|C|$ increases the sphere size and vice versa. In addition, a number of other considerations for the provision of fault–tolerance are given below:

(1) Since, in case of a failure an FCP needs to re–distribute tasks of the failed node to all the nodes in the sphere, which requires the FCP to maintain state information of load of all the nodes within the sphere (the next section regarding load re–distribution), the diameter of the sphere should be as small as possible.

(2) Also, the size of the sphere should be small because a FCP (x) needs to send/receive $|S_i(x)|$ messages for scheduling of failed–node tasks within the sphere.

We now describe the complexity of selecting a δ –uniform set (C).

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

(a) Finding a uniform set C in an arbitrary graph is NP–hard.

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

***Proof*:** see Appendix

The above theorem provides a rather pessimistic view for finding a set C for a given DT network. However, we present an interesting solution to select the set C in these networks using a combinatorial structure called *Hadamard Matrix*. The reasons for choosing Hadamard Matrix are given in section 3.2. Its definition is given below.

Definition: A Hadamard matrix M is a j by j matrix with ± 1 entries, such that $MM^T = jI$,

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

$$M_S = \begin{matrix} 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 0\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \\ 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ 0\ 0\ 1\ 0\ 0\ 1\ 1\ 1 \\ 0\ 1\ 0\ 0\ 1\ 1\ 1\ 0 \\ 0\ 0\ 0\ 1\ 1\ 1\ 0\ 1 \\ 0\ 0\ 1\ 1\ 1\ 0\ 1\ 0 \\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0 \end{matrix}$$

Figure 1. An 8×8 Hadamard Matrix in 0–1 notation along with its complement and supplement.

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. 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 a 8×8 Hadamard matrix and its complement. 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) [21].

Definition: Two Hadamard matrices in 0–1 notation which disagree in all but one entry in each row are called supplements to each other.

Theorem 2: If a Hadamard matrix M comes from SBIBD, then a supplementary matrix M also exists

Proof: see Appendix

3.1. Assignment of FCPs to Nodes for Backups

We now describe the partitioning strategy for the networks under consideration and describe rules for assigning FCPs to the nodes in the network. As mentioned earlier, each node has two types of FCPs, the primary and the secondary FCPs, in order to provide two–level fault–tolerance. There are many ways to assign nodes to FCPs. In this paper, we describe simple rules for both podal and anti–podal networks and do not carry out any optimization in this regard. However, we do provide an estimate of overhead for this assignment in Section 5, and show the average distance is constant for all the nodes in the network. Furthermore, the assignment of FCPs to nodes is symmetric and each node interacts with the same number of primary and secondary FCPs. Also, each FCP manages the same number of nodes in terms of providing backup for fault–tolerance and re–distributing the load, if required.

The set C of FCP's for DT networks is selected from the code generated by taking combinations of the rows of Hadamard matrix M, its complement M^C, or its supplement M_s. The selection depends upon the network being podal or antipodal. For antipodal networks C consists of matrices M and M^C (both suitably modified depending upon the code structure of the network). C is also called *Hadamard code*. For podal networks, the set C consists of matrices M and M_s (both also suitably modified depending upon the code structure of the network). The details of selecting set C for partitioning a given network are explained below.

3.1.1 FCPs for the Binary *n*–cube Network and Node Assignment

For Q_n, (n being multiple of 4), we take the matrices M and M^C of Figure 1. Note, for this value of n, $|C| = 2n$ for Q_n. For the rest of the values of n, the rules for selecting set C are as follows.

Case a: Q_n with n mod 4 = 1. For this case, we start with the set C obtained from Hadamard matrices M and M^C of size n (Figure 1). The modified set C for the network under

consideration can be generated by appending an all 0's and an all 1's column, to M and M^C respectively, at any fixed position, say at the extreme left position.

Case b: Q_n with n mod 4 = 2. This case is treated the same way as the Case (a), except we 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 mod 4 = 3. For this case, the set C consists of the rows of the truncated matrices M and M^C in 0–1 notation. The truncated matrices (in 0–1 notation) are generated by discarding the all 0's row and column.

The FCP Assignment Rule: Once the FCPs are identified in Q_n, the rule for the assignment of nodes to FCPs is as follows. The FCPs having the left most bit 0 serve as primary FCPs for those nodes which have also left most bit as 0, but the 0 serve as secondary FCPs for the rest of the nodes in the network. Similarly, the FCPs having the left most bit as 1 serve as primary FCPs for those nodes which have also left most bit as 1 and serve as secondary FCPs for the rest of the nodes.

It can be noticed that the above assignment is symmetric in the sense that each node is assigned the same number of primary and secondary FCPs and each FCP manages the same number of nodes in terms of providing backup for fault-tolerance and load re-distribution in the sphere.

3.1.2 FCPs for the Binomial Network and Node Assignment

Depending upon the value of n, the set C for this network can be selected as follows.

Case a: n even. For this case, the set C is selected by truncating, both row and column of all 0's of $(2n-1)$ x $(2n-1)$ M and taking also its complement (see Figure 1 for $n=4$).

Case b: n odd. For this case, the set C is generated by taking a $(2n-2)$ x $(2n-2)$ matrix M (with only all 0's row truncated), appending an all 1's column with it and taking its complement as well.

The FCP Assignment Rule: The rule of assignment is exactly the same as Q_n, that is the FCPs having the left most bit 0 serve as primary FCPs for those nodes which have also left most bit as 0 and serve as secondary FCPs for the rest of the nodes in the network. Similarly, the FCPs having the left most bit as 1 serve as primary FCPs for those nodes which have also left most bit as 1 and serve as secondary FCPs for the rest of the nodes.

It can be noticed that the above assignment is symmetric.

3.1.3 FCPs for the Bisectional Network and Node Assignment

For B_n, depending on n which is always odd, the following are the two cases, for selecting the set C.

Case a: n mod 4 = 3. For this case, we consider the set C obtained from Hadamard matrices M of size n (Figure 1), and just truncate the left most column of all 0s. Also, we translate this truncated matrix within B_n by complementing any $n-2$ columns. Figure 2, shows M and one possible translation. Note, the newly translated matrix also produce a set of FCPs which have same graphical distances among themselves as the original matrix M.

$$M = \begin{matrix} 0\ 0\ 0\ 0\ 0\ 0\ 0 \\ 0\ 0\ 1\ 0\ 1\ 1\ 1 \\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \\ 1\ 0\ 1\ 1\ 1\ 0\ 0 \\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ 1\ 1\ 1\ 0\ 0\ 1\ 0 \\ 1\ 1\ 0\ 0\ 1\ 0\ 1 \\ 1\ 0\ 0\ 1\ 0\ 1\ 1 \end{matrix} \qquad \text{Translated } M = \begin{matrix} 1\ 1\ 1\ 1\ 1\ 1\ 0 \\ 1\ 1\ 0\ 1\ 0\ 0\ 1 \\ 1\ 0\ 1\ 0\ 0\ 0\ 0 \\ 0\ 1\ 0\ 0\ 0\ 1\ 0 \\ 1\ 0\ 0\ 0\ 1\ 1\ 1 \\ 0\ 0\ 0\ 1\ 1\ 0\ 0 \\ 0\ 0\ 1\ 1\ 0\ 1\ 1 \\ 0\ 1\ 1\ 0\ 1\ 0\ 1 \end{matrix}$$

Figure 2. A Hadamard matrix and its translated matrix.

Case b: n mod 4 = 1. For this case, the set C is generated by modifying the set of case (a), by appending two columns of all 0's to M at any fixed position, say at extreme left, and then translating it as well, as described in case (a).

The uniqueness of M and its translation is established by the following lemma.

Lemma 1: The FCPs in M and its translation are unique.

Proof: Since, the translation is done by complementing $n-2$ columns, and for B_n, $L_{xy} = Min(H_{xy}, H_{xy}^{+})$ where $H_{xy}^{+} = n - H_{xy}$, the FCPs in the translated matrix are in the immediate neighbors of the original FCPs of truncated M. Since, FCPs in the original M cannot be immediate neighbors of each other (distance is equal to the diameter, this is shown in Lemma 2 in section 3.3), the translated FCPs cannot be in the original matrix M.

The FCP Assignment Rule: The assignment rule is exactly the same as for Q_n.

By noticing, that the left most bit of half of the nodes in B_n is 0, the above assignment is also symmetric.

The Binary Odd Network O_n [17]

Depending upon the value of n, the set C for this network can be selected as follows.

Case a: *n* even. For this case, the set C is selected by truncating, both row and column of all 0's of $(2n-1)$x$(2n-1)$ M and as well as taking its truncated (both row and column with all 0's) supplement matrix (see Figure 1 for $n=4$).

Case b: *n* odd. For this case, the set C is generated by taking a $(2n-2)$x$(2n-2)$ matrix M and its supplement (for both only all 0's row is truncated), and appending an all 1's column with them.

The FCP Assignment Rule: In a O_n network, the number of nodes, N, is always even (except for $n=4$) [5]. The rule of assignment is based on lexographical ordering of positions in the codewords associated with the nodes in O_n. In this ordering the first half codewords (nodes) are assigned to M (acting as primary FCPs) with its supplement acting as secondary FCPs. For the rest of the codewords (nodes) the assignment is reversed. For even N, this results in a symmetric assignment.

3.2. Sphere Identification for Load Re-Distribution

As mentioned earlier, FCPs maintain backups and load status and re-distribute load within certain regions in the network, known as spheres. The sphere of an FCP consists of all the nodes which are within a distance r from FCP, where r is the covering radius of C. The number of nodes in, the sphere, $S_r(j)$ is the total number of nodes lying at graphical distances 0 through r, from node x. Since the number of nodes at the graphical distance i is given by valency v_i^x, the total size of the sphere is given as $|S_i(x)| = \sum_{j=0}^{r} v_j^x$.

We recall that the determination of this covering radius is a non-trivial problem.. However, in Appendix we have provided an upper bound on the covering radius for DT graphs (see Lemma 3). We need to determine this radius at the time the distributed systems is designed and its topology is identified and mapped on to a suitable DT graph. Then through an exhaustive search the covering radius can be found. For that search, Lemma 3 can of great help.

In case the number of nodes in the network is not divisible by the number of FCPs (as is the case for B_{11}), the difference in sphere sizes does not exceed by 1. Figure 3 shows one of the 16 spheres in Q_8, for the FCP with the binary codeword of 00000000. The covering radius r in this case is equal to 2 and the valencies v_0^x, v_1^x and $v_2^x, \forall x$, have values 1, 8 and 28 respectively, corresponding to total volume of the sphere $|S_r(x)|$ equal to 37.

It can be noticed, that the nodes in one sphere can also be shared by other spheres, depending upon the covering radius r and the graphical distance among FCPs.

3.3. Some Properties and Examples of Set *C*

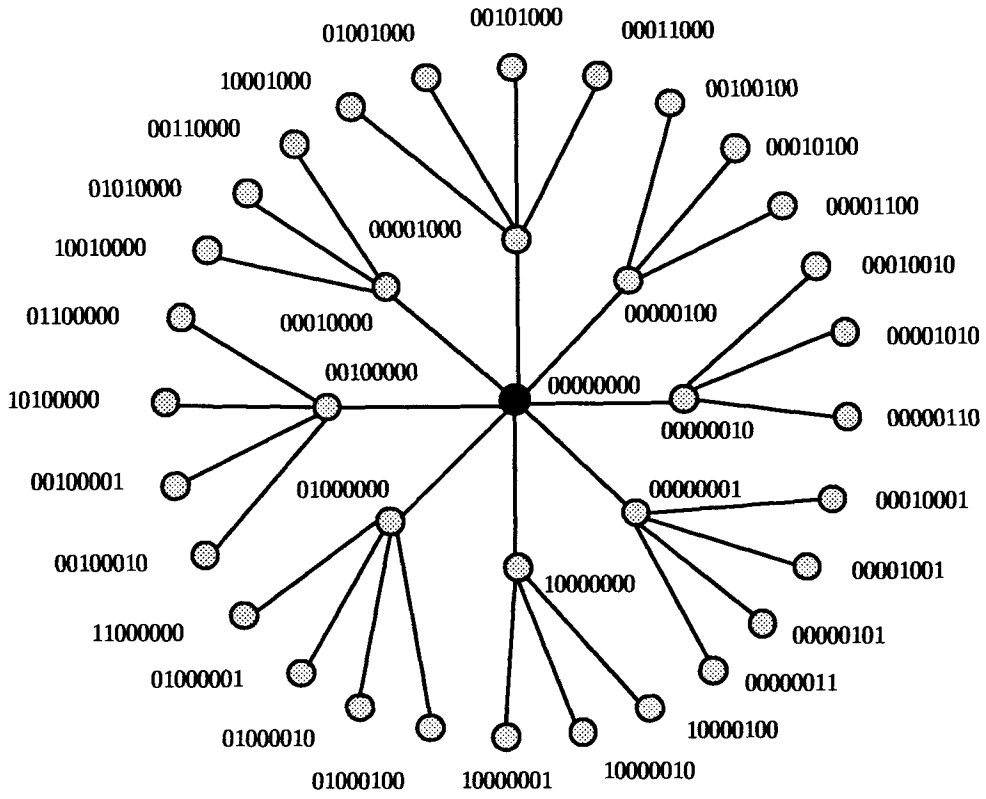

Figure 3. A sphere in Q_8 network with scheduler 00000000.

Following are the main reasons for choosing Hadamard code for the set C (we might as well select other codes such as Hamming code or BCH codes, but these codes have certain limitations as described below).

1. The range of values of n for which a Hadamard code exists, considerably exceeds the range of n for which other codes, such as the Hamming code, exist. As described earlier, it is conjectured that a Hadamard matrix exists for all values on n which are less than 428 and are multiple of 4 [21]. 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.

2. The covering radius of C is known for all values of n [23], which are even powers of two.

3. The following theorem shows that the set C, provides the maximal $k/2$ (rad*ius)–uniform set* for Q_n and Γ_n.

Theorem 3: Let $x, y \in C$. Then for Q_n and Γ_n, $L_{xy} = k/2$, and C is the maximal possible *$k/2$–uniform set.*

Proof: We first consider Q_n, for which $k=n$. The Hamming distance between any two rows of a Hadamard matrix is $n/2$, that is for Q_n, $L_{xy} = H_{xy}$. In order to prove that the cardinality of the set is the maximum possible, assume the contrary is true, and suppose there exists some codeword z, such that $H_{zx} = n/2$, for all $x \in C$. A simple counting argument reveals that there must be at least $n(n-1)/4$ 1's at those $n/2$ columns where z has 0's. If these 1's are distributed among all rows of M, then there are at least $(n-1)-n(n-2)/[4(n-1)]$ rows which can not be filled to obtain this Hamming distance. Therefore the node z is at a graphical distance less than $n/2$ from these row. The proof for Γ_n can be provided on the same line. Q.E.D.

Lemma 2: A truncated matrix M with and without all 0's row provides a *k–uniform set* for O_n and B_n networks, respectively. Proof is obvious from Theorem 3.

A truncated Hadamard matrix (the one without all 1's column) using Symmetric Balanced Incomplete Block Design (SBIBD) [21] 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 using the so called *difference set approach* [21]. Table 1 illustrates the generator codewords for various values of $n-1$. Similarly, for the supplementary matrix, we can have supplementary generators, as elaborated in the proof of Theorem 2 in Appendix.

Examples

As the first example we consider the set of FCPs 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 consists of rows with all 0's and all 1's plus the seven codewords and their complements. Therefore, $C=\{0000000, 0010111, 0101110, 1011100, 0111001,$

Table 1. Generator codes for different lengths

Length = $n-1$	Generator Codewords	Suppl. Generator Codewords
7	0010111	1101001
11	10111000101	01000111011
15	111101011001000	100010100110111
19	1001111010100001101	0110000101011110011

1110010, 1100101, 1001011, 1101000, 1010001, 0100011, 1000110, 0001101, 0011010, 0110100, 1111111}. For Q_8, the set C can be produced by choosing the untruncated matrices, 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 as 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, as described for case (a). Also, the same set can be used to generate the set C for Q_{10}, as described in the procedure of case (b). The set C for other Q_n's can be generated by the methods described above.

As a second example we consider O_6, for which the generator codeword is 10111000101. The additional 10 codewords generated by taking 10 left cyclic shifts of this generator, constitute the set C for O_6.

The third example is of O_4, which is shown in Figure 4. Its set C consisting of matrices M and M_S is also identified.

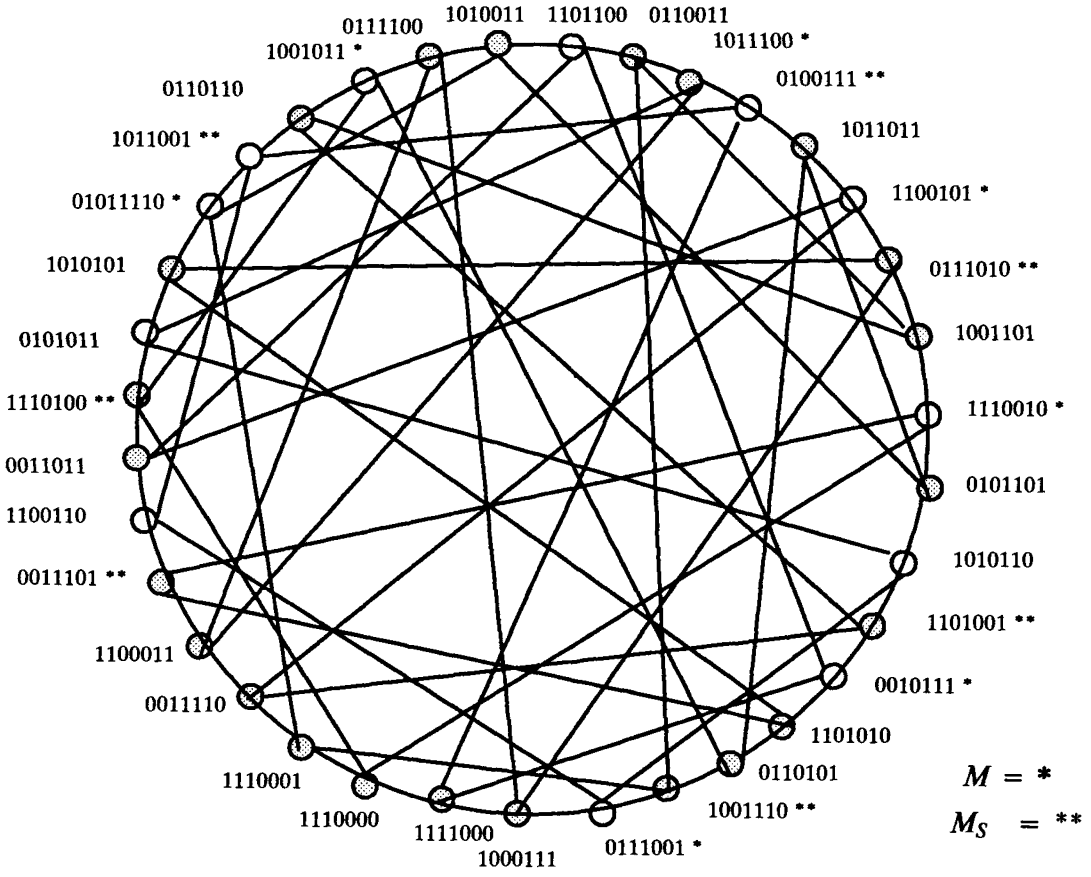

Figure 4. Odd Graph O_4 with its set C

Table II

The characteristics of various partitioned DT networks

Network		N	n	δ	$\|C\|$	r	v_1^x	v_2^x	v_3^x	$\|S_r(x)\|$
Hypercube	Q_7	128	7	4	14	1	8	-	-	9
	Q_8	256	8	4	16	2	8	28	-	37
	Q_9	512	9	5	16	2	9	36	-	46
	Q_{10}	1024	10	5	16	3	10	45	120	176
Bisectional	B_7	64	7	3	8	1	7	-	-	8
	B_9	256	9	4	8	2	9	36	-	46
	B_{11}	1024	11	5	12	3	11	55	165	232
Odd	O_4	35	4	3	7	1	4	-	-	5
	O_6	462	6	5	11	3	6	30	75	112
Binomial	Γ_4	70	4	4	14	1	4	-	-	5
	Γ_5	252	5	5	14	2	5	20	-	26

The topological characteristics and partitioned structures for Q_7, Q_8, Q_9, Q_{10}, B_7, B_9, B_{11}, O_4, O_6, Γ_4 and Γ_5 networks are summarized in Table II which shows the number of nodes N, the degree n of each node, the minimum distance δ between FCPs, the cardinality of the set C, the covering radius r, valencies v_i^x and the size of sphere $|S_i(x)|$, for each network.

As mentioned earlier, the nodes in one sphere can also be shared by other spheres, depending upon the *range of load redistribution* and the graphical distance among nodes and the set C. The distribution of shared nodes at various distances with varying range of load redistribution (f) within the sphere is given in Table III, for Q_8. For example, with the range of load re-distribution equal to 2, which is also the covering radius of C in this case, nodes at distance 1 from a FCP are present in only one sphere whereas nodes at distance 2 are shared by exactly 4 spheres. On the other hand, if the range is set to 8, the system is equivalent to the centralized model with 16 nodes trying to re-schedule failed nodes tasks to all the rest of the nodes in the network. Increasing the range of load re-distribution beyond the covering radius causes more sharing of nodes among spheres for which greater number of messages need to be generated to keep the load information consistent for all FCPs. Therefore, the optimal range of a load

re–distribution, the one which provides maximum coverage with minimum radius in all the cases is set to the covering radius of the corresponding Hadamard Code.

Table III
Distribution of nodes shared by different spheres in a 8–cube
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

4. The Proposed Fault–Tolerance System

In this section, we present the proposed semi–distributed scheme. For this purpose, we describe the system model, the load balancing and load re–distribution algorithms and the associated information collection and backup mechanisms.

4.1. Assumptions and Characteristics of the System

The system consists of N nodes which are connected by a communication topology with a constant degree, n, per node. Each node is subject to arrival of tasks and is equipped with a task scheduler. Task scheduling and load balancing, therefore, assumed to be completely decentralized. The tasks arrive at a node with rate λ tasks/time–unit which is identical for all the nodes. The execution time of a task is assumed to be known. In addition, associated with each task is a deadline. Hence, the completion time of a task must be less than its deadline.

When a task arrives at a node, the scheduler of that node tries to guarantee that the task meets it deadline. If this deadline can be met locally, the task is scheduled into the local execution queue which is served on the FCFS principle. If a task cannot meet its deadline locally, it is transferred to another node. The selection of a remote node can be done in various possible ways [31], [33], [34]. However, in this study, we assume that the local scheduler interacts with its immediate neighbors only and gathers their load status. The load status in this case consists of the accumulative execution time of the unfinished work load which is the sum of the execution times of the tasks waiting in the node's execution queue plus the remaining

execution time of the task which is currently being executed. This load index has been suggested in many studies [33], [34] since the simple queue length can not be a good load index for real–time systems. If a task is to be transferred to another node, such transfer takes certain amount of time. This time is the sum of initial channel setup time and the actual transmission time of the task. If a task is currently being transferred on the same channel, then this task has to wait in the communication queue until the prior task completes its migration. Since the communication time also counts towards the task's waiting time, the local node, while making the scheduling decision, also takes into account the communication penalty. The effective load of a neighbor is, therefore, the computation load plus task's communication time. Since, the determination of the exact communication time of a task is difficult. , the scheduler only assumes an average value of communication time. The deadline of a task , therefore, consists of its execution time plus the average communication plus some marginal value (D) which depends on the application and is supplied by the user.

A task is said to have missed its deadline if does not complete its execution anywhere in the system within that deadline. If a node cannot guarantee a task to meet its deadline and fails to find a suitable node as well, it still transfers the task to a neighboring node with the minimum load. This results in two possible advantages. First, by the time the task completes its migration to the neighboring node, the loading conditions of that task may change in favour of the task which can result in that task meeting its deadline. If this is not the case, the neighboring re–migrate the task to one of its own neighbors.

4.2. Task Replication and Backups at FCPs

In order to provide two level fault–tolerance, the replicated copies of every task are kept as backups on primary and secondary FCPs. Since the major goal of the proposed strategy is to quickly re–distribute backup load, the backup queues themselves are distributed in a round–robin fashion, first to all the primary FCPs and then again to all the secondary FCPs. Specifically, whenever a task is scheduled at a node, its copy is sent to the one of the FCPs as well as to the associated secondary FCP. The copy of the next task is sent to the next primary FCP present in the round–robin list as well as to the associated secondary FCP. In this manner, the execution queue of a node is always equally divided among its primary FCPs with further duplications at secondary FCPs. An FCP, therefore, needs to keep backup queues for those $N/2$ nodes for which it acts as primary FCP. It also maintains backup queues for the rest of the $N/2$ nodes for which it serves as their secondary FCP. It is worth mentioning that in order to provide two level redundancy, there have to be $2N$ redundant copies in the whole system. It is also possible that, any given time, an FCP(s) may be faulty. In that case, the copy of the task is sent to to next available FCP in the round–robin list. In summary, the following

task duplication algorithm is executed at each node:

```
Next FCP = ( Next FCP + 1 ) mod ( Number of FCPs )
While ( Next FCP is faulty )
Do
Next FCP = ( Next FCP + 1 ) mod ( Number of FCPs )
End Do
Send copy of the task to Next FCP
If ( the secondary FCP for the Next FCP is not faulty )
Send the copy of the task to the secondary FCP
```

When a task is completed at a node, a message is sent to both the primary and secondary FCPs to delete that task from their backup queues which also works on FCFS basis. Figure 5 shows two nodes 00000000 and 11111111 with sixteen FCPs in Q_8. In this case, eight FCPs corresponding to the binary codewords of matrix M, represent the primary FCPs for the node 00000000 and the binary codes of M_c represent secondary FCPs for that node. On the other hand, for node 11111111, the role of M and M_c is reversed. The round–robin order is also indicated in this figure.

4.3. Failure and Repair Model

As mentioned earlier in real life systems, failure of components are generally random. Also the failed components, once diagnosed and repaired off–line, can be integrated back into the system. Typically, a node remains alive most of the time and when it fails, it can be repaired quickly and can become operational. For our study, we assume only the failure of nodes in the system. Accordingly, we consider the well known failure/repair model of multiprocessor systems where the availability time as well as the repair time of the processor are assumed to be independent exponential random variables with rates γ and μ_R, respectively. Generally, the ratio of γ/μ_R is assumed to be very small. Since the number of nodes (N) in the system are fixed and it has finite population, therefore, the accumulative failure and repair rates become state dependent [13]. Accordingly, the Markov model showing the failure and repair processes is depicted in Figure 6. The state of the model represents the number of nodes (P) currently operational. At a give time, every operational node is equally probable to fail. Obviously, there can be more than one node in the faulty list. The state dependent rate of the failure and repair processes are also shown in Figure 6.

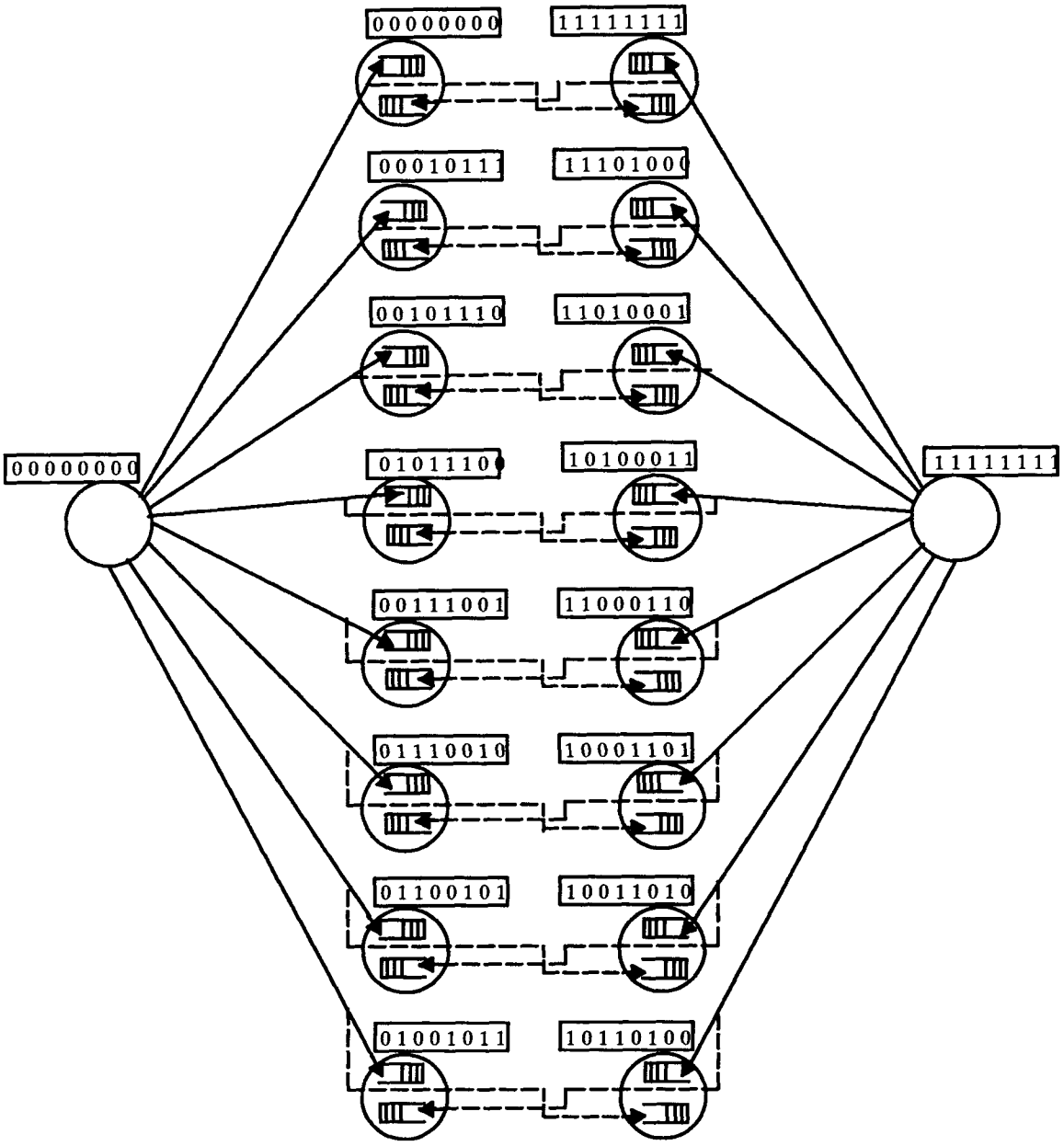

Figure 5. Task replication at FCPs assigned to node 00000000 and 111111.

4.4. Load Re-distribution under Failures

When a node fails, each primary FCPs re-schedule the backup queue (if they have any) of that node within its sphere. Due to real-time constraints, each FCP tries to make sure that the backup tasks still meet their deadlines. This is accomplished by scheduling each of the back-up tasks to the most lightly loaded nodes of the sphere. The notion of sphere provides an FCP

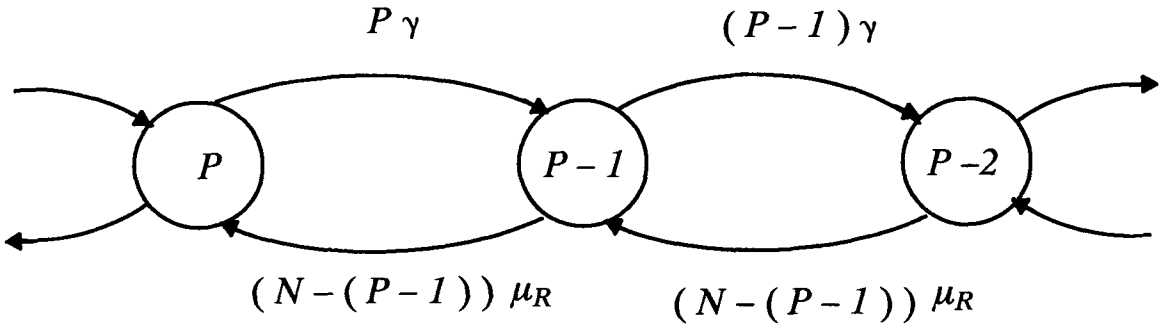

Figure 6. Markov model for failure and repair processes.

with a broader view of the state of the system and it enables to re–distribute the backup load by selecting the most suitable nodes. As a result, the backup tasks can still possibly meet their deadlines despite the failure of their original nodes. Since the backup queue of the failed node is itself distributed among multiple FCPs, each FCP needs to schedule a portion of the tasks of the failed node. This semi–distributed load management strategy has the following four advantages:

(1) Many FCPs are able to select best candidate nodes within their respective spheres.

(2) All the backup tasks can be concurrently re–scheduled in multiple sphere.

(3) Due to the round–robin distribution of tasks in backup queues, the danger of instability due to bulk arrival in a particular sphere is greatly reduced.

(4) The backup load is smoothly spread across the whole system since the graphical distances among FCPs is either equal to the diameter k or radius $k/2$.

The migration of a task from an FCP to a node within its sphere incurs some communication delay. After re–distributing the load of the faulty node, all backup queue are deleted. At the same time, the primary FCP informs the corresponding secondary FCP to delete its backup queues for the failed node. If one of the primary FCPs is also faulty, the backup queue of the faulty node is re–scheduled by the secondary FCP in a similar fashion. The node receiving the re–scheduled task treats the backup tasks as a newly arrived task. The node also sends the backup copy of the re–scheduled task to the FCP, which is next in its round–robin list. However, the re–scheduled asks are not allowed to make any further migrations.

In spite of two level redundancy, a task can still be lost if:

(1) Both primary and secondary FCPs of the failed node have also been failed.

(2) The failed node is also an FCP and no task backup at the secondary level could be made

because the secondary FCP was also faulty at the time the task was scheduled.
(3) The failed node is also an FCP and its secondary FCP has also failed.

However, the likelihood of these events is very small and the probability of lost tasks as a result of these events is presented in section 5.3. The most important advantage of the proposed scheme is that only a small percentage of the tasks can definitely get lost provided the node and one of its primary/secondary FCP pair also fail, an event of very low possibility. It can be noticed that this percentage is of the order of $\Theta(1/n)$.

5. Evaluation of the Proposed Fault-Tolerant Strategy

In this section, simulation results for the proposed strategy are presented. We have considered all four above mentioned networks for which extensive simulation experiments have been conducted. The performance measures selected include the deadline missing probability, the mean response time of a task and the percentage of re-scheduled tasks which still meet their deadlines despite node failures. Also, an estimate for the average number of control messages generated per task is provided. The sensitivity of the performance with respect to varying deadlines, fault rate, network communication rate and system load is also analyzed.

5.1. The Simulator

Our simulator models the node architecture described above with task scheduler and execution and communication queues. The simulator also takes into account the network topology with any size, the rate of communication links, different scheduling algorithms, number of FCPs and their assignments, task arrival rate, execution rate, failures and repairs processes and network partitioning for semi-distributed task re-scheduling. The simulator allows to tune various parameters such as arrival rate, execution rate, communication rate, failure rate, repair rate and average task deadline. Length of simulation with respect to time or number of tasks can also be varied. In discrete event simulation, it is very important to eliminate the initial transients for steady state system behavior. For validity of results a simulation run should be long enough and multiple runs of the same simulation, by using different set of seeds for random number generators, need to be carried out, to compute the confidence interval. Our simulator explicitly takes into account these considerations. It is written in '*C*' and runs on an Encore Multimax.

The task arrival process for this study been modeled as a Poisson process with average arrival rate of λ tasks/unit-time which is identical for all nodes. The execution and communication times of tasks have been assumed to be exponentially distributed with a mean of $1/\mu_E$ time-units/task and $1/\mu_c$ time-units/task, respectively. The task deadline has been computed by generating a random number from a uniform distribution with an average of D time-

units. Node failures and repairs rates are also assumed to be exponentially distributed with a mean of $1/\gamma$ time-units/node and $1/\mu_R$ time-units/node, respectively. In each simulation run, 100,000 tasks were generated. All results are presented with 95 percent confidence interval, with the size of the interval varying up to plus or minus 5 percent of the sample mean.

5.2. Deadline Missing Probability and Average Task Response Time

The two major performance measures are deadline missing probability and the average task response time. For soft real-time systems, the important performance measure is the missing probability but since the proposed strategy is also intended for non-real-time systems, we have also considered its performance in terms of average task response time. The average task response time also provides a perspective of how a task's system sojourn time is affected under node failures. The deadline missing probability is defined as the probability that a task does not finish its execution within its specified deadline [33]. The task response time (or sojourn time) is defined as the finish time of the task minus its arrival time. In our study, the impact of four important system parameters on these performance measures is evaluated. These parameters include the frequency of node failures (γ), the deadline (D), system load (λ/μ_E) and the task communication rate (μ_C) over the links. Simulation results are presented for all four classes of DT networks. The next section discusses the impact of frequency of failures.

5.2.1. Impact of Frequency of Failures

The frequency of failures affects the amount of load which is injected back into the system. Recall that in the proposed strategy, the load submitted to the faulty nodes, while they are under repair, is not rejected. Rather, the new arrival of tasks for the faulty node is assigned randomly to primary FCPs. Therefore, in addition to load re-distribution, FCPs are also subjected to some additional load.

Failure and repair rates are be independent with respect to the rest of the system parameters. We define a parameter which captures the effect of failure and repair rate. That parameter is the percentage of all the tasks. in a given simulation run, which are re-scheduled. The failure rate per node is varied from 0.002 to 0.2 which implies that the average interval between failure of every node has a value of 1/0.002 to 1/0.2 time-units. The repair rate has been chosen to be 0.5 which implies that the average repair time of a node is 2 time-units. This repair rate is kept fixed in the rest of this paper whereas the failure rate has been varied. For the results shown in this section, the arrival rate per node (γ) is 0.8, D is 0.5, and communication rate (μ_C) is set to be 10 tasks/time-unit. Figure 7 shows this percentage with various failure rates, for Q_8, B_9, O_6 and Γ_5. We notice that the percentage of re-scheduled tasks

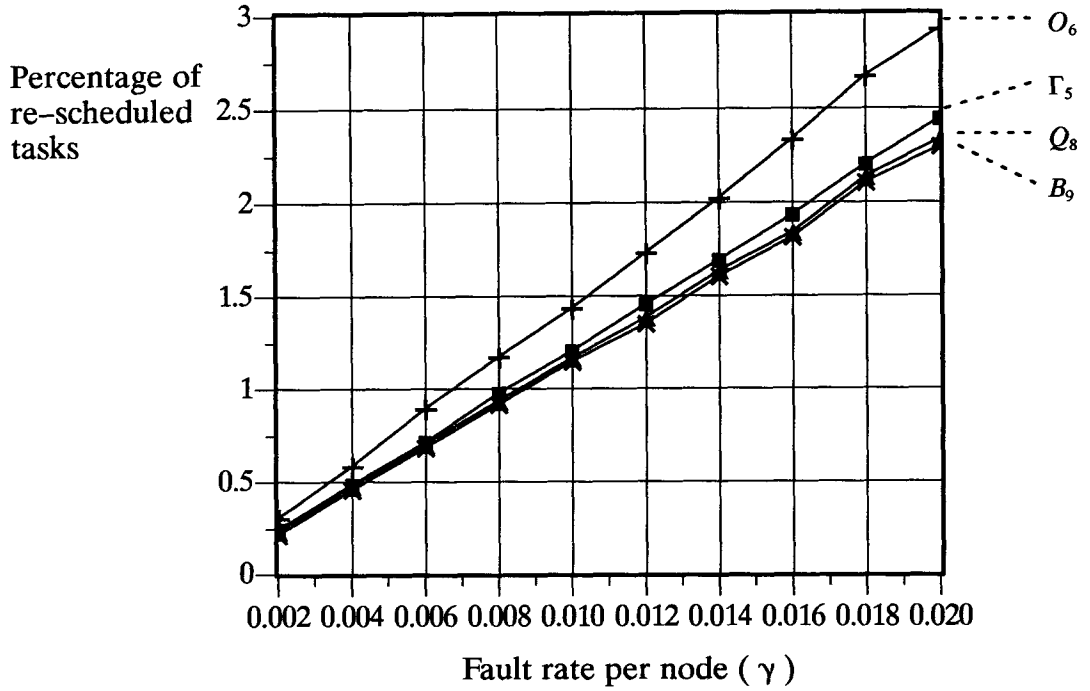

Figure 7. Percentage of re-scheduled tasks at various fault rates for all four networks at $\lambda = 0.8$ and $D = 0.5$.

increases with γ. This value is large for O_6 network, due to its bigger size, while for Q_8, B_9 and Γ_5, this value is almost the same.

For the same set of parameters, the impact of failure rate on the deadline missing probability of a task is shown in Figure 8. The missing probability at failure rate equal to 0 is the missing probability under normal system operation without any failure. As the failure rate (γ) increases, the missing probability also increases because not only the tasks (if any) waiting in the execution queues of the failed nodes have to be re-scheduled but also the tasks being executed at the time of failure have to be aborted and re-scheduled. It is important to notice that in contrast to Figure 7 where the percentages of re-scheduled tasks increases sharply with the increase in γ, the missing probability does not increase rapidly. This show that the proposed strategy is able to sustain high failure rates.

5.2.2. Impact of Deadline

Simulation results presented in this section examine the impact of deadline on the missing probability and the average response time. Clearly, a task's missing probability is dependent on the specified deadline. Recall that the deadline of a task in simulation is computed by adding a task's execution time to its associated value of D and $1/\mu_C$. For simulation, the average value of parameter D is varied from 0.25 to 2.5 time-units, λ is 0.8 and μ_C is 10. Since D is

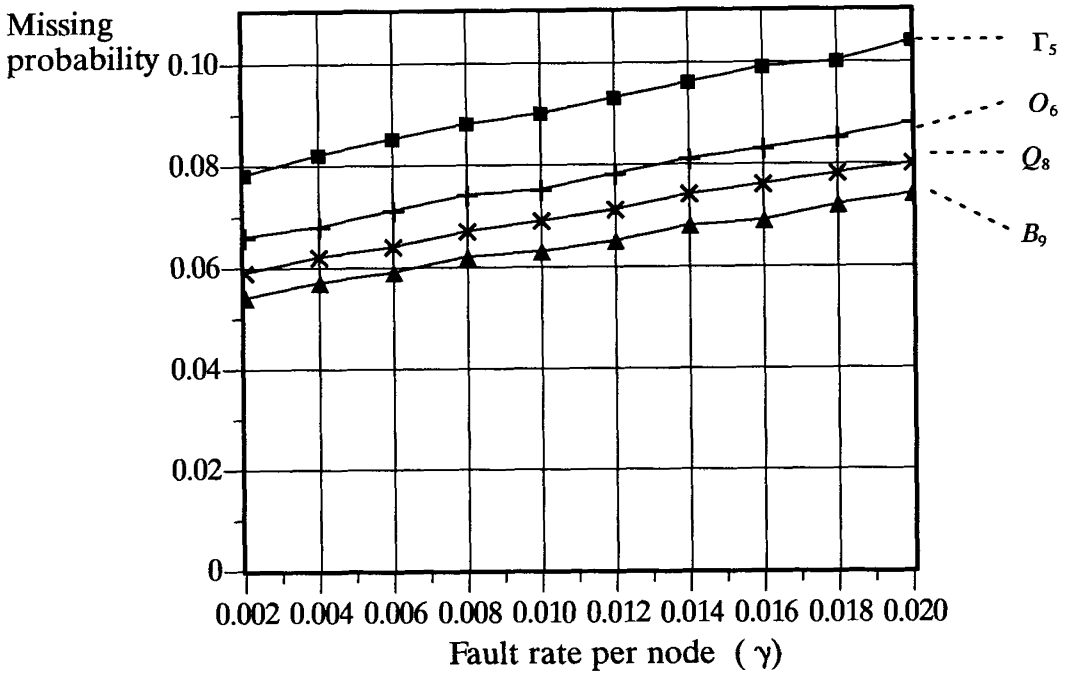

Figure 8. Deadline missing probability at various fault rates for all four networks at $\lambda = 0.8$ and $D = 0.5$.

a uniformly distributed random variable, the minimum value of D is 0 while the maximum value ranges from 0.5 to 5.0. Four different values are selected for γ, which are 0.005, 0.01, 0.015 and 0.02. Plots showing the percentage of re–scheduled tasks, the missing probability and the average response time are presented for each of the four networks. In addition, results for the missing probability and the average response time for the no failure case are also included.

For Q_8 network, these results are shown in Figure 9 to 11. The percentage of re–scheduled tasks, as shown in Figure 9, exhibits a constant increase when γ is varied from 0.005 to 0.02. It also indicates an increasing trend when D is varied from 0.25 to 2.5. This is because the local execution queue lengths increase if D is increased which in turn is due to the fact that schedulers at all nodes are frequently able to guarantee tasks locally. As a result, lesser number of tasks are transferred to neighbors. Hence, the execution queues get longer thus causing backup queues to become larger as well. Therefore, when a node fails, an FCP needs to re–schedule relatively greater number of tasks.

The deadline missing probability (Figure 10) shows a sharp decrease when D is varied from 0.25 to 1 after which it shows a saturation behavior. With constant increment in γ, the missing probability is also shown to increase with constant increments, irrespective of the value of D. Hence, the amount of degradation in performance due to failure remains the same

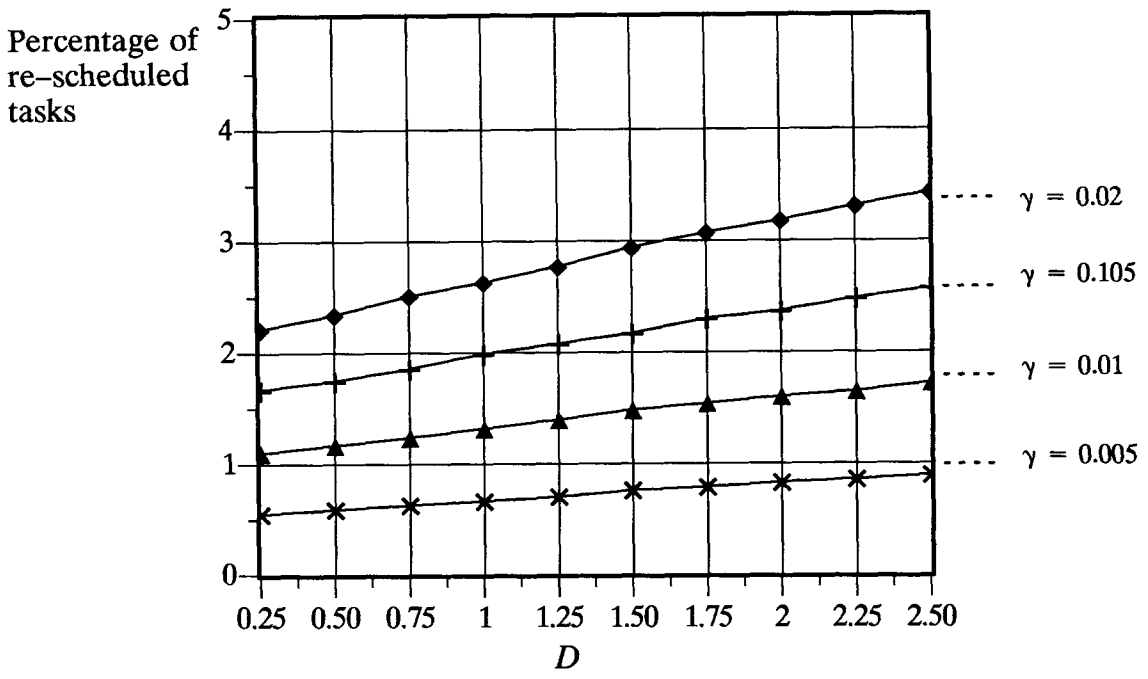

Figure 9. Percentage of re-scheduled tasks with variable deadline for Q_8 network at $\lambda = 0.8$ and various γ's.

for any value of D. This indicates that the proposed strategy is able to tolerate failures under both strict ($D = 0.25$) and relaxed ($D = 2.5$) conditions for deadline. Also the degradation in performance is dependent on γ.

The response time, as shown Figure 11, is also affected if deadline is increased. This is also because schedulers tend to schedule more tasks in local queues. A relaxed deadline with higher value of D can result in more tasks meeting their deadlines but at the expense of an increase in the average task response time. However, the increase in response time, as expected, also shows its light dependence on γ.

Simulation results for the B_9, O_6 and Γ_5 networks, providing information regarding the percentage of re-schedule tasks, the missing probability and the average response time, have also been obtained. The trends of these parameters for these networks are identical to the results for Q_8 as shown in Figures 9 through 11. We have only included results pertaining to the real-time performance of the system which are the percentage of re-scheduled tasks and the missing probabilities for these networks. Figures 12 to 17 show these results. We have not included the results for the average response time due to the space limitations of the paper, however, they exhibit the same behavior. The curve for the missing probability for B_9 although show the same pattern as for the case of Q_8, a careful look at these curves reveals that the performance of B_9 is better than Q_8, although both networks have the same number of

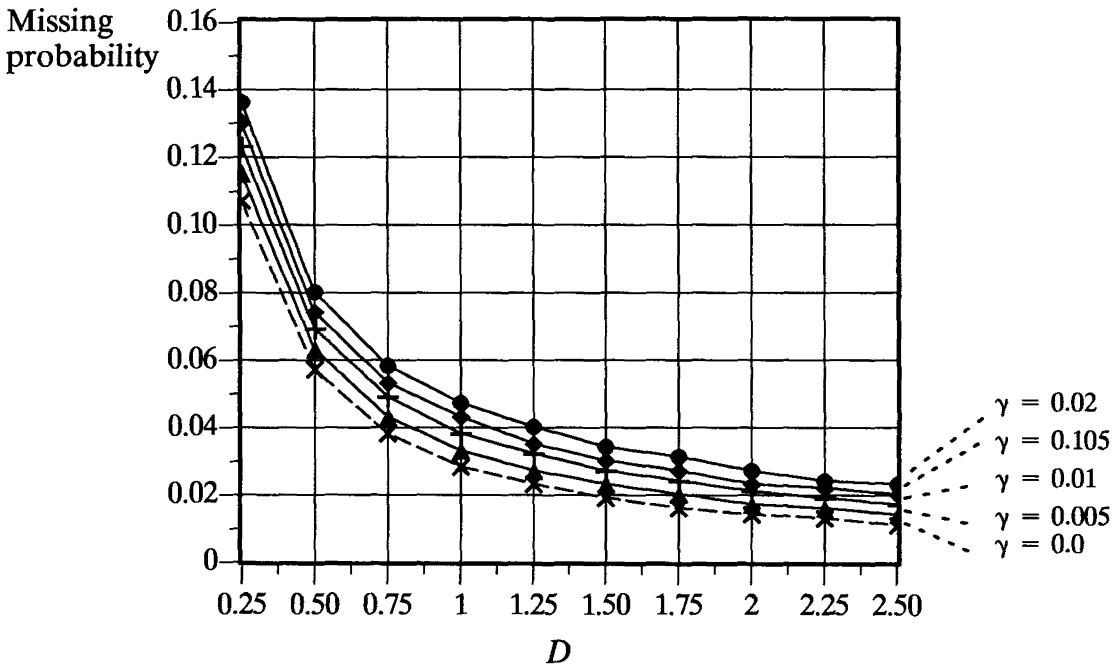

Figure 10. Deadline missing probability versus average D for various γ's at $\lambda = 0.8$ for Q_8 .

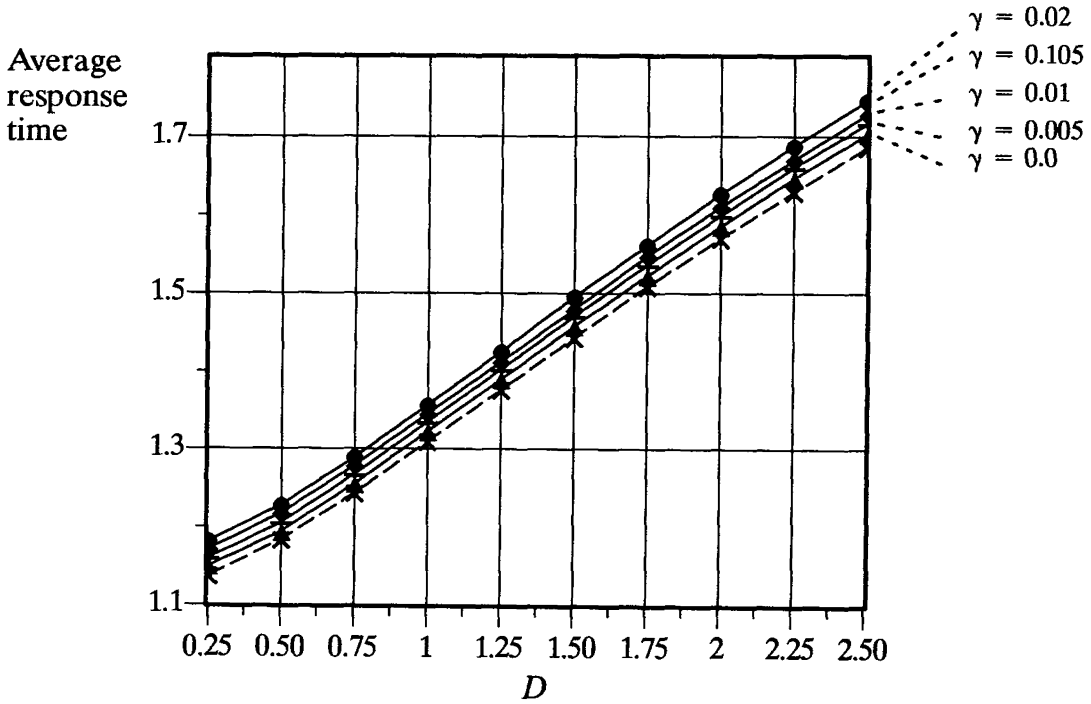

Figure 11. Average task response time versus average D for various γ's at $\lambda = 0.8$ for Q_8 .

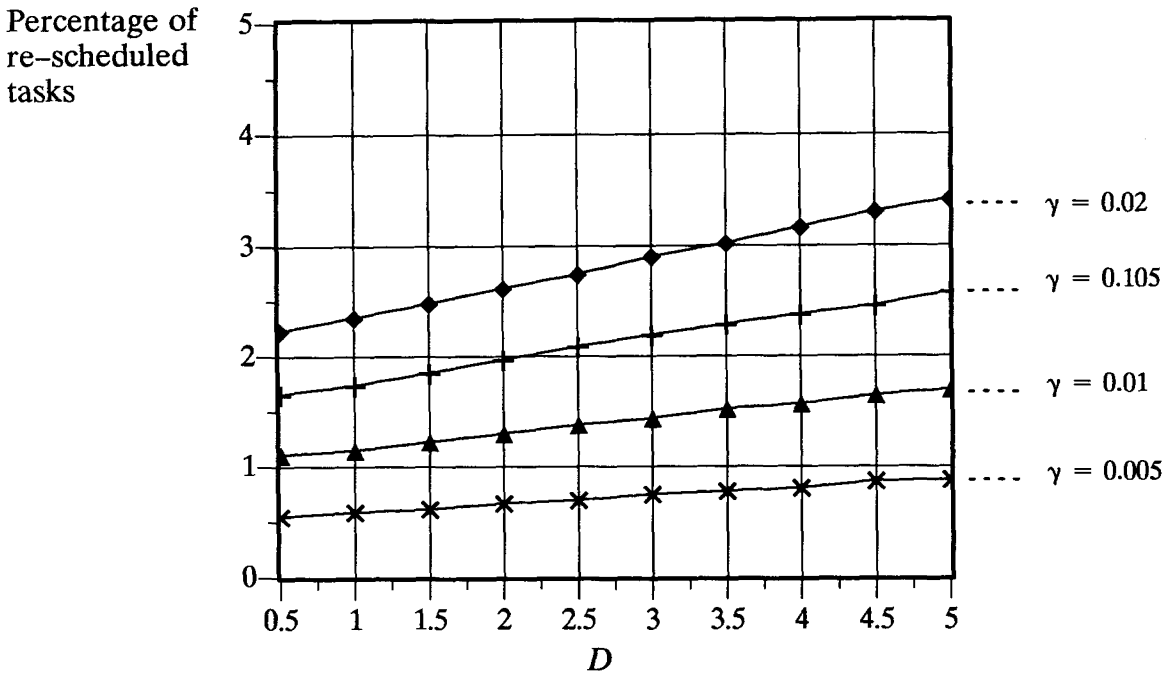

Figure 12. Percentage of re-scheduled tasks versus average D for network B_9 at $\lambda = 0.8$ and various γ's.

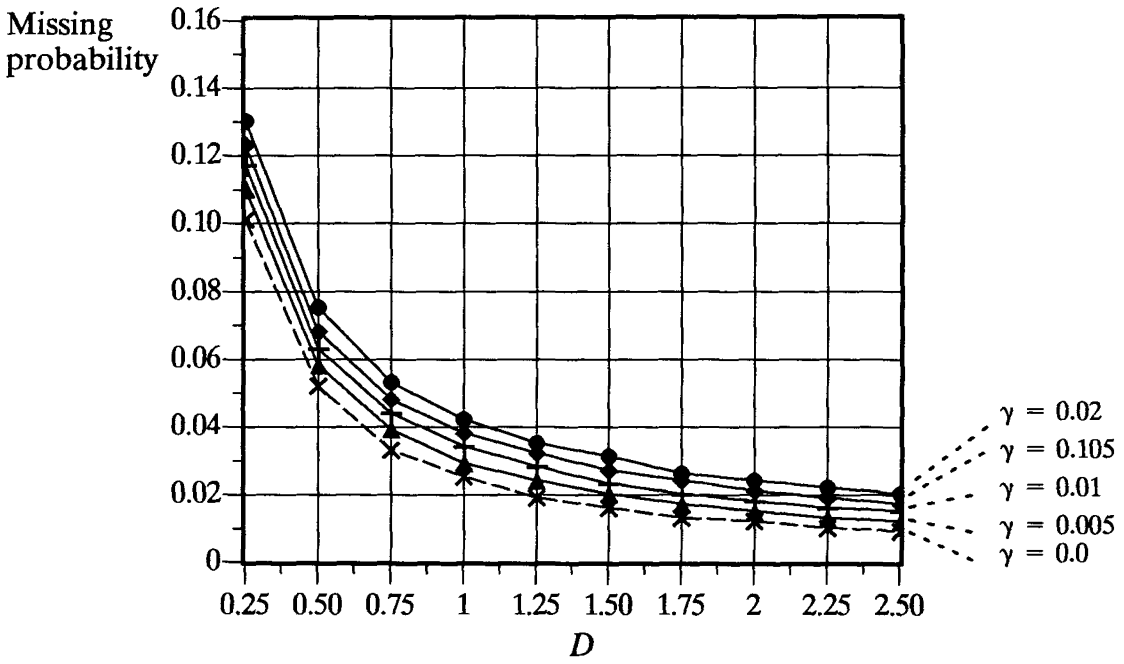

Figure 13. Deadline missing probability versus average D for various γ's at $\lambda = 0.8$ for B_9 .

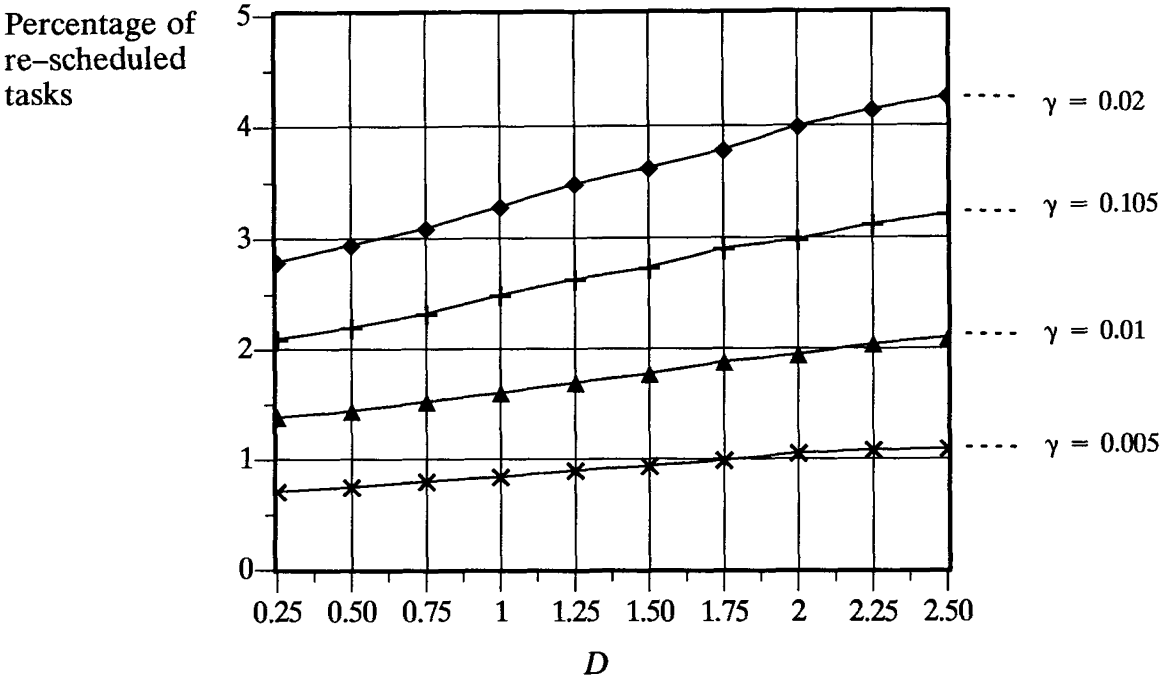

Figure 14. Percentage of re-scheduled tasks at various deadlines for O_6 network at $\lambda = 0.8$ and various γ's .

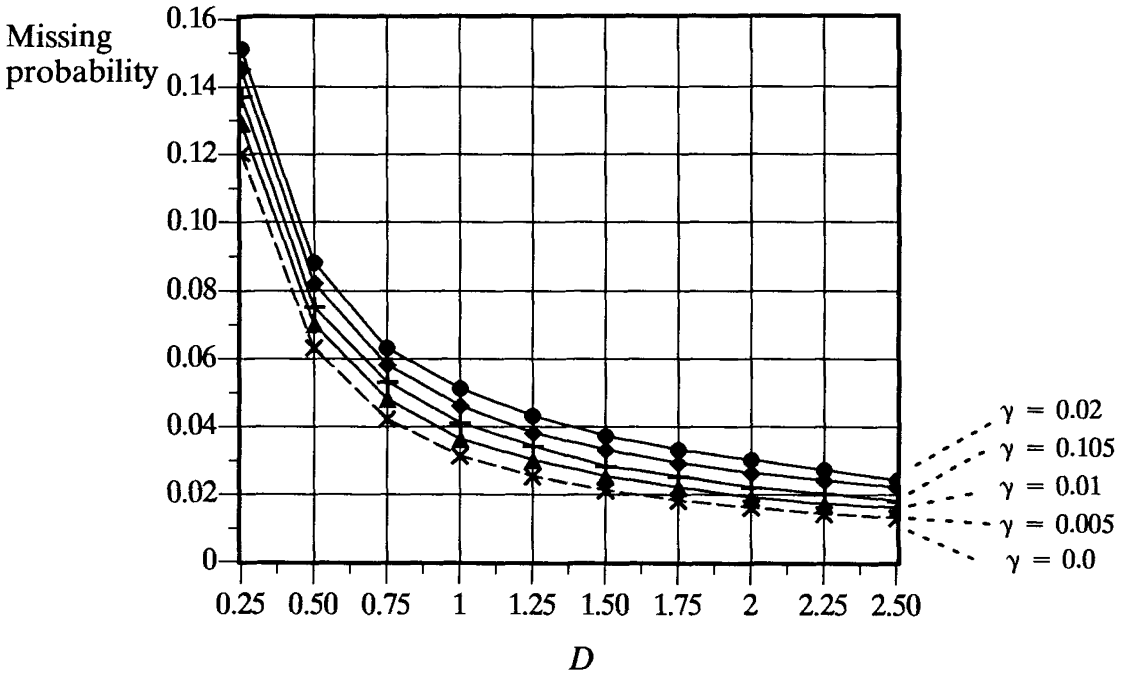

Figure 15. Deadline missing probability versus average D for various γ's at $\lambda = 0.8$ for O_6 .

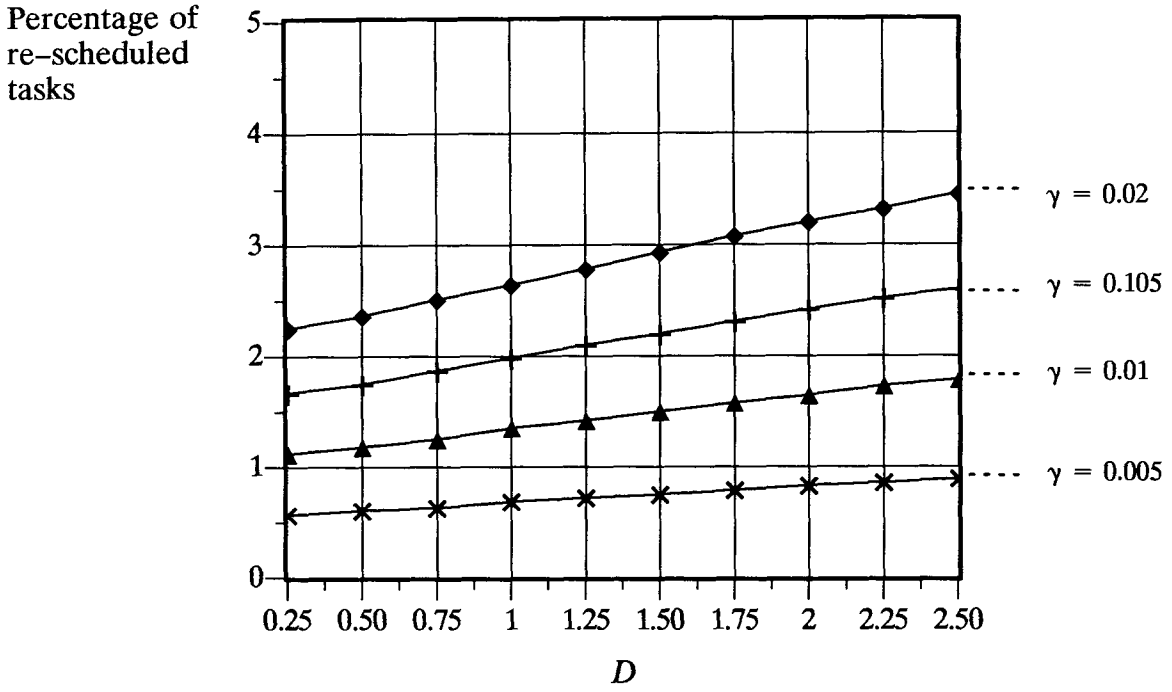

Figure 16. Percentage of re-scheduled tasks versus average D for network Γ_5 at $\lambda = 0.8$ and various γ'σ.

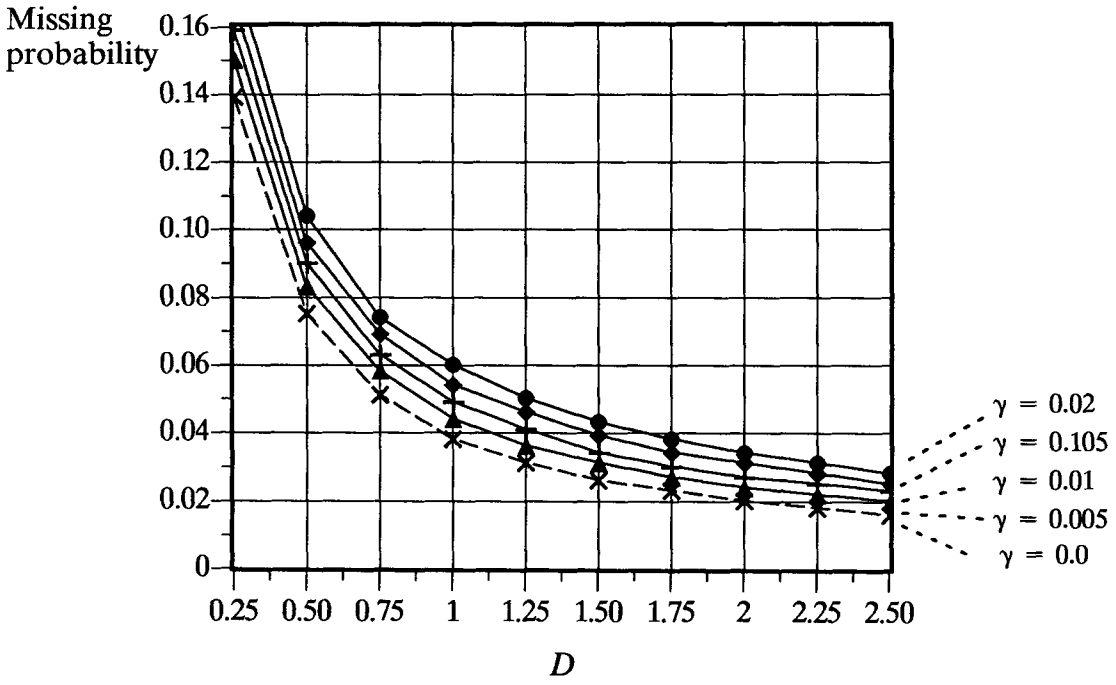

Figure 17. Deadline missing probability versus average D for various γ's at $\lambda = 0.8$ for Γ_5 .

nodes. However, B_9 is isomorphic to the folded Q_8 and thus has a degree more than one of that of Q_8. This results in better load sharing since the increased connectivity of communication links provides a better load view to the scheduler of a node. The increased connectivity is also useful for the semi-distributed based load re-distribution scheme. This is due the fact that, although the number of FCPs is also the same for both the networks, the sphere size for B_9 is 46 as compared to 37 for Q_8. The percentage of re-scheduled tasks is marginally less than that of Q_8, as expected, since the underlying decentralized algorithms performs better in B_9 resulting in slightly reduced execution and backup queues.

The percentage of re-scheduled tasks in O_6 is greater than the rest of the networks as is clear from Figure 14. This is because the size of O_6 is bigger which results in a greater accumulative node failure rate. The percentage of re-scheduled tasks for Γ_5 , as shown in Figure 16, is comparable with B_9 and Q_8. However, the missing probability of Γ_5 , for the no failure case, is worse than all the other three networks. The reason is this network has a smaller degree than Q_8 and B_9, although they have almost the same number of nodes. The effect of degree of O_6 on the missing probability places this network between Q_8 and Γ_5 .

Another reason for including these results is to show that the proposed fault-tolerant strategy is equally useful for the four different classes of network discussed in this paper. In summary the performance degradation due to failures is independent of network topology although the performance of underlying decentralized scheduling algorithm , to certain extent, may depend on the network topology. This can be noticed from the curves for missing probabilities for all of the four networks, which show different performance under no failures but the same performance degradation patterns. Although the response time curves for B_9, O_6 and Γ_5 are not shown here but the topology effect was observed.

5.2.3. Sensitivity to System Load and Network Communication Rate

Another important factor that can have a significant effect on normal load distribution as well as load re-distribution is the system load. The results presented in this section examine this factor by considering low, medium and high loading conditions. The missing probability and average task response time are obtained by varying λ from 0.6 to 0.9, for all networks. Both failure and no failure conditions are considered. Recall that throughout of this study μ_E is kept as 1 and therefore the system load ($\rho = \lambda / \mu$) corresponds to λ. For failures, γ is kept as 0.01. D and μ_C have been selected to be 0.5 and 10, respectively.

The curves for missing probability and average task response time are plotted in Figure 18 and 19. Both performance parameters, obviously, depend on the system load, under failure or no failure conditions. The important thing to notice from these curves is that under any load conditions the performance degradation (the missing probability and the average re-

sponse time) is slightly dependent on system load. We also notice that B_9 performs the best followed by Q_8, O_6 and Γ_5. The same performance ranking for these networks is retained with system under failure. The first reason is the effect of degree. The second reason is that the sphere size for load re-distribution for B_9 is large than that of Q_8 or Γ_5 which again helps in better load re-distribution. For O_6, although the sphere size is the biggest among these four graphs, its degree is lesser than B_9.

Finally, the effect of task transfer delays on the performance of the proposed strategy is also evaluated. It has been observed that for most of the systems, the task transfer delays can greatly contribute to the system response time and if these delays are significantly high, the performance with load sharing can be worse than no load sharing at all [28]. As mentioned earlier, in the proposed load re-distribution strategy, the task transfer delays from an FCP to one of its nodes in the system are also taken into account. The task transfer delay from an FCP to one of its nodes in the sphere depends on the distance of that node from the FCP. For re-scheduling algorithm, an FCP, however, first check its own load and then all the nodes at distance one followed by nodes at distance two and so on. If an FCP decides to re-schedule the task in its own local queue, no task transfer delay is incurred.

For λ equal to 0.6, missing probability and average response time are shown in Figure 20 and Figure 21, respectively. Again, both no-failure and failure conditions are considered for all four networks.

The average task communication rate is varied from 6 to 20 tasks/time-unit. Both the missing probability and average task response time, under no failure, severely reduced if the communication rate is below 10. The most important observation is that the proposed fault-tolerant strategy is not affected by low communication rate.

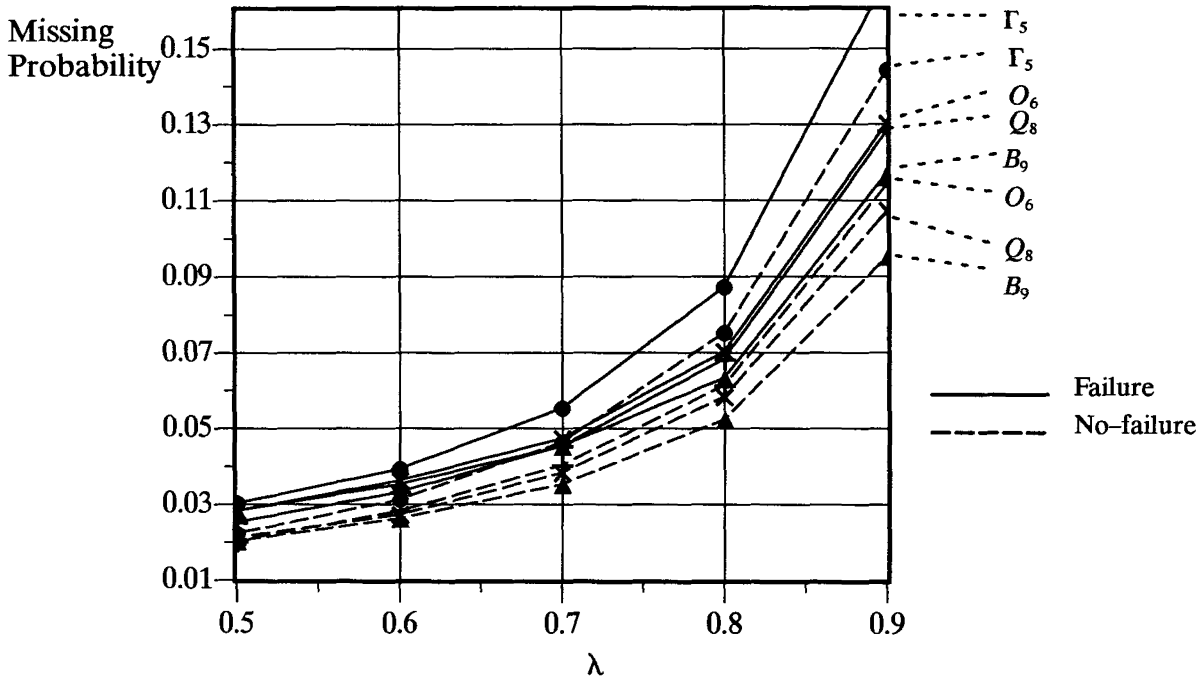

Figure 18. Missing probability versus λ for all four networks under failure and no failure conditions.

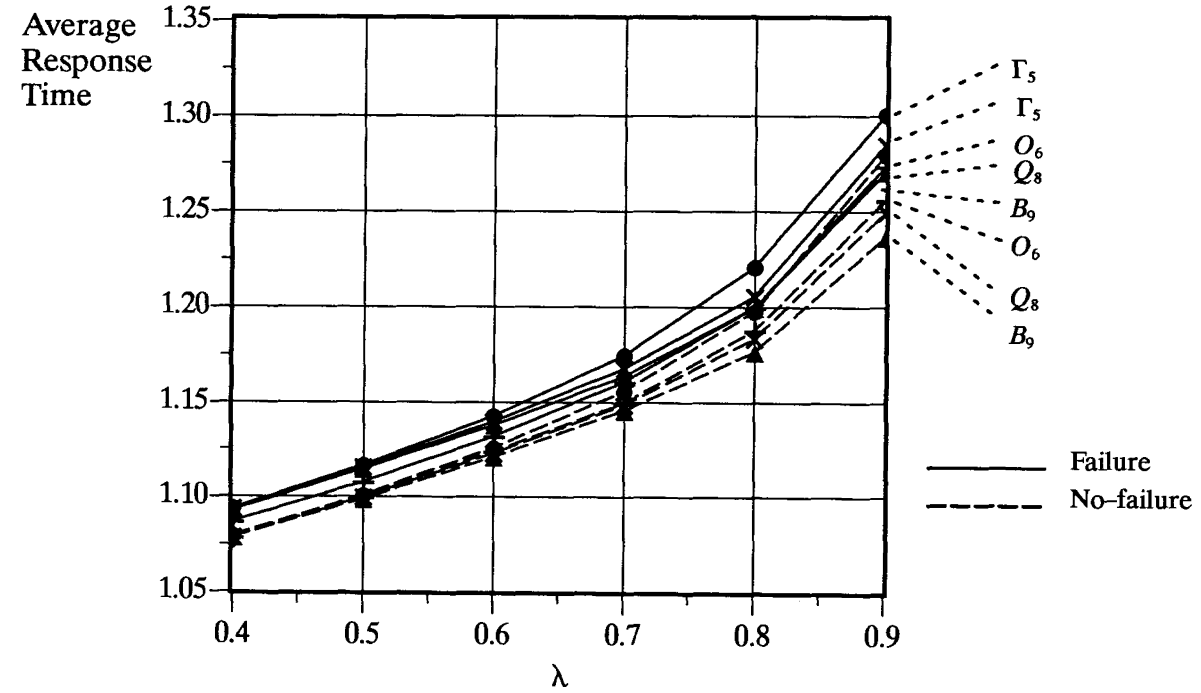

Figure 19. The average task response time versus λ for all four networks under failure and no failure conditions.

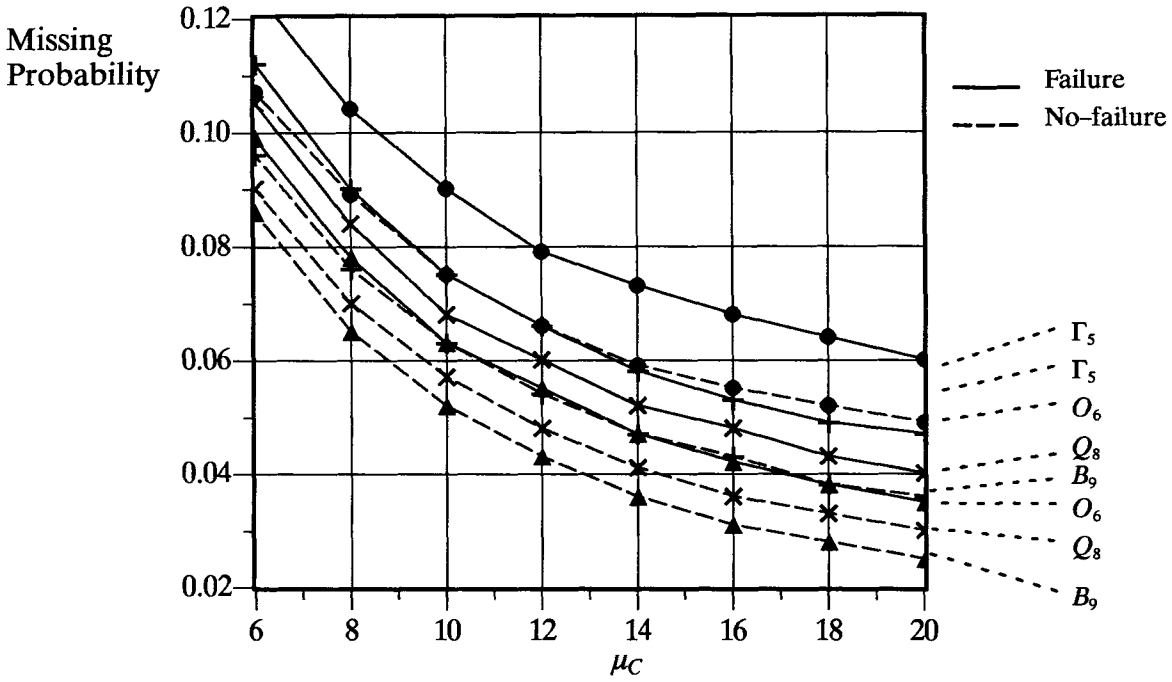

Figure 20. Missing probability versus μ_C for all four networks under failure and no failure conditions at $\lambda = 0.8$.

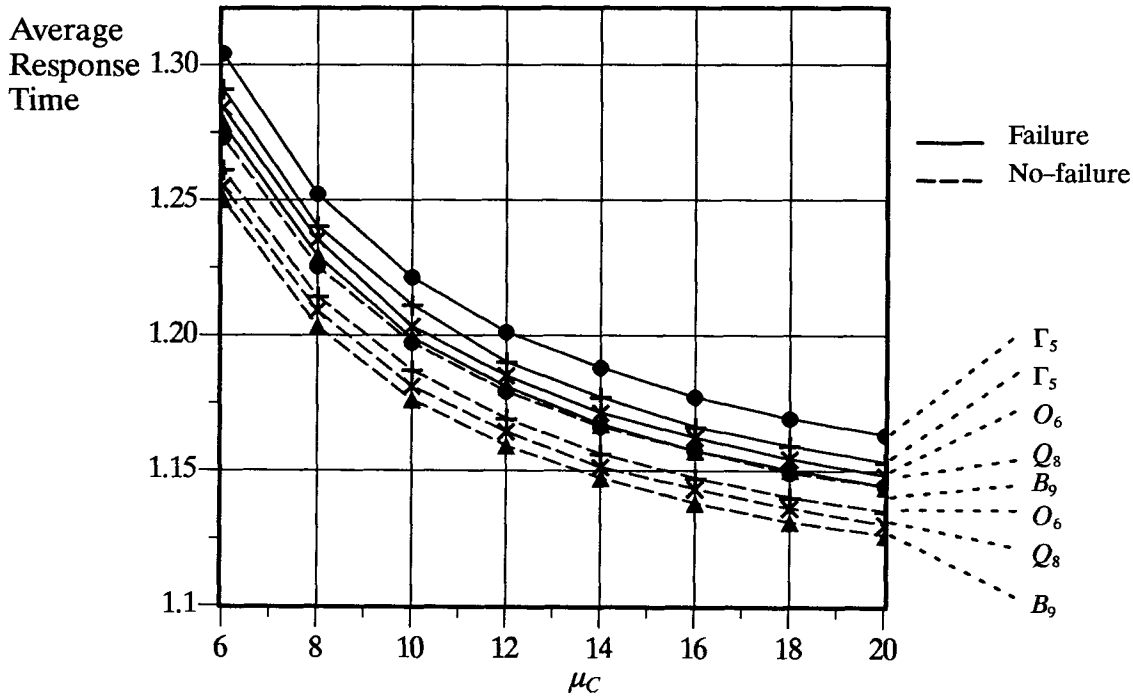

Figure 21. The average task response time versus μ_C for all four networks under failure and no failure conditions at $\lambda = 0.8$..

5.3 Measure of Goodness

So far the results presented have shown that the proposed strategy is insensitive to a number of system parameters such as D, λ and μ_C and the assignment of FCPs. This degradation in the overall missing probability and average response time performance is due to achieving our main objective of saving those tasks which would have been lost if there were no fault-tolerance and load re-distribution. The obvious question is how good is the strategy in achieving this objective. The usefulness of such a fault-tolerant strategy, therefore, should be judged by observing its ability to save the lost tasks. For this purpose, we define a measure of goodness for any fault-tolerant strategies for real-time systems, which is the percentage of tasks which met their deadline after being re-scheduled. Another measure of goodness is the amount of overhead involved for managing the fault-tolerant strategy.

For the first measure of goodness , we have observed the percentage of re-scheduled tasks which still meet their deadlines, after being re-scheduled. Three sets of these results are

Table IV Percentage of task which still meet their deadline after re-scheduling
$\gamma = 0.005$

Network	D \ λ	0.5	0.7	0.9
Q_8	0.5	39.34	34.66	27.43
	1.0	64.09	63.26	61.20
	1.5	72.54	71.53	71.21
	2.0	78.86	77.51	75.38
B_9	0.5	39.01	35.80	30.34
	1.0	69.74	65.81	64.88
	1.5	75.15	74.12	73.34
	2.0	80.57	79.25	78.41
O_6	0.5	42.74	36.23	28.19
	1.0	70.64	69.56	64.02
	1.5	76.58	75.36	74.12
	2.0	80.85	80.73	79.94
Γ_5	0.5	42.74	36.23	29.19
	1.0	62.72	60.07	58.49
	1.5	66.63	65.44	64.44
	2.0	77.33	75.92	72.81

presented in Tables IV, V and VI, for fault rates 0.005, 0.01 and 0.02, respectively. Since the normal system performance under no–failure has shown its dependence on deadline and system load, we have considered four different values of D, which are 0.5, 0.1, 1.5 and 2.0, with three different values, 0.5, 0.7 and 0.9, for λ.

This measure greatly depends on the value of D. In addition, we notice that λ has a little effect and even for λ equal to 0.9, the percentage of tasks still meeting their deadlines is only slightly lower than for the case when l is equal to 0.5. This is due to the remarkable semi–distributed load re–distribution mechanism which allows an FCP to find a lightly loaded node in its sphere, quickly. A more closer look at all three tables reveals that fault rate has some impact. Even for large failure rate such as 0.02, more than 70% of the tasks still meet their deadlines .

An important phenomena that is intuitively true and has also been observed is that as the sphere size for an FCP increases, the number of re–scheduled tasks, which meet their deadline, improves. For example B_9, Q_8 and Γ_5 are networks with almost equal size and with

Table V Percentage of task which still meet their deadline after re–scheduling
$\gamma = 0.015$

Network	D \ λ	0.5	0.7	0.9
Q_8	0.5	37.17	34.51	30.55
	1.0	63.54	62.92	62.56
	1.5	71.32	70.90	68.43
	2.0	78.00	77.11	75.03
B_9	0.5	38.42	34.02	30.53
	1.0	66.37	63.87	63.17
	1.5	74.86	73.25	67.60
	2.0	80.69	79.72	78.10
O_6	0.5	35.14	32.58	28.46
	1.0	65.62	63.17	63.60
	1.5	74.11	73.68	69.70
	2.0	80.01	78.43	76.13
Γ_5	0.5	40.57	36.77	29.65
	1.0	62.01	61.75	56.67
	1.5	70.20	68.80	62.26
	2.0	78.92	74.74	73.16

Table VI Percentage of tasks which still meet their deadline after re-scheduling
$\gamma = 0.02$

Network	D \ λ	0.5	0.7	0.9
Q_8	0.5	36.04	31.92	30.03
	1.0	62.24	62.56	59.58
	1.5	68.31	68.15	67.77
	2.0	77.95	76.55	72.56
B_9	0.5	38.23	32.46	31.98
	1.0	65.26	62.68	62.31
	1.5	72.66	71.22	69.34
	2.0	79.51	78.48	74.30
O_6	0.5	36.24	33.66	29.89
	1.0	59.48	56.15	53.85
	1.5	71.88	70.37	65.33
	2.0	75.44	75.76	71.54
Γ_5	0.5	32.76	29.30	28.49
	1.0	58.52	58.37	55.38
	1.5	69.58	69.23	64.77
	2.0	74.32	73.22	67.26

Table VII Percentage of tasks which are lost due to the failure both FCPs
($D = 0.5$)

Network	λ \ γ	0.005	0.01	0.02
Q_8	0.6	0.00	3.75×10^{-3}	7.5×10^{-3}
	0.8	0.00	6.25×10^{-3}	16.5×10^{-3}
B_9	0.6	0.00	8.75×10^{-3}	10.0×10^{-3}
	0.8	0.00	15.0×10^{-3}	17.5×10^{-3}
O_6	0.6	0.00	5.0×10^{-3}	10.0×10^{-3}
	0.8	0.00	6.25×10^{-3}	22.5×10^{-3}
Γ_5	0.6	0.00	8.75×10^{-3}	10.0×10^{-3}
	0.8	0.00	11.25×10^{-3}	13.75×10^{-3}

almost same number of FCPs. However, the size of a sphere in B_9, Q_8 and Γ_5 is 46, 37 and 26 respectively. According to the results, we note that the percentage of tasks meeting their deadlines after re-scheduling is the maximum for B_9 followed by Q_8 whereas for Γ_5, this percentage is the lowest. Hence, based on these observation, it is our believe that given two networks of the same size and same number of FCPs, the one with larger sphere will perform better than the other, for our proposed strategy.

Also if the network size increases and the number of FCP do not increase proportionally, the sphere size per FCP also increases, thereby increasing the performance. This is the case of O_6. As can be noticed from the proposed partitioning scheme (section 3), the number of FCPs are always of $\Theta(log\ N)$. Therefore, as the network size increases, the sphere size per FCP increases. Accordingly, the scheme proposed in this paper is expected to perform better for large systems.

Table VII provides information about the percentage of tasks which could not be executed at all due to the failure of the node and both the primary and the secondary FCPs. In other words, the entries in the table provide the probability of task being lost altogether. Generally, this probability depends on two factors. First, the level of redundancy which is two in this case and the number of FCPs among which the tasks are distributed in a round-robin fashion. The values shown in Table VII are negligible.

5.4. Analysis of Message Overhead for Fault Management

For the second measure of goodness, we analyzed the proposed fault-tolerant strategy in terms of the average overhead incurred. This overhead provides an estimate for the number of messages needed for communication between a node and the set C per task.This overhead directly depends upon the average distance of a node in the network to all its primary and secondary FCPs. Let α be the such average graphical distance of a node y from the set C. That is:

$$\alpha = \frac{1}{|C|} \sum_{x \in C} L_{xy} \qquad (1)$$

We show that α is independent of the choice of node y. This is an important consideration, in the sense that the proposed use of Hadamard matrix to select C provides a uniform and homogeneous access to all the nodes in the network. However, it is obvious that from a node to all the members of the set C, there are variations in distances. We, therefore, are also interested in the standard deviation σ of these distances, since it provides a measure of variations in the overhead.

The values of α and σ can be calculated by generalizing the concept of *q–covering* [6] for *DT*–networks. The *q–covering* is defined as follows. We consider a mapping q of the non–negative integers into themselves with the property that $q(x) = 0$ implies $q(y) = 0$ for all $y > x$. A set C in a graph Λ is a *q–covering* if the average of $q(L_{sx})$ for an arbitrary x in the vertex set of Λ over all $s \in C$ is independent of x. The basic idea in [6] is that t–designs in the usual sense are *q–coverings* in the Johnson Association Scheme, with $q(i) = \binom{q-i}{t}$ being a polynomial of degree at most t. The following theorem gives the values for α and σ. The proof of this theorem is given in the appendix.

Theorem 4: For antipodal networks $\alpha = k/2$, and for podal networks $\alpha \leq k/2$. For *DT* networks,

$$\sigma \leq \sqrt{\frac{4k^6 - 6k + 1}{4(2k-1)}} \qquad (2)$$

Proof: See Appendix

It is interesting to note that the average distance, and hence the average overhead for the proposed scheme in *DT* networks is always equal to or less than the radius of the network. Also, we can notice that the variations in the distances, and hence the overhead is $\Theta(\sqrt{n})$, which is considerably small as compared to the average value of α for large networks.

6. Conclusions

In this paper, we have proposed a new fault–tolerant approach for large–scale multicomputer systems. The proposed strategy is applicable to any decentralized load balancing algorithms used in soft real–time distributed systems. The study was centered around a class of interconnection structures which are distance–transitive. The use of Hadamard matrix results in an efficient strategy for identifying central points and for partitioning these systems for load re–distribution. The central points, called fault–control points, provide a two–level task redundancy and efficiently re–distribute the load of failed nodes within their spheres. The partitioning strategy results in a small number of spheres which remain of $\Theta(\log N)$.

For failure and repair processes, we have assumed a realistic failure–repair system environment. In addition, we have not ignored the load submitted to nodes while they are under repair, due to real–time constraints. Through an extensive simulation, the performance of the proposed strategy has been evaluated for both failure and no–failure cases. The degradation in the overall performance is due to achieving the objective of saving those tasks which would have been lost if there were no fault–tolerance and load re–distribution. To evaluate the pro-

posed strategy under this objective, we have define a measure of goodness namely the percentage of tasks which met their deadline despite the failure of their original nodes. It is shown that using the proposed strategy, up to 80 % of the tasks can still meet their deadlines. The second measure of goodness is the amount of overhead involved for managing the fault-tolerant strategy. We have also analyzed the sensitivity of this strategy with respect to various system parameters and have shown that the performance degradation due to failures does not greatly depend on these parameter. The probability of a task being lost altogether due to multiple failures has also been shown to be negligible.

Appendix

Proof of Theorem 1: (a) For the proof, see [37] .

(b) Finding the minimum sphere size, $|S_f(x)| \forall x \in C$, requires us to determine the minimum value of f, which is equal to the covering radius of the set C. Since all the above topologies are represented using binary codes, the problem of determining the set C is equivalent to finding a sub-code with the desired covering radius in a code, say F. For Q_n, F is the complete binary code. For a bisectional network, B_n, F represents all the codewords of length n with even weights whereas for an odd graph, O_n, F represents a constant weight code of weight n with length $n-1$. However finding the covering radius of a sub code, say C, in a code F is an NP-hard problem [26]. Since finding the minimum sphere size requires determining the covering radius, the complexity of the whole problem will not be less than NP-hard. Q.E.D.

Proof of Theorem 2: A SBIBD is called 2-design, which is a collection of distinct m-subsets (called blocks) of v-set (that is a set of v-elements), with the property that any 2-subset of v-set is contained in exactly θ blocks. This collection is denoted as 2-(v,m,θ). Most of the Hadamard matrices are incidence matrix of a 2-design, where an incidence matrix defines the relation between blocks an element. For example, the truncated M of Figure 1 is the incidence matrix of a 2-(7,4,2) design. A SBIBD is generated using the so called finite difference set approach [21]. A (v,m,θ) difference set $T_0 = (q_0, q_1, \ldots q_{m-1})$ is a collection of m residues modulo v, such that for any residue a, ($a > 0 \ mod \ v$), the congruence

$$q_i - q_j = a \mod v \qquad (3)$$

has exactly θ solution pairs (q_i, q_j). T_0 serves as one of the blocks. Rest of the blocks of SBIBD are generated by adding a constant ($mod\ v$) t o the elements of T_0. Given a SBIBD with parameters $v=2i-1$, $m=i$, $\theta=i/2$, for a given i, by taking the complement of its blocks, we generate another SBIBD which has the following parameters [21]:

$$v^*=v=2i-1,\ m^*=v-m=i-1,\ \theta^*=v-2m+\theta=\frac{i-2}{2}$$

Let the complement of T_0 be T_0^+. Clearly if $0 \in T_0$, then $0 \notin T_0^+$. Next, let us include 0 in T_0^+. Such an inclusion does satisfy the congruence in equation (3), and in fact the number of (q_i, q_j) pairs increases to $\theta^* + 1$. We can now include cyclically incremented values of 0 (i.e. 1,2, ...) in the rest of the complemented blocks. Note, that the resulting parameters of this augmented complemented SBIBD are:

$$v^* = 2i-1,\ m^* = (i-1)+1 = i,\ \theta^* = \frac{i-2}{2} + 1 = \frac{i}{2}\ ,$$

which are the same as the original SBIBD. We call this newly generated SBIBD as "augmented" SBIBD (ASBIBD). Note, that there exist pair of blocks in T_0 and T_0^+ which have only one element in common. Therefore, each block of SBIBD has exactly one block in ASBIBD, with which it has only one element in common. The proof is clear by noting that the matrices M and M_s are the incidence matrices of SBIBD and ASBIBD, respectively. Q.E.D.

Proof of Theorem 4: For antipodal networks the value of α is obvious, since C consists of both matrix M and its complement and is a self-complementary 2-design. We know that a bisectional graph is isomorphic to a folded Hypercube. If we just take the set C in Q_n, a little thought can reveal that the complementary matrix in Q_n can serve as the translated matrix in B_n. Therefore, the the average distance of Q_n is indeed the upper bound for the average distance in B_n. It can be noticed that O_n can be embedded in B_n. Also, we can prove that the shortest graphical distances between any two nodes in O_n remain within O_n even when in it is embedded in B_n. Therefore, we can use the value for B_n as an upper bound for O_n

For the standard deviation we first consider Γ_n network. Let β be the second moment of Hamming distances in Γ_n. Note that β is also the second moment for the graphical distances in Γ_n, since $L_{xy} = H_{xy}$ for this network. Since,

$$\beta = \frac{1}{|C|} \sum_{x \in C} L_{xy}^2 \qquad (4)$$

$$\beta = \frac{1}{|C|} \sum_{x \in M} \left[H_{xy}^2 + (2n - 1 - H_{xy})^2 \right] \qquad (5)$$

and $|M| = |C|/2$, we must have:

$$\beta = \frac{2}{|C|} \sum_{x \in M} H_{xy}^2 - \frac{2(2n-1)}{|C|} \sum_{x \in M} H_{xy} + \frac{(2k-1)^2}{2} \qquad (6)$$

In order to solve this expression, we start with the Johnson graph [5], with length of the codeword $v = 2n-1$ and weight n. For this graph it is known that:

$$2L_{xy} = H_{xy} = |x| + |y| - 2|x \cap y|.$$

where $|x|$ represents the weight of codeword x and $|x \cap y|$ represents the number of bits which are common between codewords x and y. Since, weight of all the codewords is n, we get the following, if we select the matrix M.

$$\sum_{x \in M} |x \cap y| = \sum_{x \in M} (n - L_{xy}) \qquad (7)$$

Using the concept of *q–covering* [6], this leads to the following expression:

$$\sum_{x \in M} |x \cap y| = \pi \binom{n}{1} = \pi n = |M|(n - \varrho) \qquad (8)$$

where π represents the number of times a 2–set appears as a subset of blocks in M [21], and ρ is the average distance in Johnson graph. Using the incidence relation of a 2–design [21] ($|M|n = v\pi$), for $v = 2n-1$ and replacing n by $n-1$, we get $\rho = n\,(1 - n/v)$. Similarly, we can find the second moment (ϕ) in Johnson graph with respect to M. Again, starting with the *q–covering* of 2–design M, we get the following expression:

$$\sum_{x \in M} \binom{|x \cap y|}{2} = \pi \binom{n}{2} = \sum_{L_{xy}} \frac{(n - L_{xy})(n - 1 - L_{xy})}{2} \qquad (10)$$

Using the value of ρ, and the incidence relationship of a 2–design we get:

$$\phi = \frac{n^2(n-1)}{2(2n-1)} \qquad (11)$$

Coming back to Γ_n network, we can now notice from equation (6) that the second moment in Γ_n can be given as $\beta = 4\phi + 1/2$. This leads to the desired expression for σ.

The above results can now be extended to other graphs. First consider the binary Hypercube network, Q_n. Since, Γ_n, is a sub–graph of a Hypercube, and the relation between the graphical distances and the Hamming distance is same for both, the equation (7) if holds for Γ_n, will also hold for the Hypercube. This equation is the basis for the whole analysis. So the results for Γ_n is equally applicable to the Hypercube, in which it is embedded.

Since a bisectional network is isomorphic to a folded Hypercube, the folding does not increase distances or their variations, rather it may decrease these parameters. Therefore, the above results serve as upper bounds for bisectional networks.

By looking at the code structure of an odd graph, it can be noticed that O_n can be embedded in B_{2n-1}, with matrices in C present at common nodes in both the graphs (except all 0s of B_{2n-1}). Therefore, the bounds continue to hold for odd graphs. Q.E.D.

Lemma 3: The covering radius r in DT graphs is bounded as follows:

$$r \leq \alpha - \sigma$$

Proof: Proof can be obtained by using a probabilistic argument similar to the second Norse bound [22].

References

[1] I. Ahmad and A. Ghafoor, " A Semi-Distributed Distributed task Allocation Strategy for large Hypercube Supercomputers " Proc. of *Supercomputing '90*, New York, Nov. 1990, pp 898-907.

[2] W. C. Athas and C. L. Seitz, "Multicomputers: Message-Passing Concurrent Computers," *IEEE Computer*, August 1988, pp. 9-24.

[3] B.W. Arden and H. Lee, "Analysis of Chordal Ring Network", *IEEE Trans. on Computers* Vol. C-30(4), April 1981. pp. 291-295.

[4] J.R. Armstrong and F.G. Gray, "Fault diagnosis in a Boolean n-cube array of microprocessors," *IEEE Trans. on Computers*, Vol. C-30 (8), August 198, pp. 581-590.

[5] E. Bannai and T. Ito, *Algebraic Combinatorics and association schemes.* Benjamin-Cummings 1984.

[6] N.L. Biggs, "Designs, Factors and Codes in Graphs", *Quart. J. of Math.* Oxford (2), 26, 1975, pp. 113-119.

[7] R. M. Bryant and R. A. Finkel, "A Stable Distributed Scheduling Algorithm," in *Proc. of 2nd Int'l. Conf. on Distributed Computing Systems,* 1981, pp. 314-323.

[8] Y. Chang and K. G. Shin, "Load Sharing in Hypercube Multicomputers in the Presence of Node Failures," in Proc. of *Fifth Distributed Memory Computing Conference, Vol. II,* April 1990, pp. pp. 1465-1474.

[9] T. C. K. Chou and J. A. Abraham, "Load ReDistribution Under Failure in Distributed Systems", *IEEE Trans. on Computers*, vol. C-32, no. 9, Sept. 1983, pp. 799-808.

[10] Y.-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, May, 1982, pp. 667-679.

[11] E. Clement, D. Logan and V. Sonnad, 'Solution of Large Scale Engineering Problems using Loosely Coupled Array of Processors ", *Numerical Methods for Modern Parallel Computer Architectures*, The IMA Volumes in Mathematics and its Applications, vol. 13, Springer-Verlag, 1988, pp. 11-27.

[12] G. Cohen, M.G. Karpovsky, H.F. Mattson, J. R. Schatz, "Covering Radius-Survey and Recent Results", *IEEE Trans. on Information Theory*, IT-31 (3), May 1985, pp.328-343.

[13] C. R. Das, J. T. Kreulen and M. J. Thazhuthaveetil, "Dependability Modeling for Multiprocessors", *IEEE Computer*, Oct. 1990, pp. 7–19.

[14] D. L. Eager, E. D. Lazowska and J. Zahorjan,"Adaptive Load Sharing in Homogeneous Distributed Systems," *IEEE Trans. on Software Eng.* ,vol. SE–12, May 1986, pp. 662–675.

[15] K. Efe and B. Groselj, "Minimizing Control Overhead in Adaptive Load Sharing," in *Proc. of 9–th Intl. Conf. on Distributed Computing Systems*, 1989, pp. 307–315

[16] D. Ferguson, Y. Yemini and C. Nickolaou "Microeconomic Algorithms for Load Balancing in Distributed Computer Systems, " in *Proc. of 8–th Int'l. Conf. on Distributed Computing Systems*, 1988, pp. 491–499

[17] A. Ghafoor and T. R. Bashkow, "A Study of Odd Graphs as Fault–Tolerant Interconnection Networks," *IEEE Trans. on Computers,* vol. 40, no. 2, February 1991, pp. 225–232.

[18] A. Ghafoor, T. R. Bashkow and Imran Ghafoor, "Bisectional Fault–Tolerant Communication Architecture for Supercomputer Systems," *IEEE Trans. on Computers,* vol. 38, no. 10, October 1989, pp. 1425–1446.

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

[20] A. Ghafoor, S.A. Sheikh, and P. Sole, "A Bipartite Distance–Regular Topology for Fault–Tolerant Multiprocessor Systems", Proc. of IEE (Part E), May 1990.

[21] M. Hall Jr., *Combinatorial Theory*, 2nd Ed., John Wiley and Sons, New York, 1986.1

[22] T. Helleseth, T. Klove, J. Mykkelveit, "On the covering radius of binary codes", *IEEE Trans. on Information Theory*, Vol. IT–24 (5), September 1978, pp. 627–628.

[23] M. Livny and M. Melman, "Load Balancing in Homogeneous Broadcast Distributed Systems," in *Proc. of ACM Computer Network Performance Symposium,* April 1982, pp. 47–55.

[24] J. F. Kurose and R. Chipalkatti, "Load Sharing in Soft Real–Time Distributed Computer Systems," *IEEE Trans. on Computers,* vol. C–36, no. 8, August 1987, pp. 993–1000.

[25] D. W. Leinbaugh and M. Yamini, "Guaranteed Response Times in a Distributed Hard–Real–Time Environment," *IEEE Trans. on Software Eng.,* vol. SE–12, Dec. 1986, pp. 1139–1144.

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

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

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

[29] J. K. Muppala, S. P. Woolet and K. S. Trivedi, "Real-Time-Systems Performance in the Presence of Failures, " *IEEE Computer*, May 1991, pp. 37–47.

[30] L. M. Ni, C. Xu and T.B. Gendreau, "A distributed Drafting Algorithm for Load Balancing," *IEEE Trans. on Software Eng.*, vol. SE–11, no. 10, October 1985, pp. pp. 1153–1161.

[31] K. Ramamritham, J. A. Stankovic and W. Zhao, "Distributed Scheduling of Tasks with Deadlines and Resource Requirements," *IEEE Trans. on Computers*, vol. 38, no. 8, Aug. 1989, pp. 1110–1123.

[32] K. G. Shin, "Introduction to the Special Issue on Real-Time Systems," *IEEE Trans. on Computers*, vol. 36, no. 8, Aug. 1987, pp. 901–902.

[33] K. 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, Aug. 1989, pp. 1124–1142.

[34] N. G. Shivrati and M. Singhal, "A Transfer Policy for Global Scheduling Algorithms to Schedule tasks with Deadlines, " in Proc. of *11–th Int'l. Conf. on Distributed Computing Systems*, May 1991, pp. 248–255.

[35] J. A. Stankovic,, "Decentralized Decision Making for Task Allocation in a Hard Real-Time System," *IEEE Trans. on Computers*, vol. 38, no. 3, March 1989, pp. 341–355.

[36] J. A. Stankovic and I. S. Sidhu, "An Adaptive Bidding Algorithm for Processes, Clusters and Distributed Groups, "in *Proc. of 4–th Int'l. Conf. on Distributed Computing Systems*, 1984, pp. 49–59.

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

[38] Y-T Wang and R.J.T. Morris, "Load Sharing in Distributed Systems," *IEEE Trans. on Computers*, vol. C–34, March 1985, pp. 204–217.