FAULT TOLERANCE AND RELIABLE ROUTING IN AUGMENTED HYPERCUBE ARCHITECTURES

M. Sultan Alam *Rami Melhem*

Dept. of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260

ABSTRACT

Two schemes for augmenting hypercubes with spare nodes are presented. In both schemes, fault coverage is local in the sense that only the neighbors of the faulty nodes need to know about the faults. Routing in the presence of faults is accomplished by distributed algorithms that deliver messages reliably to their destination nodes. A message destined to a faulty node will be delivered to the spare node that replaces the faulty node. Both node and link failures may be accommodated in both schemes, and because no switches are used for reconfiguration, there is no assumptions about fault-free switches.

The two schemes differ in the complexity of the interconnections between the spare nodes and the original cube nodes. More complex interconnections results in more flexible coverage algorithms and thus, more reliable systems. However, increased flexibility increases the complexity of the routing algorithm because more care has to be given to the possibility of messages looping in cycles, thus never reaching their destinations. The reliability of the fault coverage schemes and the efficiency of the routing algorithms are evaluated from simulation results.

1. INTRODUCTION

Hypercube multiprocessor systems are being used for a variety of scientific applications and dedicated real time systems where error free computations are expected. However, as the number of processors in a hypercube system increases, the complexity of the system increases, leading to a possible high rate of both transient and permanent failures. For applications where degraded performance is not allowed it is necessary to reassign the tasks of the failing processors to some spare processors without destroying the logical hypercube interconnection among the functioning processors.

One approach to achieve fault tolerance where degraded performance is not allowed is to initially designate only some of the processors as active and designate the rest as spares that may cover for faulty active processors. Such approach is only useful for applications that (1) require a number of processors less than the number of processors in the available hypercube and (2) exhibit an interconnection structures that may be embedded in hypercubes [7]. Another approach to achieve fault tolerance is to decompose the hypercube structure hierarchically and add redundancy at several levels [2, 8, 9]. This approach requires a global reconfiguration algorithm in which a global controller reconfigures a set of cross-bar switches. It also does not take full advantage of the available hardware because a given module at a specific level may be replaced by a spare module even when most of its components are functioning properly.

In this paper we suggest a different approach in which an N node hypercube is augmented with E extra nodes that are to be used as spares. From the application point of view, only N nodes are used and messages between them are interchanged by using the usual log$N$ hypercube addresses. When a node fails, one of the spare nodes takes over its task and inherits its address. From this point on, messages addressed to the failed node should be routed to the spare node that replaces it. Hence, the suggested fault tolerance approach consists of two components: a node replacement policy, and a routing algorithm. Our basic goal is to ensure that both components are executed distributively without the intervention of a central controller.

Two schemes are proposed in Sections 2 and 3 for achieving this goal. In both schemes, the replacement policy is extremely local in the sense that only the neighbors of a faulty node need to know of the fault and take some action in a manner transparent to the application program. Also, routing is completely distributed and may not lead to a live-lock situation in which a message circulates in a cycle, and thus never reaches its destination. The routing algorithms reduce to the bit-wise routing algorithm for hypercubes [5, 6] when no faulty nodes are encountered, and adapts to faults gracefully. No switches are used and link failures can be accommodated.

In the first scheme, one spare node is added to each 2^i original nodes, for some integer i, $0<i\leq n$. The 2^i original nodes and the spare node form a module which we call a fault tolerant basic block (FTBB). An n dimensional hypercube may be built using 2^{n-i} FTBB's. Although this scheme is simple and results in an efficient routing algorithm, it has very little flexibility for fault coverage. Specifically, more than than one failure in an FTBB results in a system failure.

In order to improve the system reliability, the second scheme is introduced in which spare nodes may be shared. In this scheme an FTBB is a 3-dimensional hypercube with four spares. The spares are connected to the original nodes such that each node may be covered by two spares, and each spare can cover four nodes. This flexibility improves the reliability considerably at the expense of a more complex routing algorithm. The reliability and the routing performance of the two schemes are analyzed in Section 4 using simulation results.

2. SCHEME I: A Modular Fault Tolerant Architecture

In this scheme, the ensemble architecture is composed of 2^n original nodes and 2^{n-i} spare nodes. The 2^n original nodes are interconnected in a hypercube structure of dimension n, called the *original cube*. As is the case in any hypercube architecture, each node in the *original cube* has an n bit address. The 2^{n-i} spare nodes are also interconnected in a hypercube structure of dimension $n-i$, which we call the *spare cube*. In order to describe the interconnection between the spare and the original nodes, we consider a decomposition of the *original cube* into 2^{n-i} disjoint subcubes, for some i, $0<i\leq n$. Each subcube consists of 2^i nodes which have identical $n-i$ most significant address bits. We then allocate one spare node to each subcube and connect this spare to all the nodes in that subcube. A subcube and its spare form a module that we call a Fault Tolerant Basic Block (FTBB) which is identified by the $n-i$ most significant bits in the addresses of its original nodes. For example if $n=5$ and $i=2$, then there is a spare node for every four nodes in the *original cube*. Each four nodes which have the same 3 most significant address bits are connected to a spare node

from the *spare cube*. The augmented hypercube architecture is shown in Fig 1.

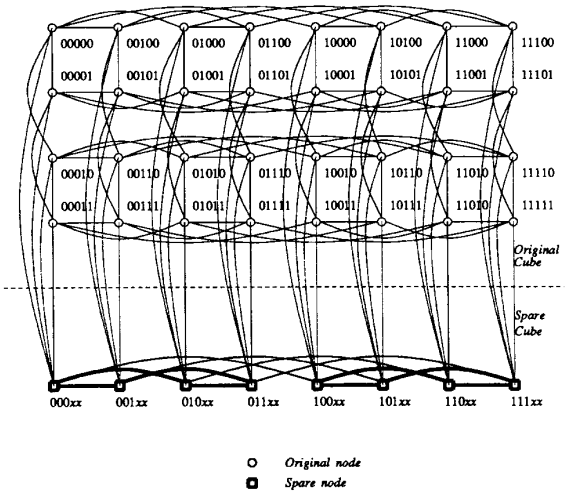

Fig 1 - The augmented hypercube architecture in SCHEME 1

Initially, each spare node has a generic address of the form $y_n...y_{i+1}X...X$, where the X bits are generic bits and $y_n...y_{i+1}$ is the address of the FTTB which contains that spare. When a node $y_n \cdots y_1$ in the *original cube* fails then the spare node $y_n...y_{i+1}X...X$ in the same FTBB replaces it and inherits its address. Since each FTBB contains only one spare, more than one node failure in the same FTBB will cause a system failure. Hence, the system in this scheme is single fault tolerant in the sense that the failure of any two nodes in the same FTBB will lead to the failure of the system.

The bit-wise routing algorithm usually used in hypercube architectures [5,6] needs to be modified to route messages around faulty nodes. Specifically, a message addressed to some node $y_n \cdots y_1$ should be routed to the original node having this address, or, if the original node is faulty, to the spare node $y_n...y_{i+1}X...X$ which replaces the faulty node. The routing algorithm that we present for achieving this goal is completely distributed and requires only local knowledge of node failures. This means that only the neighbors of a failing node need to know about the failure. The performance measure described in [3,4,4] is used to evaluate our routing algorithm. Specifically, we prove that the upper-bound on the routing performance is $2\gamma+i$ steps, where γ, $\gamma \le n$, is the number of steps required for a message to reach its destination in a fault free *n-cube*.

The routing algorithm is described in terms of the action taken at each node upon the reception of a message. Specifically, when a node $p=p_n \cdots p_1$ receives a message with a destination address $d=d_n \cdots d_1$, it computes $x=x_n \cdots x_1$, the bit-wise "exclusive or" of p and d. If $x_j=0$, for $i=1,...,n$, then p is the destination node, otherwise, the message is sent to a neighboring node according to some set of rules which depends on whether p is an original node or a spare node. The rules specify a list of alternatives for the *next node* to which the message should be sent. If a specified *next node* is down, then the next alternative in the list is tried until one is successful. For the original nodes the alternatives are (in order):

1. the node across dimension j, where j is the largest integer such that $x_j=1$
2. the spare node which is connected to p
3. any adjacent node in the current FTBB.

For the spare nodes, the alternatives are:

1. if $x_j=0$, $j=i+1,...,n$, then send the message to d. This is possible because the current node is in the destination FTBB.
2. the spare node across dimension j, where j is the largest integer between $i+1$ and n such that $x_j=1$
3. any original node within the current FTBB.

According to the above lists, if an original node can't route a message to the next FTBB because of a node failure, then it tries to route the message to that FTBB through the *spare cube*. More descriptively, If a node p in some FTBB, say *ftbb*, wants to route a message to another node p' across dimension j, where p' is in a different FTBB, say *ftbb'*, and p' is faulty, then p has $i+1$ other disjoint paths through *ftbb* to cross dimension j and reach some node in *ftbb'*. These disjoint paths are through $i+1$ nodes in *ftbb* which are neighbors to p. Out of the $i+1$ nodes, at most one node can be faulty. If p sends the message to some non-faulty node r in *ftbb*, then, r will be able to route the message across dimension j to a node r' in *ftbb'*. If the system have not failed, then neither is r' because no more than one node can fail in *ftbb'* and p' has already failed. Hence, at most 2 steps will be required to reach *ftbb'* from *ftbb*. A similar argument may be used for spare nodes.

Now, consider a message that would require γ routing step in a fault free environment. In the presence of faults, this message will require at most 2γ steps to reach a node in the destination FTBB. Once the message reaches the destination FTBB it can be shown that it may reach the destination node in at most i steps. This leads to the following proposition:

Proposition 1: The upper bound on the number of steps required for routing a message in the presence of k faults is $\min\{2\gamma+i, \gamma+k+i\}$, where γ is the number of steps required for routing the same message in a non- faulty cube and 2^i is the number of original nodes in an FTBB.

In Section 3, simulation results will show that, on the average, the routing performance is much better than the worst case bound of Proposition 1. For the maximum number of faults, that is 2^i faults, the average number of routing steps is found to be 1.2γ.

Although the fault tolerance scheme described in this section exhibits good routing performance, it leads to a relatively poor reliability. This is due to the fact that each node may be covered by only one specific spare. In the following scheme, a more flexible architecture is used at the expense of a more complex routing algorithm.

3. SCHEME II: A Double coverage Fault Tolerant Architecture.

In this scheme, the basic block (FTBB) for constructing a fault tolerant hypercube is a 3-dimensional hypercube of original nodes with an 2-dimensional hypercube of spare nodes embedded in it. That is four spare nodes are assigned to each eight original nodes. As shown in Fig 2, the spares are placed in four faces of the FTBB in a regular fashion such that each spare covers four original nodes and each original node has 2 adjacent spares that can cover it.

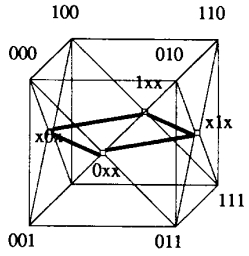

Figure 2. FTBB for scheme II

An original node in an FTBB has a three bit address and a *face* is a side/plane of the FTBB defined by a set of four original nodes. There are six faces in an FTBB, and each is identified by a generic address of the form bXX, XbX or XXb, where $b=0$ or 1 is called the face bit and X is a generic bit. One spare node is added to each of the faces $0XX$, $1XX$, $X0X$ and $X1X$, and is connected to all the nodes on that face. A spare node that is added to a face has a generic address identical to the address of that face. For example, in Fig 2 the original nodes 000, 001, 010, 011 define the face $0XX$ to which the spare $0XX$ is added.

At any time during the operation of the system, a spare node is in one of the following three states:

Ready : Ready to take over a failed node.
Busy : Already took over a failed node.
Down : Spare node has failed

If an original node fails, then any of its two adjacent spare nodes can take over. For example, the original node 011 has the spare nodes 0XX and X1X adjacent to it and, if it fails, either 0XX or X1X can take over. If one of the spares is in the *busy* or the *down* state, then the other may be used to cover for 011. If both spares are in the *busy* or *down* states, then the system fails. It will be shown in Section 4 that this double coverage feature leads to a very good spare utilization, and thus, high probability of system survival.

Fault tolerant embedded hypercubes of higher dimensions can be built by connecting several FTBB's. For example a 4-dimensional fault tolerant hypercube can be constructed by taking two FTBB's and connecting the corresponding nodes (see Fig 3). In general, a n+1-dimensional fault tolerant hypercube can be built by connecting two n-dimensional fault tolerant hypercubes.

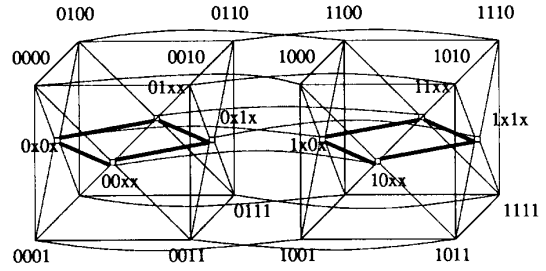

Fig 3. A 4-dimensional Scheme II architecture

In an n-dimensional (n>3) fault tolerant hypercube each node has an n-bit address. The most significant $n-3$ address bits of a node form the address of the FTBB which contains the node and the least significant address bits form the address of the node within the FTBB. For example, if the address of a node is given by $p_n p_{n-1} \cdots p_3 p_2 p_1$, n>3, then $p_n p_{n-1} \cdots p_4$ is the address of the FTBB containing the node, and $p_3 p_2 p_1$ is the address of a node within that FTBB. For simplicity, a node will be identified by its 3-bit address within an FTBB whenever a specific FTBB is identified by the context.

The routing algorithm.

As in SCHEME I, a spare that covers for a node inherits its address and it is the responsibility of the routing algorithm to deliver a message to its correct destination. This task is made complex by the fact that a failing node may be covered by one of two spares and a message addressed to the failing node should be delivered to the correct spare. Also, because of the flexibility of the architecture, a distributed routing algorithm that is not carefully designed may lead to a live-lock situation in which a message circulates in a closed loop, thus never reaching its destination. We will describe a correct, live-lock free, distributed routing algorithm in which each node has local knowledge of the system. Specifically, each node needs to know only the status of its neighboring nodes. This knowledge is used to relay any received message not destined to that node to some other node according to a set of rules.

In order to avoid live-lock, some order need to be defined among the neighbors of a given node within an FTBB. Consider first an original node whose address is $p = p_3 p_2 p_1$. The node p has five neighbors within its FTBB, three original nodes and two spare nodes. An order among the original nodes is explicitly defined by the dimensions of the cube and an order among the spare nodes may be defined according to the position of the face bits in their generic addresses. More precisely, the two spares connected to p have addresses of the form bXX and $Xb'X$, with b and b' being zero or 1. The node bXX is called the *zeroeth* spare of p and the node $Xb'X$ is called its *first* spare.

A spare node s has six neighbors within its FTBB, four original nodes and two spare nodes. The original nodes are ordered according to the bits at which they differ. For instance, if s=bXX, b=0 or 1, then its four original neighboring nodes $b00$, $b01$, $b10$ and $b11$ are called, respectively, the zeroth, first, second and third adjacent original node of s. Similarly, if s=XbX then the four nodes $0b0$, $0b1$, $1b0$ and $1b1$ are called, respectively, the zeroth, first, second and the third adjacent original nodes of s. Finally, the two spare neighbors of s are ordered according to the value of their face bit. The node with a face bit equal to zero is called the *zeroeth* spare neighbor of s, and the node with a face bit equal to one is called the first spare neighbor of s. For example, the spare node 1XX has two adjacent spare nodes X0X and X1X, which are called the zeroth and the first adjacent spare nodes of 1XX, respectively.

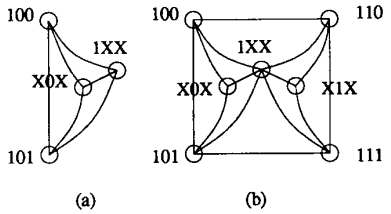

Fig 4. (a) CornerQuad (b) HalfCube

Definition 1: A node is called *k–accessible* if, within its FTBB, all but k of its neighbors are faulty. Accordingly, a *1–accessible* node has only one non-faulty neighbor within its FTBB, and all the neighbors of a *0–accessible* node within its FTBB are faulty. A *0–accessible* node is called *isolated*.

Definition 2: A *corner–quad* is a configuration (within a FTBB) of two original nodes along with two common adjacent spare nodes such that these four nodes along with their links form a complete graph of four vertices/nodes. The two original nodes are neighbors along dimension 1, the dimension that separates the two faces in the FTBB that do not contain spares, namely faces $XX0$ and $XX1$ (see Fig 4(a)).

Definition 3: A *half–cube* consists of two adjacent corner-quads of an FTBB. It has four original nodes and three spare nodes along with their links (see Fig 4(b)).

Definition 4: For any FTBB, say *ftbb*, the neighbor of *ftbb* across dimension k, k>3, is the FTBB whose nodes are neighbors to the nodes of *ftbb* across dimension k. Neighboring corner-quads and half-cubes are defined similarly.

Definition 5: A hypercube architecture, augmented using SCHEME II, fails if it contains a failing node that cannot be covered by a spare. A system which have not failed is called *live*.

Given that a corner quad contains only two spare nodes, it is implied that no more than two nodes can fail in any corner quad in a *live* system. Since a corner-quad is a fully connected graph of four nodes, any two node failures within a corner quad results in a connected graph of two nodes. This implies that if we can route a message to a non faulty node in the proper corner-quad then it should reach the destination since there exists a path to the destination. Similarly, in a half-cube, no more than three nodes can fail.

We are now ready to describe the routing algorithm. As in the algorithm for SCHEME I, a message is first routed to the FTBB containing its destination node, and once the message has reached the destination FTBB, it is then routed within that FTBB to the destination node. In order to use such a two phase routing algorithm, we have to guarantee that there always exists a non-faulty path (through non-faulty nodes) between any two FTBB's, and that there always exists a non-faulty path between any two non-faulty nodes within an FTBB. In graph theoretical terms this means that the FTBB's and the non-faulty nodes within an FTBB should remain connected.

Lemma 1: If the system is *live*, then there exist a path between any two FTBB's.

Proof: Let e, $e \le 8$, be the number of spares in an FTBB. There are $8+e$ nodes in any FTBB, which means that there exists $8+e$ different paths through $8+e$ nodes of the FTBB to any adjacent FTBB. Since no more than e nodes can fail in one FTBB, a total of $2e$ nodes can fail in any two adjacent FTBB's, implying that at most $2e$ out of the $8+e$ paths are faulty. Since $e<8$, there exists a non faulty path between any two FTBB's. □

Lemma 2: No node within an FTBB becomes isolated if the fault coverage scheme described in this section is used.

Proof: We will prove this lemma by proving that any node within an FTBB is at least *1-accessible*. Within a FTBB the original nodes have degree five and the spare nodes have degree six. In a FTBB no more than 4 nodes can fail otherwise the system would have failed. This means that no more than four neighbors of a node within the FTBB can fail implying that an original node within a FTBB is at least *1–accessible* and a spare node is at least *2–accessible*. This implies that any node within a FTBB is at least *1–accessible*. □

The above two lemmas show that there exists a path between any two non-faulty nodes in the augmented hypercube structure. For a specific message, this path may be determined distributively. That is each node on the path may determine the next node from local information about the status of its neighbors. More specifically, if a node $p=p_n....p_1$ receives a message addressed to a node $d=d_n \cdots d_1$ which is different from p, then it determines the next node on the message path by using the following information:

1) $x = p_n \cdots p_1 = p \ xor \ d$, the bit-wise exclusive or of p and d
2) For each node, g, adjacent to p within the current FTBB (the FTBB whose address is $p_n \cdots p_4$), an indication on whether g is failing, is *1–accesible* or is *k–accessible*, for some $k>1$,
3) For any neighboring node g within the current FTBB which is in the busy state, the inherited address of g. That is the address of the node that g have replaced.
4) The node $v=v_n \cdots v_1$ from which the message was received.

The routing rules at p is a list of alternatives for the "next node" to which the message is to be sent. The alternatives are attempted in order, and an alternative is skipped if: (1) it specifies a failing node, (2) the node v from which the message was received or (3) a node which is *1–accessible* unless this node is the destination node d. The reason for not sending a message to v or to a *1–accesible* node which is not d is to prevent messages from possibly looping in a cycles.

Routing Rules for original nodes:

If there exists at least one k such that $x_k=1$, then try the following alternatives in order: (message is not at the destination FTBB)
1. For $k=4,...n$, the node across dimension k if $x_k=1$.
2. the node across dimension 1
3. the spare nodes *bXX* and *XbX*:
   if the message is received from v across dimension 2 then try *XbX* first, while if it is received across dimension 3, try *bXX* first.
   Otherwise the order is irrelevant.
4. a node across dimension 2 or dimension 3

Else, try the following alternatives in order: (p is in the destination FTBB)
1. For $k=1,2,3$, the node across dimension k if $x_k=1$.
2. the spare in the same face as d if that spare is adjacent to p
3. any of the two adjacent spare nodes in the current FTBB
4. a node across dimension 2 or dimension 3
5. the node across dimension 1

Routing Rules for spare nodes:

If there exists at least one k such that $x_k=1$, then try the following alternatives in order: (message is not at the destination FTBB)
1. For $k=4,...n$, the node across dimension k if $x_k=1$.
2. If the message is received from a different FTBB ($v_k \neq p_k$, for some $n-4 \le k \le n$), then for $k=0,..,3$, the k^{th} original neighbor of p
3. the spare node in the same corner quad as p and v
4. if $p=bXX$ then try node $v_3 \bar{v}_2 \bar{v}_1$ and then node $v_3 \bar{v}_2 v_1$, where a bar denotes the bit complement
5. if $p=XbX$ then try node $\bar{v}_3 v_2 \bar{v}_1$ and then node $\bar{v}_3 v_2 v_1$,

Else, try the following alternatives in order: (message is at the destination FTBB)
1. node d, if d is adjacent to p
2. the spare in the same face as d
3. the other spare node within the current FTBB
4. if v is a spare node, then route to the original node which is at a minimum distance from d
5. If $p=bXX$ then try node $v_3 d_2 \bar{v}_1$ and then $v_3 \bar{d}_2 \bar{v}_1$
6. If $p=XbX$ then try node $d_3 v_2 \bar{v}_1$ and then $d_3 \bar{v}_2 \bar{v}_1$

Informal description of the routing algorithm

Two lists of alternative next nodes are used to route messages at each node. The first is used for routing messages among FTBB's, and is applied whenever the current node is not in the same FTBB as the destination node. That is, whenever not all x_k's, $k=4,...,n$ are equal to zero. In this case, p is said to be in the first phase of the message route. If, during this phase, routing from one FTBB to another fails then the algorithm tries to limit the routing trials, first to a *corner quad* and then to a *half–cube*. The reason is that limiting trials to a *corner quad* (or a *half–cube*) will allow the message to reach the next FTBB in at most 3 steps (or 4 steps). If the fault distribution makes it impossible to stay within the same *half–cube*, then at most 5 steps may be required to reach the next FTBB.

We clarify this by an example. Let *ftbb* be the FTBB that contains the current node, p, and assume that p tries to route a message along dimension k, $k \ge 4$, to a node p' in *ftbb'*. If p' is faulty, then, according to the routing rules, p sends the message to a node p_1 which is in the same corner quad as p. Denote this quad by C and its neighbor across dimension k by C'. If the node p_1' which is across dimension k from p_1 is not faulty, then the message can be delivered to *ftbb'* in two step. However, if p_1' is also faulty, then p_1 tries to route the message to another node p_2 in C because, in this case, the node p_2' across dimension k from p_2 may not be faulty (only two nodes can fail in the corner quad C'). If such p_2 cannot be found in C, then the next node, p_2, will be in a *half–cube* containing C. Denote this *half–cube* by H and its neighbor across dimension k by H'.

Now, if the node p_2' across dimension k from p_2 is not faulty, then the message can be delivered to *ftbb'* in three steps. But, p_2' may also be faulty. In this case, the algorithm will try to route the message to a non-faulty node p_3 in H because the neighbor of such p_3 across dimension k cannot not be faulty (at most three nodes may be faulty in H'). However, if p_2 does not have a non-faulty adjacent node in H, then the next node, p_3, will not be in H. In this case p_3' may be faulty, and, once again, p_3 will have to route the message to a fourth node p_4 within *ftbb*. Now, p_4' may not be faulty (at most 4 nodes may fail in *ftbb'*) and the message will be delivered to *ftbb'* in 5 steps.

Once the message arrives at a node p in the FTBB which contains the destination node d ($x_k=0$, $k=4,...,n$), the second phase of routing begins and the second list of alternative nodes applies. According to the routing rules for that second phase, if $d \neq p$, then the routing algorithm tries to route the message to a node in the *corner quad* that contains d. If not successful, it tries to route the message to a node in a *half–cube* that contains d. If not successful, it tries the nearest node from the destination node within the FTBB itself. In the worst case, it may be shown that the message will be delivered to d in at most 6 steps.

From the above informal argument, it is clear that the upper bound on the number of steps required for routing a message using SCHEME II is $5\gamma+6$, where γ is the number of steps required for routing the message in a non-faulty cube. However, this upper bound is achievable only if the source and the destination nodes are in two FTBB's that are neighbor across some dimension k, and the eight faults in the two FTBB's are such that no two nodes across dimension k are faulty. The probability that the system reaches the above configuration is very small, and according to some initial simulation results, the average number of routing steps is much less than this worst case bound.

Using a proof technique which is based on the construction of a routing tree at each routing step, it is possible to prove [1] that if each node applies the above routing rules, then a particular message will not visit the same node twice, and hence, messages may not circulate in loops. Such guarantee of live-lock free routing in all possible fault configurations is the

main contributor to the complexity of the routing rules. It should be noted, however, that in typical fault configurations, only the first few alternatives in the list of "next nodes" will need to be examined. Moreover, the routing rules may be easily translated into a binary function which determines, at each node, the output link to which a message should be relaid in terms of x, v and a vector which specify the status of the neighboring nodes. This function is relatively simple [1] and may be implemented in hardware if fast message relaying is desired.

4. Experimental Analysis

In order to find out the reliability of both schemes, we wrote a simulation software tool that generated 1000 different fault combinations for every j node failures, where j may vary from 1 to E, the total number of spares, and a single failure is defined as a failure in either an original node or a spare node. For each such j, we computed the number of system failures and used it to calculate the reliability curves shown in Fig 5. From these curves, it is clear that the reliability for scheme I is good for relatively small number of faults. But as the number of faults increases, the reliability of the system decreases rapidly and practically diminishes as the number of faults approaches E. In other words, the probability of fully utilizing the spare node in the system (16 spares in Fig 5(a) and 32 spares in Fig 5(b)) is practically zero. As noted earlier, this is due to the fact that each node in scheme I is covered by only one spare. The reliability of scheme II is much better than scheme I since a node may be covered by more than one spare. As portrayed in Fig 5(b), the system failure rate does not drop significantly as the number of faults increases. In fact 100% spare utilization is accomplished with probability 0.82 and 0.65 in Fig 5(a) and 5(b), respectively.

In order to study the average performance of the routing algorithms, we considered, for each number j of faults, all the configurations in which the system were *live*. For each such configuration, a 1000 messages were generated with random sources and destinations, and the routing algorithm on these messages were simulated. The total number of steps, say s, were calculated for the 1000 messages and compared with the minimum number of steps, say t, that would be required to route these messages in a fault-free

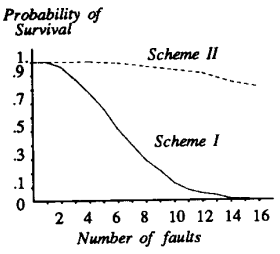

Fig 5a. 32 Nodes with 16 spares

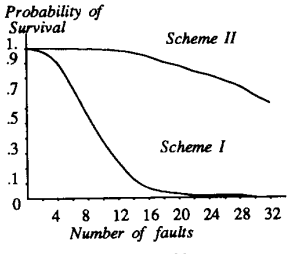

Fig 5b. 64 Nodes with 32 spares

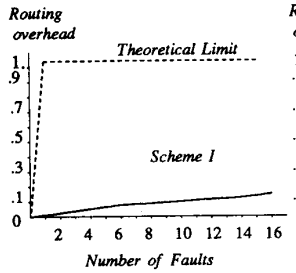

Fig 6a. 32 Nodes with 16 spares

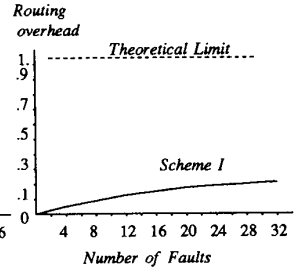

Fig 6b. 64 Nodes with 32 spares

hypercube using the usual bit-wise routing algorithm. Specifically, the routing overhead, $s-t$, were averaged for all configurations of j faults and normalized with respect to the average of t. This have been repeated for each $j=1,\cdots,E$ and the results are plotted in Fig 6 for the routing algorithm for scheme I. The Figure also shows the worst case overhead given in Proposition 1. As expected, the average overhead increases with the number of faults and reaches a maximum of 14% and 20% for the 5-dimensional and 6-dimensional hypercubes, respectively. This is clearly much less than the worst case overhead of about 100% specified in Proposition 1. We are currently working on the simulation software for the routing algorithm of scheme II and we expect similar results.

5. Conclusion

We have presented two approaches for achieving fault tolerance in a hypercube architecture where degraded performance is not allowed. The first scheme applies a straight forward reconfiguration scheme and a fairly simple routing algorithm, but suffers from rapid reliability degradation when the number of faults increases. This degradation is avoided in the second scheme by allowing any of two spares to cover for an original node. The improved reliability in this scheme is obtained at the expense of a complex routing algorithm which ensures that, for any configuration of faults, the messages will be delivered to their correct destinations without circulating in loops indefinitely. Although relatively complex, the routing algorithms are particularly attractive because, in the absence of faults, they degenerate to the ordinary bit-wise routing algorithm used in non fault tolerant hypercubes. Another possible advantage of the augmented architecture is the utilization of the many alternative message routes to alleviate traffic congestion. Such run-time adaptation of the routing algorithm to the message traffic is not considered in this paper.

Both schemes assume extremely local information at the nodes. Specifically, it is assume that each node knows only the status and address of its immediate neighbors. If this condition is relaxed, and more global knowledge is assumed at the nodes, then routing can be greatly simplified. For instance, if each node in an FTBB knows the status of all the other nodes in that FTBB, then the uncertainty at each node about future routing steps within the FTBB will be eliminated and a route that avoids cycles may be easily planed.

The augmented architectures described in this paper accommodate link failures as well as node failures. If a link fails, then a node that wants to route a message to a node d across the failed link may assume that d is faulty and apply the routing algorithm accordingly. Messages will be delivered correctly because, in our model, a node failure is considered to be a failure of a processor along with all of its links. Hence, a link failure is less severe than a node failure.

References

1. M. Alam and R. Melhem, "Fault Tolerance and Reliable Routing in Augmented Hypercube Architectures," *Technical Report - Department of Computer Science - The University of Pittsburgh.*. In preparation.
2. W. Bouricious, W. Carter, D. Jessep, P. Schneider, and A. Wadia, "Reliability Modeling for Fault Tolerant Computer," *IEEE Transactions on Computers*, pp. 1306-1311, Nov. 1971.
3. A. Broder, M. Fisher, D. Dolev, and B. Simons, "Efficient Fault Tolerant Routings in Networks," *Proc. ACM Symp. on Theory of Computation*, pp. 536-541, 1984.
4. D. Dolev, J. Halpern, B. Simons, and R. Strong, "A New Look at Fault Tolerant Network Routing," *Proc. ACM Symp. on Theory of Computation*, pp. 526-535, 1984.
5. D. Hillis, "The Connection Machine," *MIT Press, Cambridge, Mass.*, 1985.
6. K. Hwang and F. Briggs, in *Computer Architecture and Parallel Processing*, McGraw Hill, 1984.
7. F. Provost and R. Melhem, "Distributed Fault Tolerant Embedding of Binary Trees and Ring in Hypercubes," *Proceedings of the International Workshop on Defect and Fault Tolerance in VLSI Systems*, Oct. 1988. To appear.
8. D. A. Rennels, "Fault Tolerant Computing: Concepts and Examples," *IEEE Trans. Computers*, pp. 1116-1129, Dec 1984.
9. D. A. Rennels, "On Implementing Fault-Tolerance in Binary Hypercubes," *Proc. IEEE Fault Tolerant Computing*, pp. 344-349, 1985.