Contention Sensitive Fault-Tolerant Routing Algorithms for Hypercubes

Ramaraghavan Srinivasan, Vipin Chaudhary[1] , AND Syed M. Mahmud
Parallel and Distributed Computing Laboratory
Department of Electrical and Computer Engineering
Wayne State University, Detroit, MI - 48202.
Phone: (313) 577-0605
Email: vipin@eng.wayne.edu

Abstract

In this paper we present two new fault-tolerant routing algorithms for hypercubes. The first algorithm requires only local knowledge of the faults whereas the second algorithm requires global knowledge. Unlike previous fault-tolerant routing algorithms, our algorithms take into consideration the dynamic conditions (link contention) of the network. We have shown that checking for dynamic conditions in fault-tolerant algorithms is essential. Performance evaluation by extensive simulation of our algorithms and other fault-tolerant routing algorithms show that ours are better than previous algorithms by as much as 50% and 500% in time and space, respectively. We also observed that global information about the location of faults does not give us additional benefit. This observation is true regardless of the consideration of the dynamic conditions in the network.

1 Introduction

The hypercube topology has been a popular candidate for multiprocessor machines in recent years. The Cosmic Cube 1 [1], iPSC-2 [2], and Ametek 2010 are commercially available hypercube architectures. Hypercubes have many desirable properties such as structural regularity, easy scalability, strong hierarchy, and low degree and diameter [5]. A good routing algorithm is a must for efficient execution of problems in any multiprocessor machine. The routing algorithm must be adaptive. Furthermore, due to presence of thousands of components in such a machine, the routing algorithm must be capable of routing despite the failure of a few of these components. As there exist many parallel paths between any two nodes in a hypercube [5], it is easy to develop both adaptive as well as fault-tolerant routing algorithms for hypercubes. In this paper we present two fault-tolerant routing algorithms for hypercubes which consider the dynamic conditions of the network. These algorithms are shown to be much superior to the existing algorithms.

Numerous routing algorithms have been proposed for hypercubes. Sullivan [3] proposed a deadlock-free deterministic routing algorithm in which a message is routed in increasing order of dimensions. The path taken by a message is fixed for a given source and destination node pair. Though this algorithm is simple and easy to implement in hardware, it is too restrictive and does not use the rich connectivity of the hypercube. Several adaptive, minimal path routing algorithms have also been proposed for hypercubes. Li [9] proposed an algorithm called $k-1$ in which there exist k paths in one direction and one path in the opposite direction between any two nodes which are k hops apart. This algorithm is not fully adaptive and its performance is similar to the deterministic algorithm at high network traffic. Konstantinidou [8] presents another partially adaptive algorithm. In his algorithm certain nodes become the bottleneck even during moderately heavy traffic.

As the size of the network grows the probability of a component failure increases. Hence, the routing algorithm must be capable of avoiding these areas of faults. Fault-tolerant routing algorithms for several networks have been proposed. Gordon and Stout [7] present a framework for the analysis

[1]This author's research was supported in part by NSF Grant MIP-9309489 and Wayne State University Faculty research award.

0-8186-6507/94 $4.00 © 1994 IEEE

of fault-tolerant routing algorithms for hypercubes. They study the probability of a component failure versus the probability of successful delivery of messages. Esfahanian and Hakimi [4] present fault-tolerant routing algorithms for DeBruijn networks. Chen and Shin [6] present two fault-tolerant algorithms for the hypercubes. In the first algorithm each node requires knowledge only about the links connected to it, whereas in the second algorithm the nodes require global knowledge. None of these fault-tolerant routing algorithms consider the dynamic behavior of the network like avoiding link contention.

The contributions of this paper can be summarized as follows:

- We show that checking for dynamic conditions in fault-tolerant routing algorithms leads to much improved performance.
- We present two fault-tolerant algorithms for hypercubes which incorporate checks for dynamic conditions (link contention). The first algorithm requires no global knowledge of the network fault conditions while the second algorithm requires fault information for every link or node. These algorithms perform as much as 50% and 500% better than previous algorithms in time and space, respectively. Our results are based on extensive simulation of the algorithms.
- We also observe that global information about the location of faults does not give us additional benefit. This observation is true regardless of the consideration of the dynamic conditions in the network.

The rest of the paper is organized as follows. Section 2 describes the necessary notation and definitions followed by a description of the two fault-tolerant routing algorithms suggested by Chen and Shin [6] in Section 3. The deficiency of both these algorithms is also shown in Section 3 using an example to create a live-lock situation. Section 4 describes the two new adaptive algorithms. The simulation model used is described in Section 5. Section 6 discusses the results of the simulation and compares the performance of the algorithms. The paper concludes with Section 7.

2 Preliminaries

In this section we describe the necessary notation and definitions that will be used in this paper.

A hypercube of dimension 'n' or an n-cube consists of $N = 2^n$ nodes, with the nodes numbered from 0 to $2^n - 1$. Every node in an n-cube is represented as an n-bit binary number. The least significant bit or LSB represents dimension 1, the next significant bit represents dimension 2, and so on till the most significant bit or MSB, which represents dimension n. There exists a link between any two nodes iff the binary representation of their numbers vary by one bit. Each link is represented by a binary string with a "-" symbol in the corresponding dimension. For example, the link between 101 and 111 is represented by 1-1 as shown in Figure 1(a).

Following are some definitions of terms that will be used frequently in this paper.

- A connected hypercube with faulty nodes or links is called an *injured hypercube*.
- The *Hamming* distance between two nodes A and B with addresses $a_n a_{n-1} \ldots a_1$ and $b_n b_{n-1} \ldots b_1$ is defined as [6]

$$H(A,B) = \sum_{i=1}^{n} h(a_i, b_i) \text{ where } h(a_i, b_i) = \begin{cases} 1 & \text{if } a_i \neq b_i \\ 0 & \text{if } a_i = b_i \end{cases}$$

- A *path* in a hypercube can be represented as a sequence of nodes in which every two successive nodes are neighbors. For example, let $n = 4$, message source node be $3 = 0011$, and destination node be $12 = 1100$. One path between the two nodes is {3,7,15,14,12}. An optimal path is a path whose length is identical to the Hamming distance between the source and the destination.
- A *coordinate sequence* [c_1,c_2, ... c_n] represents the order of dimension in which two nodes differ. A path between any two nodes can also be represented as a coordinate sequence. For example, the coordinate sequence for the path in the previous example is [3, 4, 1, 2].

We now state a few relevant properties of a hypercube. If A and B are any arbitrary nodes in an n-cube and $k = H(A,B)$ is the Hamming distance between the two nodes then [5],

- There are k node disjoint optimal parallel paths between A and B.

- There are $k!$ optimal parallel paths between A and B.
- There are exactly n node disjoint parallel paths between A and B, of which k paths are of length k and the remaining $n-k$ paths are of length $k+2$.

From the above properties it is obvious that if the number of faulty nodes and links is less than n, then there is at least one path of length $k+2$ between any two non faulty nodes, k hops apart [5].

3 Drawbacks of previous fault-tolerant algorithms

In this section we describe two fault-tolerant routing algorithms suggested by Chen and Shin [6]. The deficiency of both these algorithms is also shown by using an example to create a live-lock situation.

Previous work on fault-tolerant routing [4, 6, 7] do not take into consideration the dynamic conditions in the network. In this section we discuss two fault-tolerant algorithms, namely, **A1** and **A2** [6]. Both **A1** and **A2** route a message in a hypercube with a maximum of $n-1$ faulty components. Besides the message itself, every packet carries a coordinate sequence of the path and a tag field given by $(d_0 d_1 ... d_{n-1})$. The latter is used to keep track of *spare dimensions* that are used to bypass faulty nodes or links. The n bits of the tag are reset to 0 at the start of the message. So, a message is represented as $(k, [c_1, c_2, ...c_k], tag, message)$ where k is the number of hops remaining to be traversed by the message. On receipt of a message each node checks the value of k. A message reaches the destination if $k = 0$, else each node attempts to route the message via an available optimal path along the dimensions in the coordinate sequence. If all such paths are faulty then the node uses a spare dimension to bypass the injury, and the information that the message used a spare dimension is stored in the tag carried along with the message.

The difference between the two algorithms **A1** and **A2** is in the amount of network information present at each node. In **A1**, each node has information only about those links connected to itself. Due to the limited information the path chosen by a message is not always optimal. On the other hand, in **A2**, enough information is available at each node to enable the message to be routed along an optimal path (if one exists) [6]. As a result the message is capable of bypassing faulty nodes or links along the path to the destination. For details about algorithms **A1** and **A2**, refer to [6].

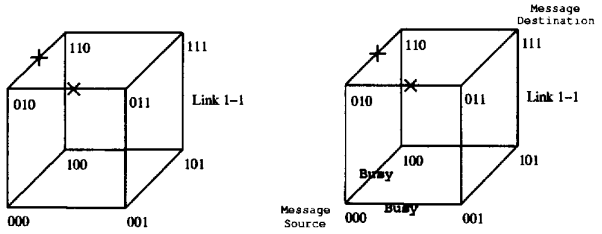


Figure 1. *Illustrating the failure of Algorithm A1.*

We now present an example to illustrate algorithm **A1**. Consider a 3-cube shown in Figure 1(a) where the links -10 and 01- are faulty. Let node 1 generate a message "m" for the destination node 2. The original message as semt by the source is $(2, [2,1], 000, m)$. It sends $(1, [1], 000, m)$ to node 3. Since the first dimension is faulty and no other optimal path exists, node 3 will use a spare dimension and route $(2, [1,3], 101, m)$ to node 7, which sends $(1, [3], 101, m)$ to node 6. Now dimension 3 is faulty and no other optimal path exists. So node 6 uses a spare dimension and

sends $(2,[3,2],111,m)$ to node 4 which sends $(1,[2],111,m)$ to node 0, and finally node 0 sends $(0,[],111,m)$ to 2, which is the destination. The total number of steps required to route is six.

Based on the above algorithm we can observe the following [6]:

- The number of hops is increased by two whenever a spare dimension is used.
- The maximum number of hops required to route using A1 algorithm is $H(A,B)+2*(n-1)$. This follows from the fact that a message may use at most $n-1$ spare dimensions.

Algorithms **A1** and **A2** are both fault-tolerant but they do not take care of the dynamic conditions of the network. In each of the above algorithms, the condition of a link is either *faulty* or *not faulty*. A message requiring a link that is busy must wait until that link is free. It cannot be routed on one of the remaining dimensions even though these links may be idle. The major advantage of the hypercube architecture is its rich connectivity. The number of optimal parallel paths between two nodes A and B is $H(A,B)!$. But these algorithms are not capable of using all the potential paths. We now present an example to illustrate the need for new algorithms. Figure 1(b) shows the occurence of a live-lock situation if we use algorithm **A1** to take care of dynamic conditions.

Figure 1(b) shows a 3-cube with links -10 and 01- faulty. The source and destination are nodes 0 and 7, respectively. Further, let links -00 and 00- be busy with other messages. Initially, the message at node 0 is $(3,[3,2,1],000,m)$. Node 0 finds link -00 busy and sends message $(2,[3,1]000,m)$ to node 2 along dimension 2. Since there is no optimal path from node 2 to node 7, the message $(3,[3,1,2],111,m)$ is sent back to node 0. If the links -00 and 00- are still busy then the message $(2,[3,1],111,m)$ will be routed again to node 2 along dimension 2. When a search for spare dimension is made, the message finds all bits of the tag to be set. Thus, the message will not be able to route from node 2 and suffers *live-lock*.

```
Algorithm N1
if (the message has not terminated) then
  Try routing along a dimension in the remaining coordinate sequence.
  if (spare dimension was not used last hop) then
    check all k dimensions
    if (one of the required link is not faulty and not busy) then
      route message
    if (at least one of the required links is not faulty but busy) then
      block message
    if (All required links are faulty) then
      Record all faulty dimensions.  Set tag_modified bit.
      Choose a spare dimension (a link along a non-optimal path)
      if (that channel is free) then
        route message
      else
        block message
  else /* Spare dimension was used last hop */
    check only the first 'k - 1' dimensions
    if (the required link is not faulty and not busy) then
      route message
    if (at least one of the required links is not faulty but busy) then
      block message
    if (All required links are faulty) then
      backtrack on the k^th dimension
end.
```

4 New fault-tolerant algorithms

In this section we present two new fault-tolerant algorithms for an injured hypercube with a maximum of $n-1$ faulty nodes or links. These algorithms take into consideration the dynamic

condition of the network. In algorithm **N1** a node has information only about the links connected to it, whereas in **N2** the nodes have sufficient information to route a message using optimal path. The procedure to acquire global knowledge is the same as in Chen and Shin's algorithm [6]. Detailed pseudo-code of algorithms **N1** and **N2** are given in appendix A.

4.1 Algorithm N1

In algorithm **N1**, the message is represented as (k, $[c_1, c_2, ..., c_k]$, *tag*, *tag_modified*, m) where

- k is the number of hops from current node to destination node,
- $[c_1, c_2, ..., c_k]$ is the coordinate sequence from current node to destination node,
- *tag* is the n bit tag field containing information on faulty nodes visited (all n bits are initially cleared),
- *tag_modified* is a single bit that is set if *tag* is modified during the last hop (this bit is initially clear), and
- m is the message itself.

In algorithm **N1** the coordinate sequence is checked to see whether the message has reached the destination. If it has not reached then the *tag_modified* bit is checked. If the *tag_modified* bit is cleared then the message did not use a spare dimension during the last hop and the algorithm attempts routing on all the k dimensions. If all the k dimensions are faulty then the message uses a spare dimension (i.e., a non-optimal path). Else, the message blocks till a link becomes available.

When the *tag_modified* bit is set, the algorithms attempts to route the message only along the first $k-1$ dimensions in the coordinate sequence. This is due to the fact that the k^{th} dimension was a spare dimension during the last hop and the message may backtrack *iff* the first $k-1$ links are faulty. Otherwise, there is a possibility that the message could *ping pong* between two nodes. If one of the required links is not faulty and not busy then the message is routed along that channel. Again, if none of the links are available, but not all of these are faulty, then the message blocks till the link(s) become available. Finally, if all the $k-1$ links are faulty then the message backtracks along the k^{th} dimension. For detailed psuedocode please refer to [10].

4.2 Algorithm N2

```
Algorithm N2
if (the message has not terminated) then
  Try routing along a dimension in the remaining coordinate sequence.
  if ((the required link is not faulty) and (not busy) and
  (there exist an optimal path along that link)) then
    route message
  if ((at least one of the required links was non-faulty) and (busy)) then
    message blocks
  if (all the required links are faulty) then
    Record all faulty dimensions.
    Choose a spare dimension (a link along a non-optimal path)
    if (that link is free) then
      route message
    else
      block message.
end.
```

In algorithm **N2** we do not require the additional bit (*tag_modified*) in the message header. If a message takes a spare dimension, say from node A to node B (because of all optimal paths from A to destination being blocked), then node B has the information that there exist *no optimal path to the destination through A*. So the message will never go back to node A from which it arrived.

Thus, we do not need *tag_modified* in the message to show whether or not a spare dimension was used during the last hop. While checking the coordinate sequence the algorithm sets a *flag* if (an) optimal path(s) exists. If the message is not routed due to link contention, it must block. But if no such optimal path exists, then the flag stays reset and the message uses a spare dimension. We now present an example to illustrate algorithm **N2**.

Consider the injured cube shown in Figure 1(b) with links -10 and 01- faulty. Further, let node 2 and node 7 be the source and destination of the message, respectively, and links -00 and 00- be busy with other messages. So, initially the message generated at node 2 is $(2, [3, 1], 000, m)$. Since all optimal paths from node 2 to node 7 are blocked, the message $(3, [3, 1, 2], 111, m)$ reaches node 0. Though links -00 and 00- are busy, the message will not try link 0-0 to node 2, but instead it sets a flag and blocks till one of the two busy links become free (since node 0 has information that there is no path to node 7 via node 2). Subsequently when the links become free the message is routed to node 7. For detailed psuedocode please refer to [10].

5 Simulation Model

This section describes in detail the model used during simulation. A total of 5 event driven simulators were written. The algorithm simulated were **A1** and **N1** (with no global information), **A2** and **N2** (with enough global information for optimal path routing), and finally the un-injured hypercube as a reference. In all these simulations the unit of time was the time required to transmit one bit between two adjacent nodes. Each node was assumed to have an infinite buffer to queue the messages that contend for the same channel. The simulation also uses a uniform reference pattern which means that all nodes are equally likely to be the destination. Message lengths are geometrically distributed in multiples of bytes with a mean of 25 bytes or 200 bits.

The performance of a routing strategy is measured by means of the following parameters, namely, *average message latency* and the *average message buffer length*. A good routing algorithm has a small value for both these parameters. The effect of injuries on the maximum network traffic is studied, i.e., the value of inter-arrival time at which the network saturates due to hot-spots or large message buffer. We also study the effect of proximity to injured nodes or links on message latency and queue length. This was done by calculating the mean latency of nodes as a function of number of injured links they have.

Each of these programs were executed for

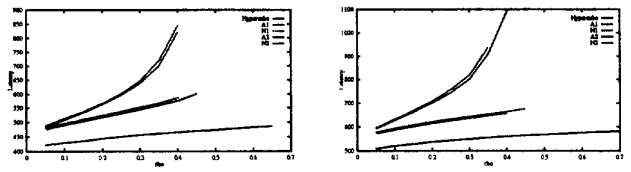
- 4-cube and 5-cube.
- Keeping the value of n constant, the value of ρ was varied from 0.05 to 0.40 in steps of 0.05 (By varying ρ we vary the mean interval).
- For each value of n and ρ, 50 different sets of $n-1$ nodes from 2^n nodes were randomly chosen. These nodes were designated as faulty nodes.
- For a given value of n and ρ and for a given set of injured nodes the simulation was executed with ten different initial values of random number. For each set the confidence interval is 95% with an error of less than 5%.
- Each of the simulation was run for a duration of two million time units.

For further details on the simulation model please refer to [10].

6 Results and Discussion

In this section we present simulation results of the fault-tolerant routing algorithms in an injured hypercube. We show that performance improves dramatically for our algorithms and also that additional global information provided at each node to route optimally does not improve performance at all.

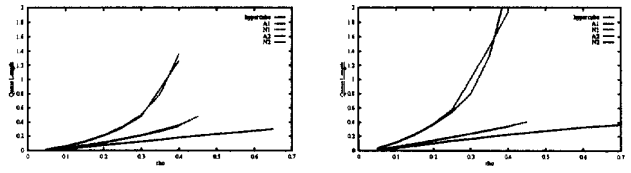
Figure 2(a) and Figure 2(b) show the variation of message latency with respect to the injection ratio ρ. It can be seen that there is a dramatic difference between algorithms **A1** and **N1**, and between **A2** and **N2**. The improvement is more than 50% at high traffic. On the other hand the difference between **N1** and **N2**, and **A1** and **A2** is negligible, which illustrates the fact that



(a) 4 Cube (b) 5 Cube

Figure 2. *Average Latency vs ρ.*

additional global information does not help. This is due to the fact that the number of optimal paths that exist between two nodes in an injured hypercube is less than in an uninjured hypercube. So the remaining channels are loaded more heavily, which causes the queues to be much longer, thereby increasing the message latency. This problem gets worse as the traffic increases and finally these crucial links are used fully at which point the network can no longer sustain the traffic and we have unbounded queues on the nodes connected to these faulty links. Thus, the advantage of routing through an optimal path, namely, fewer number of hops, is offset by a reduction in number of parallel paths available.



(a) 4 Cube (b) 5 Cube

Figure 3. *Average Queue Length vs ρ.*

Figure 3(a) and Figure 3(b) show the average length of the message buffers or queues in the network. We again see that the performance of algorithms N1 and N2 are far superior to A1 and A2. At high traffic algorithms A1 and A2 require 5 to 10 times as much buffer space as N1 and N2. The performance of N1 and N2 is only slightly inferior to routing in un-injured hypercube. Furthermore, the negligible difference in queue length between N1 and N2, and A1 and A2 clearly shows that additional global information does not improve performance.

7 Conclusion

In this paper we have proposed two fault-tolerant routing algorithms for injured hypercubes which also take into account the dynamic conditions of the network like link contention. Algorithm N1 requires each node to know only the fault information of its links and routes messages along optimal paths. Non-optimal paths are used only if all the optimal paths are blocked as a result of faulty components. However, due to the insufficient information on faulty components, N1 does

not guarantee optimal paths. To ensure shortest path routing, we proposed algorithm N2 which requires each node to have enough information to enable the message to be routed along optimal path, if one exists.

From our extensive simulation study we can conclude that checking the dynamic conditions of the network is very important. The proposed algorithms performed by as much as 50% and 500% better than previous fault-tolerant algorithms in terms of time and space, respectively.

We also observed that routing algorithms with global knowledge do not seem to perform much better than algorithms with knowledge only about their own links, when routing along optimal path whenever possible. The improvements in time and space using global knowledge is marginal whereas the overhead in acquiring this global knowledge can be substantial.

8 Acknowledgements

We would like to thank Sumit Roy for his comments on a previous version of this paper.

References

[1] C. L. Seitz, "The Cosmic Cube," *Commun. ACM*, Vol. 28, N0.1, pp. 22-33, Jan. 1985.

[2] Intel Corporation, "iPSC User's Guide," Aug. 1985.

[3] H.Sullivan and T.R.Brashkow, "A large scale homogeneous machine," *In Proc. 4^{th} Annu. Symp. Comput. Archi.*, pages 105-124, 1977.

[4] Abdol-Hossein Esfahanian and S. L. Hakimi, "Fault-Tolerant Routing in DeBruijn Communication Networks," *IEEE Trans. on Computers*, Vol. C-34, No. 9, pp. 777-788, Sept. 1985.

[5] Y.Saad and M.H. Schulz, "Topological Properties of hypercubes," *IEEE Trans. on Computers*, Vol.37, No.7, pp. 867-872, July 1988.

[6] Ming-Syan Chen and Kang G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers," *IEEE Trans. on Computers*, Vol.39, No.12, pp. 1406-1416, Dec 1990.

[7] Jesse M. Gordon and Quentin F. Stout, "Hypercube Message Routing in the Presence of Faults," *Proc. Third Conf. Hypercube Concurrent Comput. Appl.*, Jan. 1988, pp. 318-327.

[8] Smaragda Konstantinidou, "Adaptive, Minimal Routing in Hypercubes," *Proceedings of the 6^{th} MIT Conference*, pp. 139-153, 1990.

[9] Qiang Li, "A Deadlock-free Adaptive Routing Algorithm for Direct Binary Hypercubes", in *Algorithms, Software, Architecture* (Editor: J. van Leeuwen), pp. 545-554, *Information Processing* 92, Volume I, Elsevier Science Publishers B.V. (North Holland).

[10] R. Srinivasan, Vipin Chaudhary and Syed M. Mahmud, "Contention Sensitive Fault-Tolerant Routing Algorithms for Hypercubes" it Technical Report TR-93-10-10, Department of Elect. and Comp. Engg., Wayne State University, Detroit, MI, 1994.