Proceedings of of the 5th Intl. Conf. on PDCAT, Dec. 8-10, 2004 Singapore. pp.196-201

An Efficient Algorithm for Fault Tolerant Routing Based on Adaptive Binomial-Tree Technique in Hypercubes

Yamin Li[1], Shietung Peng[1], and Wanming Chu[2]

[1] Department of Computer Science, Hosei University, Tokyo 184-8584 Japan
[2] Department of Computer Hardware, University of Aizu, Fukushima 965-8580 Japan

Abstract. We propose an efficient fault-tolerant routing algorithm for hypercube networks with a very large number of faulty nodes. The algorithm is distributed and local-information-based in the sense that each node in the network knows only its neighbors' status and no global information of the network is required by the algorithm. For any two given nonfaulty nodes in a hypercube network that may contain a large fraction of faulty nodes, the algorithm can find a fault-free path of nearly optimal length with very high probability. The algorithm uses the adaptive binomial-tree to select a suitable dimension to route a node. We perform empirical analysis of our algorithm through simulations under a uniform node failure distribution and a clustered node failure distribution. The experimental results show that the algorithm successfully finds a fault-free path of nearly optimal length with the probability larger than 90 percent in a hypercube network that may contain up to 50 percent faulty nodes.

1 Introduction

Hypercube networks are among the most popular network models studied by researchers and adopted in many implementations of parallel computer systems, such as Intel iPSC, the nCUBE, the Connection Machine CM-2, and the SGI's Origin 2000 [1] and Origin 3000.

Much effort has been devoted to introduce realistic definitions to measure the networks' ability to tolerate faults, and to develop routing algorithms under a large number of faults. For example, Najjar et al [2] demonstrated that for the 10-cube, 33 percent of nodes can fail and the network can still remain connected with a probability of 99 percent. Gu and Peng [3] proposed an efficient global routing algorithm for a k-safe n-cube with up to $2^k(n-k)-1$ faulty nodes. Chen et al [4] also proposed a distributed routing algorithm for hypercube with up to 37.5% faulty nodes based on local subcube-connectivity.

In this paper, based on the divide-and-conquer and adaptive binomial-tree techniques, we propose an efficient fault-tolerant routing algorithm. For hypercube network that may contain a large fraction of faulty nodes, our algorithm finds a fault-free path for any two given nonfaulty nodes with very high probability. The algorithm uses the adaptive binomial-tree to select a suitable dimension to route a node. We perform empirical analysis of our algorithm through simulations under a variety of probability distributions of node failures. The experimental results show that the algorithm successfully finds a fault-free path of nearly optimal length with the probability larger than 90 percent in a hypercube network that may contain up to 50 percent faulty nodes.

2 Fault-tolerant Routing Algorithm

For node $s \in H_n$, let $s^{(i)}$ be the neighbor of s in i-dimension. Similarly, $s^{(i,j)}$ denotes $(s^{(i)})^{(j)}$. A fault-free k-binomial-tree rooted at a given node s is denoted as $T_k(s)$.

Algorithm Hypercube_Routing(n, s, t, max)
Input: n-cube, nonfaulty nodes $s = s_1 s_2 \ldots s_n$ and $t = t_1 t_2 \ldots t_n$,
max: the maximum size of binomial-tree.
Output: a fault-free path $P = (s \rightarrow t)$ or report *failure*
begin
$P = \emptyset$; $D = \{1, \ldots, n\}$; $w = s$;
for $i = 1$ to n **do**
if (there is a dimension $j \in D$ such that $w_j \neq t_j$)
$D = D - \{j\}$;
if ($w' = w_1 \ldots w_{j-1} \overline{w_j} w_{j+1} \ldots w_n$ is nonfaulty) $P = P : w$; $w = w'$;
else routed = false; $k = 0$;
while (routed = false) AND ($k < max$) **do**
(w', T, fail) = Binomial_Tree(n, w, t, j, k, D);
if ($w' \neq -1$) $P = P : T$; $w = w'$; routed = true;
else **if** (fail = true) **return** *failure*;
else $k = k + 1$;
return P; /* finish, path constructed */
end

We first partition H_n along with a dimension j into two $(n-1)$-cubes, H^0_{n-1} and H^1_{n-1}, such that s and t are separated. Assume $s \in H^0_{n-1}$ and $t \in H^1_{n-1}$. Then, we route s to a node $s' \in H^1_{n-1}$.

Intuitively, the routing algorithm can be described below. In the first run, if $s^{(j)}$ is nonfaulty then we are done, where j is the dimension on which we are routing; otherwise, we build a $k = 0$ binomial-tree $T_0(s) = \{s\}$ and try to find a fault-free path $s \rightarrow s'$ of length 2, $(s : s^{(i)} : s^{(i,j)})$, where $s^{(i)} \in H^0_{n-1}$ and $i \neq j$.

If we fail to find a fault-free path in the first run then let $k = k + 1$ and try to build a fault-free k-binomial-tree $T_k(s)$. If there exists a node $u \in T_k(s)$ such that $u^{(j)}$ is nonfaulty then we are done. Otherwise, for each node $u \in T_k(s)$, we

```
Procedure Binomial_Tree(n, w, t, j, k, D)
begin
    w' = −1; T = ∅; fail = false;
    if (k = 0)
        B = {w};
        reorder(w, t, D);
        if (there is an i ∈ D: u = w_1 ... w_{i−1} w̄_i w_{i+1} ... w_n is nonfaulty)
            if (v = u_1 ... u_{j−1} ū_j u_{j+1} ... u_n is nonfaulty)
                T = u; w' = v;
                return (w', T, fail);
        return (w', T, fail);
    N = ∅;
    for (each node s ∈ B) do                    /* 1. construct k-binomial-tree */
        reorder(s, t, D);
        if (there is an i ∈ D: u = s_1 ... s_{i−1} s̄_i s_{i+1} ... s_n ∉ (B ∪ N) is nonfaulty)
            if (v = u_1 ... u_{j−1} ū_j u_{j+1} ... u_n is nonfaulty)
                T = w → u; w' = v;
                return (w', T, fail);
            N = N ∪ u;
        else fail = true;
    for (each node s ∈ N) do                    /* 2. search k-binomial-tree */
        reorder(s, t, D);
        if (there is an i ∈ D: u = s_1 ... s_{i−1} s̄_i s_{i+1} ... s_n is nonfaulty)
            if (v = u_1 ... u_{j−1} ū_j u_{j+1} ... u_n is nonfaulty)
                T = w → u; w' = v;
                return (w', T, fail);
    B = B ∪ N;                                  /* constructing (k + 1)-binomial-tree */
    return (w', T, fail);                       /* w' = −1; T = ∅; fail = true or false */
end
```

try to find a node $u^{(i)}, i \neq j$, such that $u^{(i)}$ and $u^{(i,j)}$ are nonfaulty. If such u exists then we are done and $s' = u^{(i,j)}$. The algorithm terminates if the fault-free path $s \rightarrow s'$ is found or the algorithm fails to build $T_k(s)$.

To extend $T_{k-1}(s)$ to $T_k(s)$, we use a dimension set D to keep the dimensions that are not yet routed. Initially, $D(u) = \{1, \ldots, n\}$. Once a dimension j is routed, j is deleted from D. When we check the neighbors of a node in the binomial-tree to find a nonfaulty node to extend the binomial-tree, we want those nodes whose jth bit have the same value as the destination node t to be searched first. Therefore, we re-order D before it is used for selecting neighbors or finding new nodes. We call this *adaptive binomial-tree routing*.

The construction of $T_k(s)$ is as follows: We apply a tree traversal on $T_{k-1}(s)$. While $u \in T_{k-1}(s)$ is visited, we try to find a nonfaulty node $v \in \{u^{(i)} | i \in D(u)\}$. If such v exists then we include edge $(u : v)$ and node v in $T_k(s)$. Otherwise, the extension fails and the algorithm terminates unsuccessfully.

The running time of the routing algorithm is analyzed as follows: For every k, each iteration for finding $s \rightarrow s'$ takes $O(|T_k(s)| \times n)$ time. In the worst case, the time for finding a fault-free path $s \rightarrow s'$ or reporting a failure will take

$\sum_{k=0}^{max} O(|T_k(s)| \times n) = O(n)$, where max is the largest k we try for finding $s \rightarrow s'$. In all practical cases as we will show in the next section, we have $max \leq 4$. Then, the running time of the algorithm that performs binomial-tree routing $O(n)$ is $O(n^2)$, independent of the size of F.

3 Theoretical Analysis and Simulations

In this section, we first give a theoretical analysis on the successful routing rate of our algorithm under the uniform distribution of node failures. Then we show the simulation results of our algorithm under the uniform and clustered distributions of node failures.

The causes for the failure of the binomial-tree routing are twofolds: failure in finding a nonfaulty k-binomial-tree or failure in finding a fault-free path ($s \rightarrow s'$) using the found binomial-tree. For simplicity, we assume that k is fixed. Let p_f be the node failure probability. We calculate the successful routing rate p_s for the given values of n, p_f, and k. The formula for the approximate p_s is given in the following theorem.

Theorem 1 *Suppose that every node in the n-cube has an equal and independent failure probability p_f. For a given k, the probability of successful routing of the algorithm $p_s \approx \prod_{i=1}^{n/2}(1-p_f(1-(1-p_f)^2)^{n-i})$ if $k = 0$; otherwise, $p_s \approx \prod_{i=1}^{n/2}(1-p_f^{2^k}(1-(1-p_f)^2)^{2^k(n-i-k)})(1-p_f^{n-i+1})^{2^{k-1}}$.*

We have performed extensive experiments to study the performance of our routing algorithm based on the uniform distribution and the clustered distribution of node failures. We have done simulation for the n-cube with $n = 10, 15, 16$ and 20. For each n, we simulated the node failure probability $p_f = 10\% \sim 70\%$ with 10% increment. For each pair (n, p_f), we randomly picked up 10,000 pairs of nonfaulty nodes in H_n, and simulated our routing algorithm.

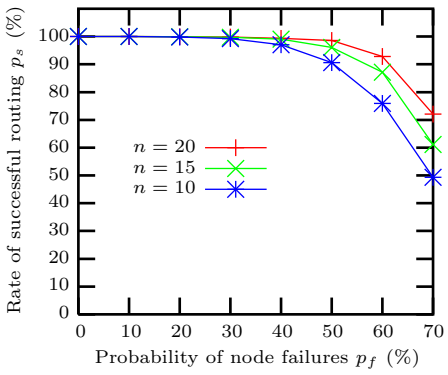

Fig. 1. Successful routing rate

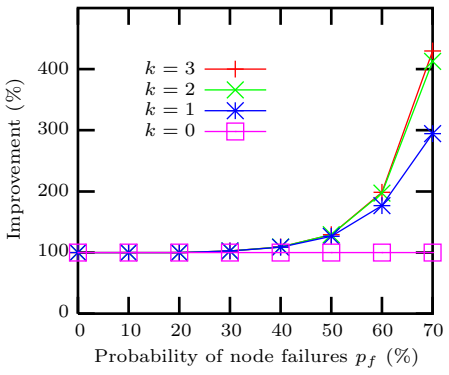

Fig. 2. Ratio of p_s for $max = 1$, 2, and 3 over $max = 0$ when $n = 16$

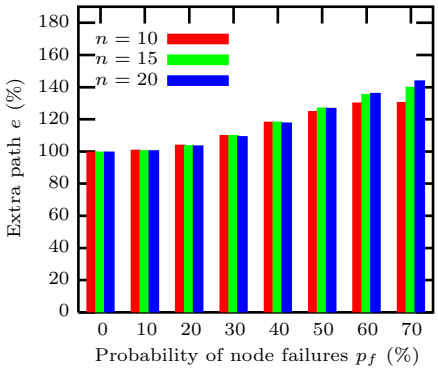

Fig. 3. Ratio of the max. length to $d(s,t)$

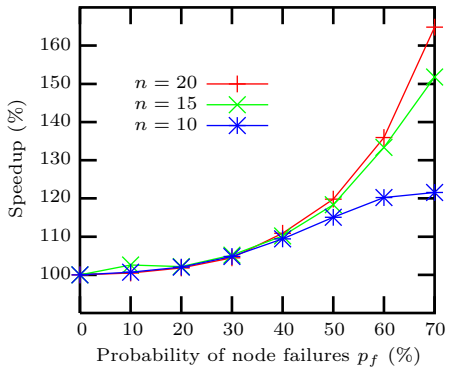

Fig. 4. Speedup of adaptive routing

From Figure 1, we can see that, when the node failure probability is not very large (i.e., $p_f \leq 20\%$), the successful routing rate p_s is nearly 100% ($\approx 99.9\%$). When the p_f is 30% or more, the p_s starts to drop slowly, and when the p_f is 60% or more, the p_s drops faster. However, the p_s still maintains at about 50% for $n = 10$, and 70% for $n = 20$. For $n = 10$ and $p_f = 30\%$, Chen's algorithm finds a fault-free path at the probability of 92% [4]. In contrast, our algorithm finds a fault-free path at the probability of 99.28% with $k = 2$, much higher than Chen's algorithm.

Figure 2 indicates the improvement on p_s over the increment on max (the upper bound of the binomial-tree to be checked) for $n = 16$. We let the values of max vary from 0 to 3. From the resulting data, we see the improvement is significant for max as p_f increases. For example, when $p_f = 70$, the p_s of $max = 3$ is 5 times of that of $max = 0$. However, for $max > 3$, the improvement on performance is very little and can be ignored. We conclude that setting $max = 2$ or $max = 3$ should be enough for all practical cases. Therefore, the running time of the algorithm is efficient since the constant is small. Figure 3 shows the average ratio (path_plus) of the lengths of the paths generated by the algorithm over the distance $d(s,t)$ for $n = 16$. The path_plus is nearly optimal. In all cases, the maximum length of the paths generated is bounded by $1.5 \times d(s,t)$. Figure 4 shows the performance improvement of the adaptive binomial-tree routing compared to the basic binomial-tree routing that does not reorder the dimensions of D. It can be seen that the speedup increases as the size of hypercube and the p_f increase.

We have also simulated on another probability distribution of node failures. The results show that the performance under the clustered distribution are similar to that under the uniform distribution. For $p_f > 20\%$, the clustered node failure distributions get a little bit better performance than the uniform distribution.

References

1. SGI: Origin2000 Rackmount Owner's Guide, http://techpubs.sgi.com/ (1997)

2. Najjar, W., Gaudiot, J.L.: Network resilience: A measure of network fault tolerance. IEEE Transactions on Computers **39** (1990) 174–181
3. Gu, Q.P., Peng, S.: Unicast in hypercubes with large number of faulty nodes. IEEE Transactions on Parallel and Distributed Systems **10** (1999) 964–975
4. Chen, J., Wang, G., Chen, S.: Locally subcube-connected hypercube networks: Theoretical analysis and experimental results. IEEE Transactions on Computers **51** (2002) 530–540