Proceedings of the ISCA 17th International Conference on Parallel and Distributed Computing Systems, San Francisco, California, Sep. 2004, pp222-228

Adaptive-Subcube Fault Tolerant Routing in Dual-Cube with Very Large Number of Faulty Nodes

Yamin Li and Shietung Peng
Department of Computer Science
Hosei University
Tokyo 184-8584 Japan

Wanming Chu
Department of Computer Hardware
University of Aizu
Aizu-Wakamatsu 965-8580 Japan

Abstract

The dual-cube is a newly proposed interconnection network for linking a large amount of nodes with low node degree. It uses low-dimensional hypercubes as building blocks and keeps the main desired properties of the hypercubes. In this paper, we give an efficient algorithm for fault tolerant routing in dual-cube networks with a large number of faulty nodes. Our algorithm uses the adaptive-subcube technique to select a suitable dimension to route a node. This technique not only increases the routing speed but also shortens the path and improves the successful routing rate. The experimental results show that, with high percentages of node failures, the algorithm can build routing paths with a very high probability. Our simulation results show that when a dual-cube with 32,768 nodes contains up to 20 percent faulty nodes, the success rate of constructing a fault-free path between any two nonfaulty nodes is 99.5 percent with a 4-subcube.

1 Introduction

As the size of computer networks increases continuously, the node failures are inevitable. Routing in computer networks with faults has been more important and has attracted considerable attention in the last decade. Hypercube is a popular network studied by researchers and adopted in many implementations of parallel computer systems, such as Intel iPSC, the nCUBE [3], the Connection Machine CM-2 [9], and the SGI's Origin 2000 (1996) [8] and Origin 3000 (2000).

In an n-dimensional hypercube, or n-cube for short, removing n neighbors of any node u will disconnect u from the other part of the network. Therefore, the fault tolerance of an n-dimensional hypercube is $n-1$.

This is a theoretical definition of hypercube's fault tolerance. In the realistic case, the probability that the n faulty nodes are exactly are the neighbors of a nonfaulty node is very low. Also, the ratio of the number of faulty nodes to the total number of nodes is too small to have a practical impact. Previous research have shown that a hypercube can tolerate a constant fraction of faulty nodes. For example, Najjar et al [7] 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 [2] proposed efficient routing algorithm for a k-safe n-cube with up to $2^k(n-k)-1$ faulty nodes. Chen et al [1] also proposed a distributed routing algorithm in hypercube with large amount of faulty nodes based on local subcube-connectivity.

Chen's algorithm routes two nonfaulty nodes, u and v, in hypercube with large amount of faulty nodes. The algorithm is based on the *local subcube-connectivity*: A $(k+1)$-cube can be divided to two k-subcubes. In each k-subcube, if the number of faulty nodes is less than half of the total number of node in the k-subcube, and all the nonfaulty nodes keep connected, then all the nonfaulty nodes in the $(k+1)$-cube also keep connected. The job is to find a minimum k. Then we say the n-cube is a k-subcube connected network. The algorithm routes each address bit of u to the corresponding bit value of v within a k*-subcube*. Therefore, the complexity of the algorithm is $O(n2^k)$. If the faulty nodes are equally distributed, the k will be very small and the 2^k can be treat as a constant value. In such case, the algorithm will run at $O(n)$. In particular, if there is no faulty node at all, then $k=0$.

A dual-cube is a recently proposed network [6]. A k-dimensional dual-cube consists of 2^k clusters and each cluster is a $(k-1)$-dimensional hypercube. In this paper, we use m*-dual-cube* to denote an k-dimensional dual-cube where $m=k-1$. Then the total number of nodes in an m-dual-cube will be 2^{2m+1}. With the same node degree $m+1$, the dual-cube contains 2^m times more nodes than the hypercube and, with the same amount of nodes 2^n where $n=2m+1$, the dual-cube has approximately 50% less links than the hypercube.

In this paper, based on the algorithm presented in [1], we give efficient algorithms for fault tolerant routing in dual-cube networks with a large number of faulty nodes. The algorithms are local-information-based in the sense that each node knows only its neighbors' status and no global information of the network is required by the algorithm. The algorithm runs in linear time and builds routing paths of nearly optimal length. We used a *adaptive-subcube* technique in the algorithm's implementation that selects some suitable dimensions to form a subcube. This technique speeds up the routing speed, shortens the path and improves the probability of the successful path construction. Our simulation results show that, with high percentages of node failures, the algorithm can build routing paths with a very high probability.

The remainder of the paper is organized as following. In the next section, we introduce the dual-cube structure briefly. Section 3 gives the detailed refinement of the fault-tolerant routing algorithm in hypercube. Section 4 proposes the new fault-tolerant routing algorithm in dual-cube. Section 5 discusses the time complexity of the algorithm. Section 6 does the simulations and presents the experimental results. The final section concludes the paper.

2 The Dual-Cube Network

A dual-cube uses hypercubes as basic components. Each hypercube component is referred to as a *cluster*. Assume that the number of nodes in a cluster is 2^m. In a dual-cube, there are two *classes* with each class consisting of 2^m clusters. The total number of nodes in a dual-cube m-dual-cube is $2^m \times 2^m \times 2$, or 2^{2m+1}. Each node in a m-dual-cube has $m+1$ links: m links are used within cluster to construct an m-cube and a single link is used to connect a node in a cluster of the other class. There is no link between the clusters of the same class. If two nodes are in one cluster, or in two clusters of distinct classes, the distance between the two nodes is equal to its *Hamming distance* (the number of bits where the addresses of the two nodes have different values). Otherwise, it is equal to the Hamming distance plus two: one for entering a cluster of the other class and one for leaving.

An $(m+1)$-connected dual-cube m-dual-cube is an undirected graph on the node set $\{0,1\}^{2m+1}$ and there is a link between two nodes $u = (u_{2m} \ldots u_0)$ and $v = (v_{2m} \ldots v_0)$ if and only if the following conditions are satisfied:

1. u and v differ exactly in one bit position i,
2. if $0 \leq i \leq m-1$ then $u_{2m} = v_{2m} = 0$ and
3. if $m \leq i \leq 2m-1$ then $u_{2m} = v_{2m} = 1$.

The link connecting two nodes in two clusters of distinct classes is called *cross-link*. In the other word, $e = (u : v)$ is a cross-link if and only if u and v differ in the leftmost bit position.

Each node in a m-dual-cube is identified by a unique $(2m+1)$-bit number, an *id*. Each *id* contains three parts: 1-bit *class_id*, m-bit *cluster_id* and m-bit *node_id*. In the following discussion, we use *id* = (*class_id*, *cluster_id*, *node_id*) to denote the node address. The bit-position of *cluster_id* and *node_id* depends on the value of *class_id*. If *class_id* $= 0$ (*class_id* $= 1$), then *node_id* (*cluster_id*) is the rightmost m bits and *cluster_id* (*node_id*) is the next (to the left) m bits.

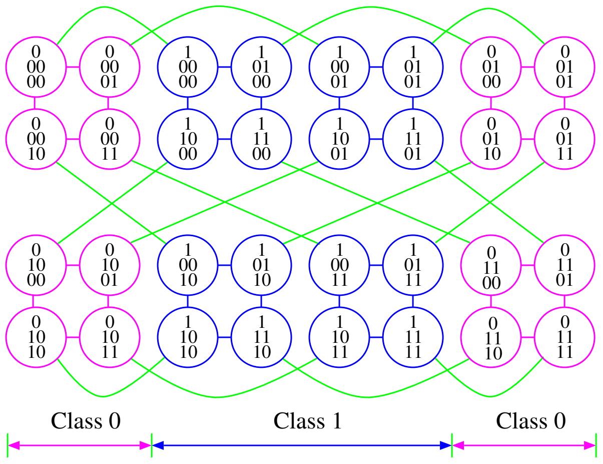

Figure 1: An 2-dual-cube

Figure 1 depicts a DC(2) network. In each node, *class_id* is shown at the top position. For the nodes of class 0 (class 1), *node_id* (*cluster_id*) is shown at the bottom and *cluster_id* (*node_id*) is shown at the middle.

The dual-cube has a binary presentation of nodes, similar to a hypercube, in which two nodes are connected by a link only if their addresses differ in one bit position. This feature is the key for designing efficient routing and communication algorithms in dual-cube. Another important feature of a dual-cube is that, within the given bound to the number of links per node, say $m+1$, the network can have up to 2^{2m+1} nodes. The m-dual-cube topological properties are given in [4] and the collective communication schemes in m-dual-cube can be found in [5].

3 Fault Tolerant Routing in Dual-cube

We first show an algorithm for routing in hypercube with faulty nodes. It will be used as a subroutine for routing in dual-cube. A path consists of a sequence of nodes where two adjacent nodes are connected by a link. We use $(u : v)$ to denote a link connecting nodes u and v, and $(u \rightarrow v)$ or $(v_1 : v_2 : \ldots : v_r)$ to denote a path or a cycle.

```
Algorithm 1 (routing_cube ($m, u, v, k$, u_class_id, u_cluster_id))
Input routing two nonfaulty nodes $u = u_1u_2 \dots u_m$ and $v = v_1v_2 \dots v_m$ in $m$-cube with $k$-subcube;
    u_class_id and u_cluster_id are class_id and cluster_id of the $m$-cube, respectively;
Output ($w$, PC): if success, $w = v$ and PC = $(u \to v) - \{v\}$; or report failure with $w = -1$;
Begin
    PC = empty;
    $D = \{1, \dots, m\}$;                 /* $D$: a set of dimensions which can be used for routing */
    $w = u$;                                /* $w = w_1w_2 \dots w_m$ */
    for $i = 1$ to $m - k$ do {
        if (there is a dimension $j \in D$ such that $w_j \neq v_j$) {          /* i.e., $w_j = \overline{v_j}$ */
            $D = D - \{j\}$;    /* delete $j$ from $D$, $j$ is the dimension along with which we are routing in current iteration */
            ($t$, C) = routing_k_subcube ($m, w, v, k, j, D$, 1, u_class_id, u_cluster_id, $*$);    /* route $j$th bit of $w$ */
            if ($t \neq -1$) {
                PC = PC : C;                 /* $j$th bit routed, add path C to PC */
                $w = t$;
            } else { return ($-1$, PC); }    /* cannot route in $k$-subcube */
        }
    }
    if ($w \neq v$) {                        /* route the last $k$ bits */
        ($t$, C) = routing_k_subcube ($m, w, v, k, *, D$, 0, u_class_id, u_cluster_id, $*$);    /* route $w \to v$ */
        if ($t = v$) {
            PC = PC : C;                     /* routed; add path to PC */
            $w = t$;
        } else { return ($-1$, PC); }        /* cannot route in $k$-subcube */
    }
    return ($w$, PC);                        /* path constructed */
End
```

Chan [1] introduced the concept of *locally subcube-connected hypercube*. An m-cube is locally k-subcube-connected network if, in each k-subcube, $k \leq m$, less than half of the nodes in k-subcube are faulty and the nonfaulty nodes of the k-subcube make a connected graph.

We use a binary string $b_1b_2 \dots b_m$ to label a node where each b_i is either 0 or 1. A k-subcube has 2^k nodes whose addresses differ in the right-most k bits. The address of a node in the k-subcube is of the form $b_1b_2 \dots b_{m-k}x_{m-k+1} \dots x_m$, where $b_1b_2 \dots b_{m-k}$ is the $(m-k)$-bit *id* of the k-subcube and $x_{m-k+1} \dots x_m$ is the k-bit *id* within the k-subcube.

Suppose we have two nonfaulty nodes $u = u_1u_2 \dots u_m$ and $v = v_1v_2 \dots v_m$. To construct a path $u \to v$, we route each bit u_i of u to v_i, for $i = 1, 2, \dots, m-k$. After $i-1$ bits are routed, a node w in the path has an address of $w = w_1w_2 \dots w_m$ with $w_j = v_j$ for $j = 1, 2, \dots i-1$. If the node $w_1 \dots w_{i-1}v_iw_{i+1} \dots w_m$ is faulty when routing the ith bit, we try to route w to a node

$$s = w_1 \dots w_{i-1}w_iw_{i+1} \dots w_{m-k}x_{m-k+1} \dots x_m$$

in the k-subcube. If

$$s' = w_1 \dots w_{i-1}v_iw_{i+1} \dots w_{m-k}x_{m-k+1} \dots x_m$$

is nonfaulty, then the ith bit is routed and the path is ($u \to w \to s : s'$). Let $w = s'$ and route the next bit until the first $m-k$ bits are routed. Finally, we route w to v in the k-subcube.

The algorithm described above uses the "basic" k-subcube of form $b_1b_2 \dots b_{m-k} * * \dots *$. We revised this basic algorithm to allow to use other k bits than the "most-right" k bits for routing in k-subcube. We divide m bits of a node address into two sets. The 1st set contains those bits already routed and the 2nd set contains the other bits which can be changed for routing. In the k-subcube routing, we select k bits from the 2nd set in such a way that those bits on which the current node w and the destination node v have different values will be used first. We call this method *adaptive-subcube routing*. The benefit of adaptive-subcube routing is that when we route the ith bit, the bits to be routed later can be done in the current iteration. This method not only speeds up the routing but also shortens the path and consequently increase the probability of the successful routing. The adaptive-subcube routing gets a significant performance improvement which we will show in Section 4.

The algorithm for construct $u \to v$ in an m-cube is shown in Algorithm 1 (routing_cube). Note that the al-

```
Algorithm 2 (routing_k_subcube ($m, u, v, k, j, D, c$, u_class_id, u_cluster_id, d_cluster_id))
Input two nodes $u = u_1u_2 \dots u_m$ and $v = v_1v_2 \dots v_m$ in $m$-cube, routing with $k$-subcube;
    u_class_id and u_cluster_id are class_id and cluster_id of the $m$-cube, respectively;
    $D$: a dimension set that can be used for routing;
    if $c = 0$, routing $u \rightarrow v$; return node $r = v$;
    if $c = 1$, routing $j$th bit of $u$ with $k$-subcube; return node $r$, where $r_h = v_h$ for $h \in (\{1, \dots, n\} - D)$ (including $j$);
    if $c = 2$, finding $u \rightarrow s : s' \rightarrow t : t'$; return node $r = t'$;
    if $c = 3$, finding $u \rightarrow w : w' \rightarrow s : s' \rightarrow t : t'$; return node $r = t'$;
    d_cluster_id is the cluster_id of the final destination node, used in $c = 2$ and $c = 3$;
Output: ($r$, C); $r$: as described above. if cannot find $r$, return $r = -1$;
    if $r$ is found, C $= (u \rightarrow r) - \{r\}$, a path $(u \rightarrow r)$ but does not contain $r$;
Begin
    $r = -1$; C = empty;
    for each node $x \in m$-subcube do { color[$x$] = WHITE; }
    $Q$ = empty;  color[$u$] = GRAY;  enqueue($Q$, $u$);  prede[$u$] = NULL;
    while ($Q \neq$ empty)) {
        $x$ = head[$Q$];  $s = x$;  $w = x$;                                        /* $x = x_1x_2 \dots x_m$ */
        case ($c = 0$)                                                              /* routing $u \rightarrow v$ */
            if ($x = v$) { C $= u \rightarrow x$; return ($v$, C); }
        case ($c = 1$)                                              /* routing $j$th bit of $u$ with $k$-subcube */
            $r = x_1x_2 \dots x_{j-1}\overline{x_j}x_{j+1} \dots x_m$;
            if (getid(u_class_id, u_cluster_id, $r$) is nonfaulty) { C $= u \rightarrow x$; return ($r$, C); }
        case ($c = 2$)                                                  /* finding $u \rightarrow s : s' \rightarrow t : t'$ */
            if (getid($\overline{u\_class\_id}$, $s$, u_cluster_id) is nonfaulty) {                  /* $s'$ is nonfaulty */
                ($t$, PC) = routing_cube ($m$, u_cluster_id, d_cluster_id, $k$, $\overline{class\_id}$, $s$);      /* PC $= s' \rightarrow t$ */
                if ($t \neq -1$) AND (getid(u_class_id, d_cluster_id, $s$) is nonfaulty) {               /* $t : t'$ */
                    C $= u \rightarrow s$ : PC;
                    return ($s$, C);                                    /* C $= u \rightarrow s : s' \rightarrow t : t'$ */
                }
            }
        case ($c = 3$)                                      /* finding $u \rightarrow w : w' \rightarrow s : s' \rightarrow t : t'$ */
            if (getid($\overline{class\_id}$, $w$, u_cluster_id) is nonfaulty) {                      /* $w'$ is nonfaulty */
                $D = \{1, \dots, m\}$;                   /* $D$: a set of dimensions which can be used for routing */
                ($t'$, C) = routing_k_subcube ($m$, u_cluster_id, $*, k, *, D$, 2, $\overline{class\_id}$, $w$, d_cluster_id);
                if ($t' \neq -1$)) {                                                                 /* $t : t'$ */
                    C $= u \rightarrow w$ : C;
                    return ($t'$, C);                           /* C $= u \rightarrow w : w' \rightarrow s : s' \rightarrow t : t'$ */
                }
            }
        $i = 0$;                                                               /* add neighbors of $x$ to queue */
        reorder($m, x, v, D$);   /* reorder $D$ so that the dimensions at which $x$ and $v$ have different values are checked first */
        while ($i < k$) {
            $y = x$ xor $2^{D[i]}$;                                /* select a neighbor of $x$ using re-ordered $D$ */
            if (getid(u_class_id, u_cluster_id, $y$) is nonfaulty) AND (color[$y$] = WHITE) {
                enqueue($Q$, $y$);   prede[$y$] $= x$;
            }
            $i = i + 1$;
        }
        dequeue($Q$);   color[$x$] = BLACK;
    }
    return ($r$, C);
End
```

```
Algorithm 3 (routing_dualcube ($m$, $u$, $v$, $k$))
**Input** routing two nonfaulty nodes $u$ and $v$ in an $m$-dual-cube with $k$-subcube;
    $u$ = (u_class_id, u_cluster_id, u_node_id);
    $v$ = (v_class_id, v_cluster_id, v_node_id).
**Output**: a path $u \to v$ or report failure
**Begin**
    if (u_class_id = v_class_id) AND (u_cluster_id = v_cluster_id) {    /* Case 1: $u$ and $v$ are in same cluster, PD = $u \to v$ */
        return routing_cube ($m$, u_node_id, v_node_id, $k$, u_class_id, u_cluster_id);
    } else {
        $D = \{1, \ldots, m\}$;                    /* $D$: a set of dimensions which can be used for routing */
        if (u_class_id = v_class_id) {            /* Case 2: $u$ and $v$ are of same class, path = $u \to s : s' \to t : t' \to v$ */
            ($t'$, PC1) = routing_k_subcube ($m$, u_node_id, $*, k, *, D$, 2, u_class_id, u_cluster_id, v_cluster_id);
        } else {            /* Case 3: shortest path = $u \to t : t' \to v$ or path = $u \to w : w' \to s : s' \to t : t' \to v$ */
            ($t'$, PC1) = routing_cube ($m$, u_node_id, v_cluster_id, $k$, u_class_id, u_cluster_id);        /* shortest path */
            if ($t' = -1$)
                ($t'$, PC1) = routing_k_subcube ($m$, u_node_id, $*, k, *, D$, 3, u_class_id, u_cluster_id, v_cluster_id);
            }
        }
        if ($t' \neq -1$) {
            ($w$, PC2) = routing_cube ($m, t'$, v_node_id, $k$, v_class_id, v_cluster_id);
            if ($w \neq -1$)  return (PC1 : PC2);                /* path constructed */
        }
        return failure;
    }
End
```

gorithm will be used for fault tolerant routing in dual-cube. Nodes u and v are identified by the *node_id* field in the dual-cube. Therefore, when routing in k-subcube and checking the faultiness of a node, a $(2m+1)$-bit address should be used. This is done by getid(*class_id*, *cluster_id*, *node_id*). See algorithm 2 (routing_k_subcube).

Algorithm 2 uses a *breadth first search* strategy. There are four cases in Algorithm 2. For the hypercube routing, Algorithm 1 uses case 0 and case 1. The case 1 routes a bit j of node w and the case 0 routes w to v. The cases 3 and 4 are for dual-cube routing which will be discussed in the next section. In order to adopt the adaptive-subcube routing, we use D to keep the dimensions which contains the bits not yet routed. Once a bit i is routed, i is deleted from D. The breadth first search algorithm uses a queue to hold nodes and adds the neighbors of the head node into the queue. Only the neighbors in dimensions in D are allowed to be added to queue. Suppose we are routing the ith dimension of a node w. We want those neighbors whose ith bit have the same value as the destination node v to be searched first. Therefore, we re-order D before it is used for selecting neighbors of the head node in queue.

The routing algorithm in dual-cube is shown in Algorithm 3 (routing_dualcube (m, u, v)). We want to construct a path from node u and node v. There are three cases, distinguished by the locations of the nodes u and v. In Case 1, u and v are in a same cluster. This is the simplest case and we apply the routing_cube algorithm directly. Note that if the routing within the cluster is not successful, there may exist a path by going through other clusters.

In Case 2, u and v are in different clusters of the same class. The path $u \to v$ must go through a cluster of the other class. We route u to a node, say s, such that there is path $s' \to t$ and t' in the cluster of v is nonfaulty. Then we route t' to v. Note that s ans u may be the same node.

In particular, assume that u and v are of class 0, then the routing path is shown below with full address format.

		class id	cluster id	node id
u	=	(0,	*u_cluster_id*,	$\underline{u\_node\_id}$)
finding s	=	($\underline{0}$,	*u_cluster_id*,	***s_node_id***)
s'	=	(**1**,	*s_node_id*,	$\underline{u\_cluster\_id}$)
routing to t	=	($\underline{1}$,	*s_node_id*,	***v_cluster_id***)
t'	=	(**0**,	*v_cluster_id*,	$\underline{s\_node\_id}$)
routing to v	=	(0,	*v_cluster_id*,	***v_node_id***)

The algorithm first calls "routing_k_subcube" (Algorithm 2) to find a node s and route s' to t (the case 2 in Algorithm 2). Then it calls "routing_cube" (Algorithm 1) to route t' to v.

Assume that u and v follow the uniform distribution. Let p_i be the probability that nodes u and v are in Case i, for $i = 1, 2,$ and 3. Then $p_1 = 2^m/2^{2m+1}$, where 2^{2m+1} is the total number of nodes in the m-dual-cube; $p_2 = 1/2 - p_1$; and $p_3 = 1/2$. Let t_i be the time required for Case i routing. Then the expected total time $t = \sum_{i=1}^{3} t_i \times p_i$.

We have $t_1 = m2^k$ assuming that the m-cube is locally k-subcube connected; In Case 2, the first step of the algorithm is to find a node s in k-subcube such that s' is nonfaulty. That is, this step routes u in the most-left bit (class_id). It takes 2^k time. Then in the second step, the algorithm routes s' to t in the cluster of s' (an m-cube) and check the faultiness of the node t'. It takes $m2^k$ time. In the last step, it routes t' to v in the m-cube the node v is located. Therefore, $t_2 = 2^k m2^k + m2^k$.

In Case 3, the algorithm tries to route the first part (node_id field in an m-cube) of the shortest path. It takes $m2^k$ time. Let $p_{shortest}$ be the probability the shortest path is constructed. If it is not successful, the algorithm routes u to v by going through other two clusters. We first route u in the class_id bit in the k-subcube. It takes 2^k time. If it fails, then algorithm stops. If it successes, the rest steps are just the same as that of Case 2. Therefore, $t_3 = m2^k + (1 - p_{shortest})(2^k + 2^k m2^k) + m2^k \le m2^k + 2^k + 2^k m2^k + m2^k$.

The running time of the algorithm, as discussed above, is bounded by $O(\sum_{i=1}^{3} t_i \times p_i) = O(m2^{2k})$. Compared to the running time in hypercube, the additional 2^k item comes from the time for finding s in Case 2 or Case 3.

4 Experimental Results

We have performed the simulations to verify the efficiency of the proposed algorithm based on uniform probability distribution of node failures. We assume that each node has an equal and independent failure probability p_f. Seven m-dual-cubes ($m = 3, 4, 5, 6, 7, 8, 9$) with $k = 3, 4, 5,$ and m are simulated. For each dual-cube, we change the node failure probability p_f from 0% to 90%, steped by 10%. We tested 10,000 times to get the average results. Both the fixed-subcube and the adaptive-subcube versions are simulated.

From the analysis of the previous section, we knew that the running time of the algorithm is bounded by $O(m2^{2k})$. It is expected that the path can be constructed with high probability even with a small k-subcube. Fig. 2 shows the performance improvement of the adaptive-subcube version compared to the fixed-subcube version.

$$Speedup = \frac{\text{Routing success rate with adaptive-subcube}}{\text{Routing success rate with fixed-subcube}}$$

It can be seen that the speedup increases as the size of dual-cube increases. More importantly, this technique has

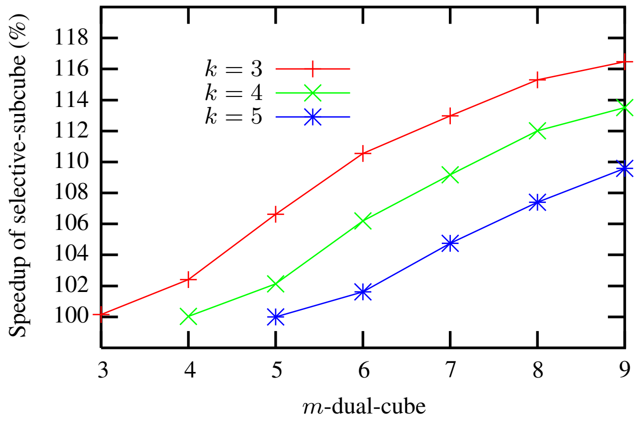

Figure 2: Speedup of adaptive-subcube

the obvious performance improvement on small k, which is what we expected. In what follows, we only show the performance of routing with the adaptive-subcube.

The simulations were done for $m = 5, 7,$ and 9, and for $k = 3, 4,$ and 5. Let $p_f(\%)$ be the node failure probability. We tested $p_f = 0$ to 90 percent, stepped by 10 percent. For a given p_f, the expected total number of faulty nodes in an m-dual-cube is $2^{2m+1} \times p_f$. Let $p_s(\%)$ be the ratio of the number of times in which a routing path is successfully constructed for a given pair of nodes by our algorithm over the total number of tested pairs of nodes. Let $e_p(\%)$ be the average ratio of the length of the constructed routing path over the length of shortest path of the given two nodes.

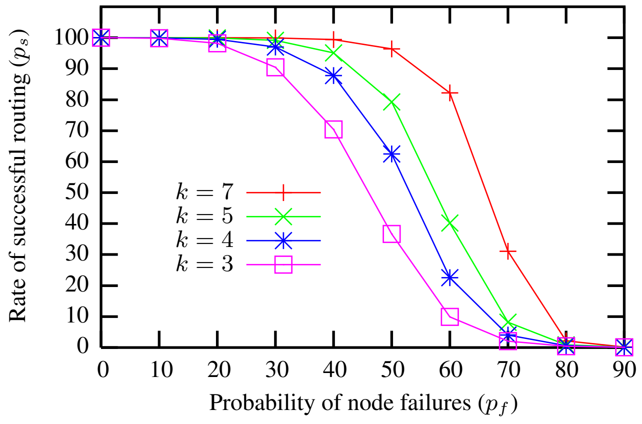

Figure 3: Successful routing rate ($m = 7$)

Fig. 3 depicts the p_s for routing in a 7-dual-cube (32,768 nodes). As the k becomes larger, the p_s increases. This is because that a larger dimension k of k-subcubes can make better connectivity of the dual-cubes. We can see that p_s becomes low very quickly when the p_f is larger than 40%.

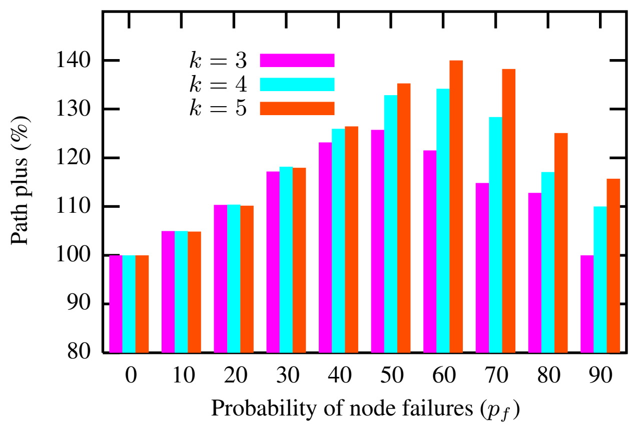

Figure 4: Ratio of path plus ($m = 7$)

We simulated 7 dual-cubes with different m and we found that, for a given k, the dual-cubes with different sizes (m) have almost the same performance.

Fig. 4 depicts the $e_p(\%)$ for a 7-dual-cube. When the p_f is larger than 50% ($k = 3$), or 60% ($k = 4$ and 5), the $e_p(\%)$ decreases as the p_f increases. The reason is that those routings that require a large number of extra nodes fail; while the figure shows only the successful cases.

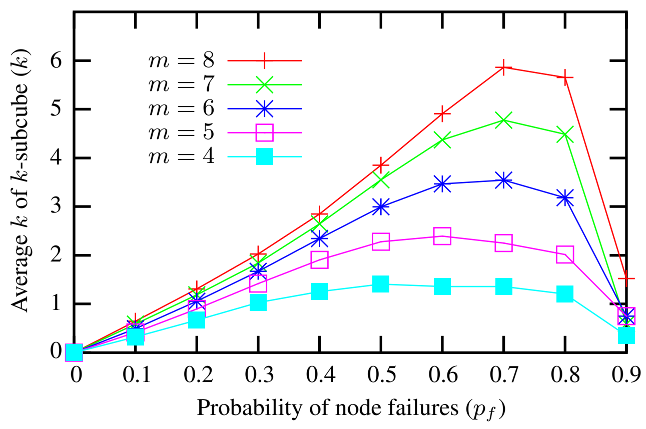

Figure 5: Average k of k-subcube

The results presented above are obtained by a fixed k for routing in the k-subcube. We also developed another program that starts from $k = 0$ and automatically increases k until a path is successfully constructed or k reached its maximum value m. Fig. 5 gives the average k of the k-subcube for dual-cubes with $m = 4,\ 5,\ 6,\ 7$ and 8. The k increases as the p_f increases but, depending on m, after p_f reaches to a value between 0.6 and 0.7, it starts to decrease. Again, this is because that those routings requiring larger k failed due to the high probability of node failures.

5 Concluding Remarks

In this paper, we gave a fault-tolerant routing algorithm in dual-cube with a large amount of faulty nodes. The algorithm requires only local information about the status of failures and runs at nearly linear time. The simulations were also performed. The results show that the dual-cube has excellent capacity for fault-tolerant routing with a large amount of faulty nodes. However, when the probability of node failures is above 30%, the rate for successful routing drops fast. It is interesting to find other efficient routing algorithm in dual-cube that can tolerate even higher node failure rate with an acceptable successful routing rate.

References

[1] Jianer Chen, Guojun Wang, and Songqiao Chen. Locally subcube-connected hypercube networks: Theoretical analysis and experimental results. *IEEE Transactions on Computers*, 51(5):530–540, May 2002.

[2] Q-P Gu and S. Peng. Unicast in hypercubes with large number of faulty nodes. *IEEE Transactions on Parallel and Distributed Systems*, 10:964–975, October 1999.

[3] J. P. Hayes and T. N. Mudge. Hypercube supercomputers. *Proc. IEEE*, 17(12):1829–1841, Dec. 1989.

[4] Y. Li and S. Peng. Dual-cubes: a new interconnection network for high-performance computer clusters. In *Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture*, pages 51–57, ChiaYi, Taiwan, December 2000.

[5] Y. Li, S. Peng, and W. Chu. Efficient collective communications in dual-cube. In *Proceedings of the Thirteen IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 266–271, Anaheim, USA, Aug. 2001.

[6] Yamin Li, Shietung Peng, and Wanming Chu. Efficient collective communications in dual-cube. *The Journal of Supercomputing*, 28(1):71–90, April 2004.

[7] W. Najjar and J. L. Gaudiot. Network resilience: A measure of network fault tolerance. *IEEE Transactions on Computers*, 39(2):174–181, Feb. 1990.

[8] SGI. *Origin2000 Rackmount Owner's Guide, 007-3456-003*. http://techpubs.sgi.com/, 1997.

[9] L. W. Tucker and G. G. Robertson. Architecture and applications of the connection machine. *IEEE Computer*, 21:26–38, August 1988.