

## Short Paper

---

# Dynamic Reconfiguration of Complete Binary Trees in Faulty Hypercubes

CHUI-CHENG CHEN

*Department of Information Management  
Southern Taiwan University of Technology  
Tainan, 710 Taiwan*

In this paper, we present a method for reconfiguring dynamically a complete binary tree in faulty hypercubes. First, we use a dynamic algorithm to reconfigure a complete binary tree of height  $h$  ( $h \geq 0$ ) in an  $(h + 1)$ -dimensional faulty hypercube. If there is a faulty node in the hypercube, both the *dilation* and *congestion* values are 2 after reconfiguration. If there are two faulty nodes in the hypercube, both the *dilation* and *congestion* values are 3 after reconfiguration. If there are more than two faulty nodes in the hypercube, we impose a constraint on the faulty node type, and both the *dilation* and *congestion* values are 3 after reconfiguration. Then we reconfigure a complete binary tree of height  $h$  in an  $(h + 2)$ -dimensional hypercube with at most  $2^{h+1} - 1$  nodes, and the *dilation* and *congestion* values are, respectively, 2 and 1 after reconfiguration. The number of affected nodes is minimized following reconfiguration.

**Keywords:** reconfiguration, complete binary tree, hypercube, faulty node, embedding

## 1. INTRODUCTION

The hypercube is one of the most effective as well as popular network architectures for parallel machines. The hypercube offers a rich interconnection topology, a recursive structure, and a small diameter. The structure of a hypercube can simulate many computational structures, such as arrays, binary trees, and meshes of trees, with only small constant factor slowdown [1].

Over the years, tree topologies have been designed to describe many computations, for example, searching, sorting, and merging problems [2, 3]. Particularly interesting among these trees is the complete binary tree, which is a natural computational structure for parallel algorithms, such as the “divide-and-conquer” type of algorithm [4].

Many researches have discussed the embedding of binary trees into hypercubes [1, 5-10]. In [1, 5], it was proven that a double-rooted complete binary tree of height  $h$  ( $h \geq 0$ ), denoted by  $DT_h$ , which is a complete binary tree with its root replaced by a path of length two, is a subgraph of an  $(h + 1)$ -dimensional hypercube, denoted by  $H_{h+1}$ . In [6, 7], it was shown that a complete binary tree of height  $h$ ,  $T_h$ , which has  $2^{h+1} - 1$  nodes, can be

embedded into  $H_{h+2}$  so that the adjacency of  $T_h$  is preserved. There exists no one-to-one node embedding of  $T_h$  into  $H_{h+1}$ , and the adjacency of  $T_h$  is preserved [6]. Wagner [8] described the embedding of a binary tree of height  $h$  into  $H_h$ ; this binary tree is complete for the first  $h - 2$  levels. Wu *et al.* [9] presented the embedding of binomial trees in hypercubes with link faults. In [10-13], the authors provided a method that can be used to reconfigure binary trees in faulty hypercubes.

The purpose of this paper is to present a method for reconfiguring dynamically a complete binary tree in a hypercube with faulty nodes. First, we discuss how one can reconfigure  $T_h$  in an  $(h + 1)$ -dimensional hypercube with faulty nodes, and then we discuss how one can reconfigure  $T_h$  in an  $(h + 2)$ -dimensional hypercube with at most  $2^{h+1} - 1$  faulty nodes. It is found that the number of affected nodes is minimized after fault recovery.

The remaining sections are organized as follows. Section 2 gives the notations and definitions used in this paper. In section 3, we show how one can reconfigure  $T_h$  in  $H_{h+1}$  with one or more faulty nodes. A free node of the hypercube is assigned to recover one faulty node, and the leaf nodes of  $T_h$  are embedded into other faulty nodes. In section 4, we give an algorithm for reconfiguring  $T_h$  in  $H_{h+2}$  with at most  $2^{h+1} - 1$  faulty nodes, where the *dilation* and *congestion* values of reconfigurable embedding are respectively 2 and 1, and the number of affected nodes is minimized after fault recovery. Finally, a conclusion is given in section 5.

## 2. PRELIMINARIES

The root of the complete binary tree of height  $h$ ,  $T_h$ , is in level 0, two nodes are in level 1, four nodes are in level 2,  $2^i$  nodes are in level  $i$ , *etc.*, and the total number of  $T_h$  is  $2^{h+1} - 1$ , where  $h \geq 0$ . The  $n$ -dimensional hypercube,  $H_n$ , has  $2^n$  nodes. These nodes of  $H_n$  are labeled  $\{0, 1, 2, \dots, 2^n - 1\}$  with binary numbers. Two nodes in the hypercube are linked with an edge if and only if their binary numbers differ by a single bit. The *Hamming distance* is the number of different bits between two nodes. To conveniently describe the embedding, we use two colors, black and white, to correspond to the binary number. If the node has an even number of 1's, it is black. Otherwise, it is white. Since the hypercube has a perfect matching,  $H_n$  has  $2^{n-1}$  black nodes and  $2^{n-1}$  white nodes.

The cost of one-to-one node embedding of a *guest graph* into a *host graph* is measured in terms of *dilation* and *congestion*. The dilation of an edge of the *guest graph* is the length of embedded path of the *host graph*. The *dilation* of an embedding is the maximum dilation over all the edges of the *guest graph*. The *congestion* of an edge of the *host graph* is the number of edges of the *guest graph* that are embedded using the same edge of the *host graph*. The *congestion* of an embedding is the maximum congestion over all the edges of the *host graph*. Hence, we have to consider the tradeoff between the *dilation* and *congestion* of an embedding.

The faulty model of the hypercube is defined as follows [14, 15].

- (1) The computational part of a faulty node is not utilized, while its links are fault-free.
- (2) A free node is not assigned initially; it can be used to recover faults but can not be reused to recover any other faults later.

- (3) The task of the faulty node is allowed to migrate to the free node.
- (4) The faulty diagnosis mechanisms are assumed to be fault-free.

Hence, all the free nodes can be used to recover from faults in the hypercube.

### 3. RECONFIGURING $T_h$ IN A FAULTY $H_{h+1}$

In this section, we discuss how one can reconfigure  $T_h$  in a faulty  $H_{h+1}$ .  $T_h$  can be embedded into  $H_{h+1}$  with *dilation 2* with a remaining free node in  $H_{h+1}$  [6]. When an arbitrary faulty node occurs in  $H_{h+1}$ , we can reconfigure  $T_h$  in  $H_{h+1}$  since the hypercube is symmetrical, and the reconfiguration results in all the nonfaulty nodes of  $T_h$  will be affected. We have to consider how to reduce the overhead of data communication after fault recovery; that is, the number of affected nodes has to be as small as possible after reconfiguration. Therefore, we present below a dynamic algorithm that can be used to reconfigure the complete binary tree in a faulty hypercube.

**Theorem 1**  $T_h$  can be reconfigured dynamically to embed into  $H_{h+1}$  with *dilation 2* and *congestion 2* when an arbitrary faulty node occurs in  $H_{h+1}$ .

**Proof:** First, we construct  $T_h$  from  $DT_h$ . The *dilation* value is 2, and the *congestion* value is 1 for such a construction of  $T_h$  in  $H_{h+1}$  (see Fig. 1) [1]. When an arbitrary node becomes faulty in  $H_{h+1}$ , there are two cases that need to be considered as follows.

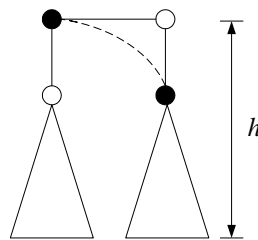


Fig. 1. Construction of  $T_h$  from  $DT_h$ .

- Case 1.** If the faulty node is one of either of the roots of  $DT_h$ , we let the nonfaulty root become the root of  $T_h$ , and the *dilation* and *congestion* are not altered.
- Case 2.** In this case, the faulty node occurs in the leaf nodes or the internal nodes of  $T_h$ . Without loss of generality, assume that the faulty node is in the left subtree of root  $r_l$  of  $DT_h$ . The path from the faulty node to root  $r_l$  has to be modified; that is, let  $r_r$  become the root of  $T_h$ , and let the nodes of the path be re-embedded into their parent nodes. If the faulty node occurs in the leaf nodes of  $T_h$ , each node of the path links its two sons using one edge whose dilation value is 2 and the other edge whose dilation value is 1 (see Fig. 2 (a)). If the faulty node occurs in the internal nodes, each node of the path links its two sons using one edge whose

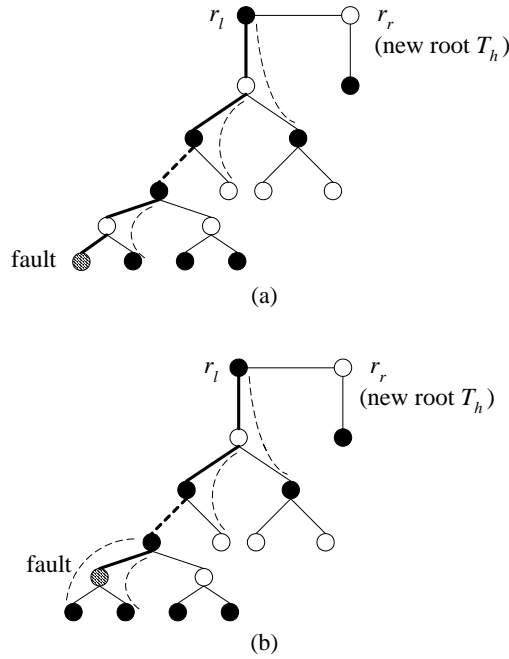


Fig. 2. The path from the faulty node to  $r_l$  is shown by the solid lines.

dilation value is 2 and the other edge whose dilation value is 1, while the parent node of the faulty node uses two edges whose dilation values are 2 to link its two sons (see Fig. 2 (b)). The congestion of the edges of the path are equal to 2 in  $H_{h+1}$  when the faulty node occurs in the leaf nodes or the internal nodes of  $T_h$ .

Therefore, a faulty node occurs in level  $i$  ( $i > 0$ ) of  $T_h$ , and the number of edges with dilation 2 is  $i - 1$  if the faulty node is a leaf node, or it is  $i + 1$  if the faulty node is an internal node. The congestion of the edges of the path increases by 1 in  $H_{h+1}$ .  $T_h$  can be reconfigured dynamically in  $H_{h+1}$  with *dilation* 2, and *congestion* 2, and there are  $i + 1$  affected nodes after reconfiguring when an arbitrary faulty node occurs in  $H_{h+1}$ .  $\square$

Now we will consider  $H_{h+1}$  with at least two faulty nodes. To reconfigure  $T_h$  in  $H_{h+1}$ , we need the following lemma.

**Lemma 1**  $DT_{h-1}$  ( $h \geq 1$ ) can be embedded into  $H_n$  as each leaf node of  $DT_{h-1}$  is linked to a certain internal node of  $DT_{h-1}$  via an edge in  $H_h$ .

**Proof:** We color the nodes of  $DT_{h-1}$  black or white in  $H_h$ . Suppose the leaf nodes of the left subtree of the roots are black and the leaf nodes of the right subtree of the roots are white. We can prove the lemma by induction on  $h$ .

**Hypothesis:**  $DT_{h-2}$  can be embedded into  $H_{h-1}$  as each leaf node of  $DT_{h-2}$  is linked to a certain internal node of  $DT_{h-2}$  via an edge in  $H_{h-1}$ .

**Basis Step:** When  $h = 1, 2$ , it is trivial.  $DT_2$  can be embedded into  $H_3$  as shown in Fig. 3. The figure shows the links between the leaf nodes and internal nodes in  $H_3$ ; for example, leaf nodes  $n_3, n_4, n_7$  and  $n_8$  link to internal nodes  $n_2, n_5, n_1$  and  $n_6$ , respectively.

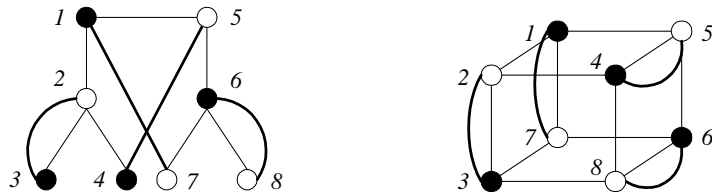


Fig. 3. Linking leaf nodes to certain internal nodes of  $DT_2$ . The solid lines represent the links in  $H_3$ .

**Induction Step:** We partition  $H_h$  into two  $H_{h-1}$ 's by means of the most significant bit, and each of the  $H_{h-1}$ 's contains  $DT_{h-2}$  as shown in Fig. 4 (a). By the symmetry properties of the hypercube, we can embed  $DT_{h-1}$  into  $H_h$  by adding the matching edges and by cross linking the trees as shown in Fig. 4 (b) [1]. The links between leaf nodes and certain internal nodes in  $DT_{h-1}$  are the same as described in the above hypothesis. Therefore, the lemma is proved.  $\square$

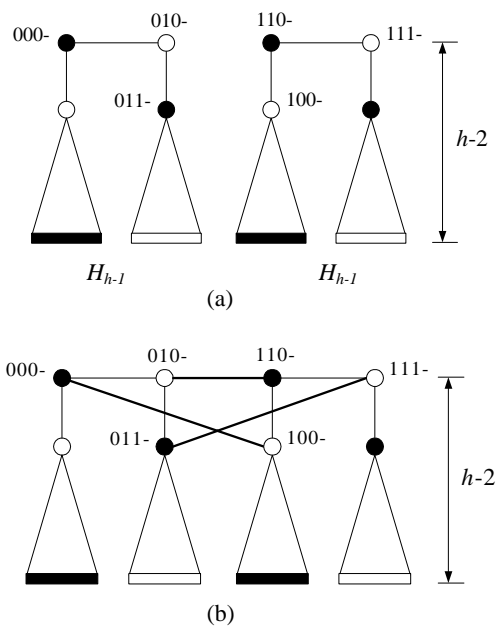


Fig. 4. Construction of  $DT_{h-1}$  from two  $DT_{h-2}$ 's. The added edges are represented by the solid lines. Nodes 010- and 110- are both roots of  $DT_{h-1}$ .

**Theorem 2**  $T_h$  can be reconfigured dynamically and embedded into  $H_{h+1}$  with *dilation* 3 and *congestion* 3, and a leaf node of  $T_h$  can be embedded into one faulty node of  $H_{h+1}$  when two arbitrary faulty nodes occur in  $H_{h+1}$ .

**Proof:** Likewise, we can construct  $T_h$  from  $DT_h$  in  $H_{h+1}$  (see Fig. 1). When two arbitrary faulty nodes  $u$  and  $v$  occur in  $H_{h+1}$ , there are two cases that need to be considered as follows.

**Case 1.** The two faulty nodes  $u$  and  $v$  are not adjacent in  $H_{h+1}$ , and if  $u$  is in level  $i$  and  $v$  is in level  $j$ , where  $i \leq j$  (see Fig. 5). Faulty node  $u$  can be reconfigured as described in Theorem 1, and the other faulty node  $v$  has to be reconfigured and embedded into a leaf node of  $T_h$  to reduce the influence of the structure of  $T_h$ . If  $v$  is a leaf node of  $T_h$ , we can complete the reconfiguration of  $T_h$ . If  $v$  is an internal node of  $T_h$ , then faulty node  $v$  has to be re-embedded into a leaf node of  $T_h$ . Since internal node  $v$  of  $DT_h$  has an edge to link it with a certain leaf node,  $w$ , according to Lemma 1, faulty node  $v$  can be re-embedded into leaf node  $w$ . Hence, the dilation of an edge which links the parent node of  $v$  and  $w$  is 2, and the dilation of edges which link  $w$  and two sons of  $v$  is 2. The congestion of edge  $(v, w)$  of  $H_{h+1}$  is 3.

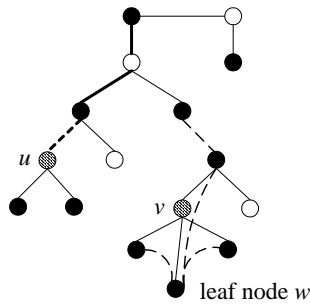


Fig. 5. Two faulty nodes  $u$  and  $v$  are not adjacent in  $H_{h+1}$ .

**Case 2.** The two faulty nodes  $u$  and  $v$  are adjacent in  $H_{h+1}$ , and  $u$  and  $v$  are in level 0, or  $u$  is in level  $i$  and  $v$  is in level  $j$ , where  $j - i = 1$  (see Fig. 6). When nodes  $u$  and  $v$  are in level 0 (see Fig. 6 (a)), we let leaf node  $w$ , which has an edge to link it with faulty node  $u$ , become the root of  $T_h$ . The dilation values of two edges linking  $w$  and its two sons are, respectively, 3 and 2. The congestion of edge  $(w, u)$  of  $H_{h+1}$  is 2.

When faulty node  $u$  is in level  $i$  and  $v$  is in level  $j$ , where  $j - i = 1$  (see Fig. 6 (b)), faulty node  $u$  can be reconfigured as described in Theorem 1. If faulty node  $v$  is a leaf node of  $T_h$ , we can complete the reconfiguration of  $T_h$ . If  $v$  is an internal node of  $T_h$ , faulty node  $v$  has to be re-embedded into a leaf node of  $T_h$ . According to Lemma 1, assume that leaf node  $w$ , which has an edge to link it with faulty node  $v$ , is used to recover

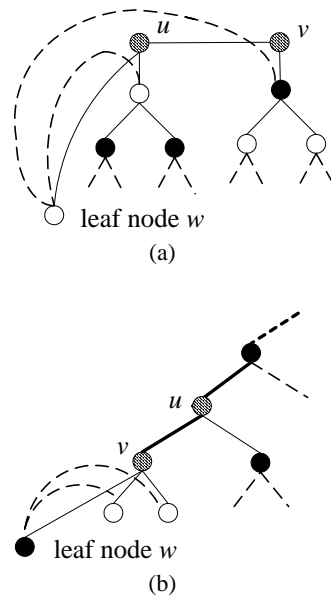


Fig. 6. Two faulty nodes  $u$  and  $v$  are adjacent in  $H_{h+1}$ .

faulty node  $v$ . The dilation value of an edge which links the parent node of  $u$  and  $w$  is 3 after re-embedding, and the dilation and congestion values of the remaining edges are the same as in case 1.

Therefore,  $T_h$  can be reconfigured dynamically and embedded into  $H_{h+1}$  with two arbitrary faulty nodes, and a leaf node of  $T_h$  can be embedded into one faulty node. Both the *dilation* and *congestion* values are 3, and there are at most  $h + 2$  affected nodes after reconfiguring.  $\square$

When more than two faulty nodes appear in  $H_{h+1}$ , we impose a constraint on the number and type of faulty nodes as follows.

**Constraint:** When an internal node of the double-rooted complete binary tree in a hypercube is faulty, the nodes which are adjacent to the faulty node have to be nonfaulty. At least two of the leaf nodes, which have edges to link the faulty node and its adjacent nonfaulty nodes, have to be nonfaulty.

Now we will consider how to reconfigure  $T_h$  with this constraint.

**Theorem 3** With the above constraint,  $T_h$  can be reconfigured dynamically and embedded into  $H_{h+1}$  with *dilation* 3 and *congestion* 3, and the leaf nodes of  $T_h$  can be embedded into the faulty nodes of  $H_{h+1}$ .

**Proof:** First, we construct  $T_h$  from  $DT_h$  in  $H_{h+1}$  (see Fig. 1). Each internal node has an edge to link it with a certain leaf node by Lemma 1; such linking edges are described by

the dashed lines shown in Fig. 7. Without loss of generality, we can consider the following probable cases. Assume that node  $n_2$  is faulty, and that node  $a$  is nonfaulty (see Fig. 7). Node  $n_2$  can be re-embedded into node  $a$ ; hence, the dilation values of edges  $(n_1, a)$  and  $(a, n_3)$  and  $(a, n_4)$  are all 2, and the congestion value of edge  $(n_2, a)$  of  $H_{h+1}$  is 3.

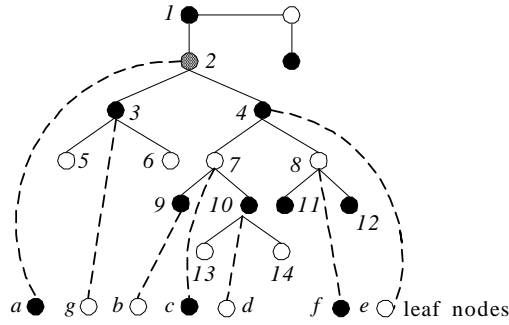


Fig. 7. The dashed lines describe the linking between the internal nodes and the leaf nodes.

Moreover, assume that node  $a$  is also faulty, while at least one of the two leaf nodes  $g$  and  $e$ , which link, respectively, to nodes  $n_3$  and  $n_4$ , is nonfaulty. Without loss of generality, let node  $e$  be a nonfaulty node; then, node  $n_2$  can be re-embedded into node  $n_4$ , and node  $n_4$  can be re-embedded into node  $e$ . Hence, the dilation values of edges  $(n_1, n_4)$ ,  $(n_4, n_3)$ ,  $(e, n_7)$  and  $(e, n_8)$  are all 2, the dilation value of edge  $(n_4, e)$  is 1, and the congestion value of edge  $(n_4, e)$  of  $H_{h+1}$  is 3.

Furthermore, assume that node  $n_7$  or  $n_8$  is faulty. Now we will consider node  $n_7$  only. At least one of the three leaf nodes  $b$ ,  $c$  and  $d$ , which link, respectively, to internal nodes  $n_9$ ,  $n_7$  and  $n_{10}$ , is nonfaulty. Let node  $d$  be nonfaulty; node  $n_7$  can be re-embedded into node  $n_{10}$ , and node  $n_{10}$  can be re-embedded into node  $d$ . Hence, the dilation values of edges  $(e, n_{10})$ ,  $(n_{10}, n_9)$ ,  $(n_{10}, d)$ ,  $(d, n_{13})$  and  $(d, n_{14})$  are, respectively, 3, 2, 1, 2 and 2, and the congestion value of edge  $(n_{10}, d)$  of  $H_{h+1}$  is 3. Similarly, if nodes  $n_8$  or  $n_{13}$  or  $n_{14}$ , etc., are faulty, then these faulty nodes can be reconfigured with a dilation value of at most 3 and a congestion value of at most 3.

Therefore,  $T_h$  with the constraint can be reconfigured dynamically and embedded into  $H_{h+1}$  with *dilation* 3 and *congestion* 3, and there are at most three affected nodes for reconfiguring each faulty node if there are more than two faulty nodes appearing in  $H_{h+1}$ . According to the definition of the constraint, the number of internal nodes which are faulty is at most

$$2^{h-2}(\text{in level } h-1) + 2^{h-3}(\text{in level } h-2) + \dots + 2^0(\text{in level } 1) + 2^0(\text{in level } 0) = 2^{h-1}.$$

At least  $2^{h-1}$  leaf nodes for reconfiguring are nonfaulty. Hence, at most  $2^{h-1}$  leaf nodes are faulty. □

### 4. RECONFIGURING $T_h$ IN $H_{h+2}$ WITH FAULTS

It has been shown that  $T_h$  can be embedded into  $H_{h+2}$  so that the adjacency of  $T_h$  is preserved [6, 7]. There remain  $2^{h+1} + 1$  free nodes for embedding  $T_h$  into  $H_{h+2}$ . In this section, we will present a reconfigurable algorithm for embedding  $T_h$  into  $H_{h+2}$  with faults. The algorithm allows at most  $2^{h+1} - 1$  faulty nodes in  $T_h$ , only the faulty nodes are affected by the reconfiguration, and the dilation and congestion values of reconfigurable embedding are, respectively, 2 and 1.

First, we will give a definition and a lemma before reconfiguring  $T_h$  in  $H_{h+2}$ .

**Definition 1** Let  $LT_h$  ( $h \geq 0$ ) denote a graph which has two complete binary trees of the same height  $h$ , and let there be an augmented edge to link two nodes at the same position between two complete binary trees (see Fig. 8 for  $LT_2$ ).

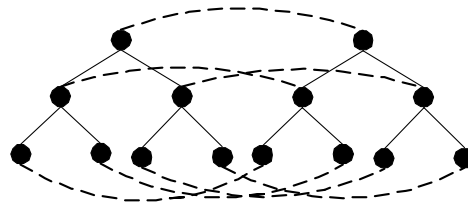


Fig. 8.  $LT_2$  (The augmented edges are described by the dashed lines).

**Lemma 2**  $LT_h$  ( $h \geq 0$ ) can be embedded into  $H_{h+2}$  with *dilation 2* and *congestion 1*.

*Proof:* We prove the theorem by induction on  $h$ .

**Hypothesis:** Embedding  $LT_{h-1}$  into  $H_{h+1}$  with *dilation 2* and *congestion 1* is true.

**Basis Step:** When  $h = 0, 1$  and  $2$ , it is trivial. The embedding of  $LT_2$  into  $H_4$  is shown in Fig. 9. It is a one-to-one node embedding, where the dilation value of edges  $(n1, n6)$  and  $(a, f)$  of  $LT_2$  is 2, and the *congestion* value is 1.

**Induction Step:** We use  $h + 2$  bits to label each node of  $H_{h+2}$ .  $H_{h+2}$  can be partitioned into four sub-hypercube  $H_h$ 's by means of the leftmost two bits of the hypercube, and the binary numbers of the leftmost two bits of the four sub-hypercubes correspond to 00, 01, 10 and 11 (see Fig. 10 (a)). Since any hypercube is symmetric, we embed  $DT_{h-1}$  into the sub-hypercube 00 according to nodes  $a, n1$  and  $n5$ , and embed  $DT_{h-1}$  into the sub-hypercube 01 according to nodes  $b, n3$  and  $n7$ . Then we can construct  $LT_{h-1}$  in the two sub-hypercubes. Likewise, we construct  $LT_{h-1}$  in the sub-hypercubes 10 and 11 according to nodes  $n6, n2, c, n8, n4$  and  $d$  as shown in Fig. 10 (a). We can construct  $LT_h$  in  $H_{h+2}$  by adding the matching edges and cross linking the  $LT_{h-1}$ 's as shown in Fig. 10 (b). There are eight  $T_{h-2}$ 's in two  $H_{h+1}$ 's, which are denoted  $t1, t2, t3, t4, t5, t6, t7$  and  $t8$  from left to right, respectively.

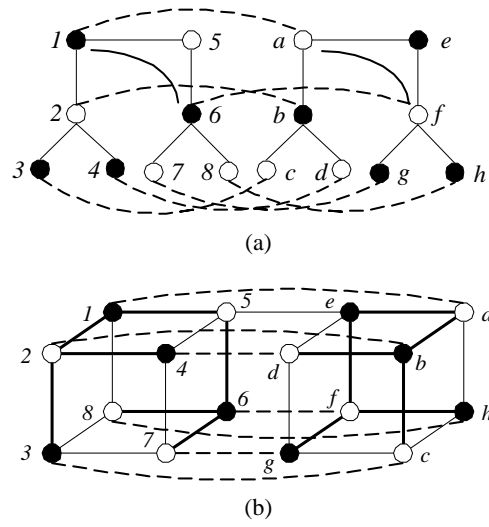


Fig. 9. (a)  $LT_2$  in two  $DT_2$ 's. (b) The embedding of  $LT_2$  into  $H_4$ . (The augmented edges are described by the dashed lines.)

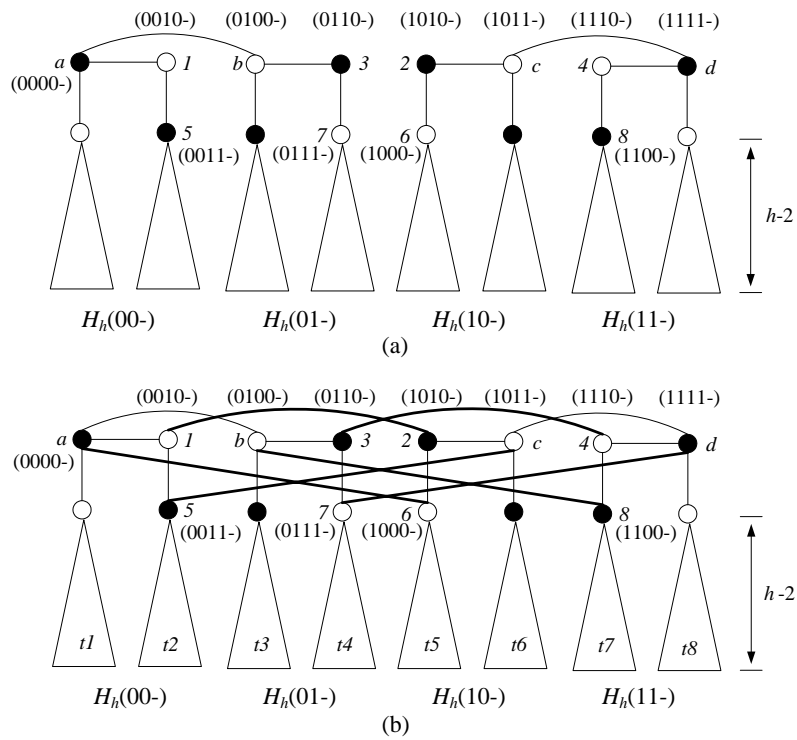


Fig. 10. Construction of  $LT_h$  from  $LT_{h-1}$ 's in four  $H_h$ 's. The added edges are shown by the solid lines. The binary numbers of the leftmost four bits of the nodes are written beside the nodes.

The links among nodes at the same positions are the same as described in the hypothesis except for the two new roots,  $n1$  and  $n3$ , in  $LT_h$ , while there is an unused edge for linking  $n1$  and  $n3$  (see Fig. 11). The *dilation* and *congestion* values are the same as in the hypothesis. Therefore, this theorem is true for any dimension of hypercube according to induction.  $\square$

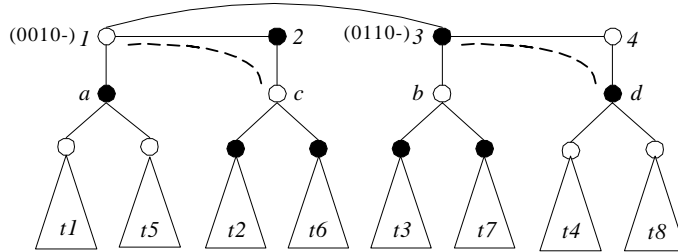


Fig. 11.  $LT_h$ .

**Theorem 4**  $T_h$  can be reconfigured dynamically and embedded into  $H_{h+2}$  with *dilation* 2 and *congestion* 1, and only the faulty nodes will suffer from the impact of reconfiguration when there are at most  $2^{h+1} - 1$  faulty nodes in  $T_h$ .

**Proof:** We can embed  $LT_h$  into  $H_{h+2}$  with *congestion* 1; the dilation value of the two edges will be 2, and those of the others will be 1 according to Lemma 2. Each node of one  $T_h$ , denoted by  $T_l$ , of  $LT_h$  has an edge to link it with a node (at the same position) of the other  $T_h$ , denoted by  $T_r$ , of  $LT_h$ .

Assume that  $T_h$  is initially embedded into  $T_l$  with *dilation* 2 and *congestion* 1. When node  $n1$  (or  $n3$ ) is faulty in  $T_l$  (see Fig. 12), we let node  $n1$  (or  $n3$ ) be re-embedded into node  $n8$ . Moreover, if nodes  $n1$  and  $n3$  are faulty in  $T_l$ , we let both faulty nodes be re-embedded into nodes  $a$  and  $c$  of  $T_r$ , respectively. Similarly, if other arbitrary faulty nodes appear in  $T_l$ , we re-embed the faulty nodes into the free nodes (at the same position) of  $T_r$ , and link them to two sons of the free nodes, then return to the two sons of the faulty node of  $T_l$ . The dilation value of the edges which link the free nodes of  $T_r$  to its parent and sons is at most 2, and the congestion value of the edges of  $H_{h+2}$  is 1.

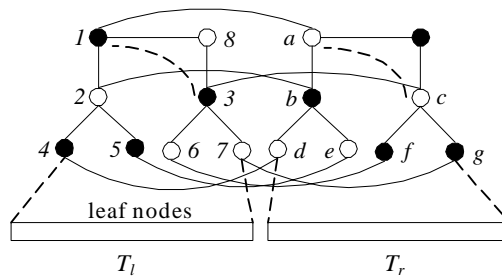


Fig. 12.  $T_h$  is reconfigured in  $LT_h$ .

Therefore,  $T_h$  can be reconfigured dynamically and embedded into  $H_{h+2}$  with *dilation* 2, and *congestion* 1. The number of affected nodes will be minimized after reconfiguration when there are at most  $2^{h+1} - 1$  faulty nodes in  $T_h$ .  $\square$

## 5. CONCLUSIONS

This paper has presented simple but effective algorithms for reconfiguring dynamically complete binary trees in hypercubes. If there is an arbitrary faulty node in  $H_{h+1}$ , then both the *dilation* and *congestion* values will be 2 after reconfiguring  $T_h$ . If there are two arbitrary faulty nodes in  $H_{h+1}$ , then both the *dilation* and *congestion* values will be 3 after reconfiguration. In this case, there are at most  $h + 2$  affected nodes after reconfiguring  $T_h$ . Moreover, if there are more than two faulty nodes in  $H_{h+1}$ , we have discussed how  $T_h$  can be reconfigured dynamically with the constraint in  $H_{h+1}$ . The *dilation* and *congestion* values will be 3 after recovery, and there will be at most three affected nodes after each faulty node is reconfigured.

In addition, we have present an algorithm that can be used to reconfigure  $T_h$  dynamically with at most  $2^{h+1} - 1$  faulty nodes in  $H_{h+2}$ , where the *dilation* and *congestion* values are, respectively, 2 and 1, and the number of affected nodes is minimized following reconfiguration.

## REFERENCES

1. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann, Reading, MA, 1992.
2. J. Bentley and H. T. Kung, "A tree machine for searching problems," in *Proceedings of the IEEE International Conference on Parallel Processing*, 1979, pp. 257-266.
3. Q. F. Stout, "Sorting, merging, selection, and filtering on tree and pyramid machine," in *Proceedings of the International Conference on Parallel Processing*, 1983, pp. 214-221.
4. E. Horowitz and A. Zorat, "Divide-and-conquer for parallel processing," *IEEE Transactions on Computers*, Vol. 32, 1983, pp. 582-585.
5. L. Nebesky, "On cubes and dichotomic tree," *Časopis pro Pěstování í Matematiky*, Vol. 99, 1974, pp. 164-167.
6. A. Y. Wu, "Embedding of tree networks into hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 2, 1985, pp. 238-249.
7. E. L. Leiss and H. N. Reddy, "Embedding complete binary trees into hypercubes," *Information Processing Letters*, Vol. 38, 1991, pp. 197-199.
8. A. S. Wagner, "Embedding the complete tree in the hypercube," *Journal of Parallel and Distributed Computing*, Vol. 20, 1994, pp. 241-247.
9. J. Wu, E. B. Fernandez, and Y. Luo, "Embedding of binomial trees in hypercubes with link faults," *Journal of Parallel and Distributed Computing*, Vol. 54, 1998, pp. 59-74.
10. M. Y. Chan and S. J. Lee, "Fault-tolerant embedding of complete binary trees in hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, 1993, pp. 277-288.

11. B. M. Y. Chan, F. Y. L. Chin, and C. K. Poon, "Optimal simulation of full binary trees on faulty hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 6, 1995, pp. 269-286.
12. P. J. Yang and C. S. Raghavendra, "Embedding and reconfiguration of binary trees in faulty hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 7, 1996, pp. 237-245.
13. A. Wang, R. Cypher, and E. Mayr, "Embedding complete binary trees in faulty hypercubes," in *Proceedings of the 3rd IEEE symposium on Parallel and Distributed Processing*, 1991, pp. 112-119.
14. J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a hypercube in the presence of faults," in *Proceedings of 19th Annual ACM Symposium Theory of Computing (STOC)*, 1987, pp. 274-284.
15. N. Krishnakumar, V. Hegde, and S. S. Iyengar, "Fault tolerant based embeddings of quadrees into hypercubes," in *Proceedings of the International Conference on Parallel Processing*, 1991, pp. 244-249.

**Chui-Cheng Chen (陳垂呈)** received his B.S. degree in Information Engineering from Tatung Institute of Technology, Taiwan, R.O.C., in 1989, the M.S. degree in Computer and Information Science from National Chiao Tung University, Taiwan, in 1991, and the Ph.D. degree in Computer Science and Information Engineering from National Chiao Tung University, in January 1996. He is now an Associate Professor in the Department of Information Management of Southern Taiwan University of Technology in Tainan, Taiwan. His current research interests include parallel computation, e-commerce, and data mining.