

Constructing Fault-Tolerant Communication Trees in Hypercubes*

PO-JEN CHUANG, SHIH-YUAN CHEN AND JUEI-TANG CHEN

Department of Electrical Engineering

Tamkang University

Tamshui, 251 Taiwan

A communication tree is a binomial tree embedded in a hypercube, whose communication direction is from its leaves to its root. If a problem to be solved is first divided into independent subproblems, then each subproblem can be solved by one of the hypercube processors, and all the subresults can be merged into the final results through tree communication. This paper uses two random search techniques, the genetic algorithm (GA) and simulated annealing (SA), to construct fault-tolerant communication trees with the minimum data transmission time. Experimental evaluation shows that, with reasonably low search time, the proposed GA and SA approaches are able to find more desirable communication trees (i.e., trees with less data transmission time) than the minimal cost approach can. A distributed approach which applies parallel search to communication subtrees in disjoint subcubes is also provided to reduce the search time of the proposed approaches.

Keywords: fault-tolerant communication trees, hypercubes, genetic algorithms, simulated annealing, data transmission time, search time, maximal fault-free subcubes

1. INTRODUCTION

A communication tree is a binomial tree embedded in a hypercube, whose communication direction is from its leaves to its root. If a problem to be solved in a hypercube is first divided into independent subproblems, then each subproblem can be solved by one of the hypercube processors, and all the subresults can be merged into the final results through tree communication (i.e., from the leaves to the root.) To obtain a communication tree, a sink (the receiver node of the final results) and the order of dimensions (for merging the subresults) should be determined in the beginning because different sinks and dimension orders lead to different communication trees. Thus, in an n -dimensional hypercube, there will be a total of $n! \times 2^n$ communication trees, each with n communication stages and $2^n - 1$ links.

The number of communication trees in a hypercube can be large. It is, therefore, desirable to find and use a fault-free tree, that is, a tree with no link failures. (A faulty node is a node all of whose links to neighbors are faulty.) Finding a fault-free tree in a faulty hypercube is, nevertheless, quite complex as all $n! \times 2^n$ possibilities (trees) and $2^n - 1$ used links in each tree need to be checked. If a fault-free communication tree does not

Received January 31, 2003; accepted July 4, 2003.

Communicated by Shih-Pyng Shieh.

* A preliminary version of this paper has been presented at the 2002 International Computer Symposium, 2002.

exist, it becomes essential to locate an optimal tree – the tree with the fewest faulty links – and to reroute messages dynamically according to the fault pattern so that the data transmission time (the time needed to collect data into the sink) can be kept as short as possible. By checking all possible sinks and dimension orders one by one, *exhaustive search* is able to find an optimal communication tree. This operation is, however, inefficient and complicated: It has to check all $n! \times 2^n$ possibilities and $2^n - 1$ used links in a tree. The time complexity will be as much as $O(n! \times 2^n) \times O(2^n - 1) = O(n! \times 4^n)$, and when the dimension is high, the number of possibilities may be too large to handle.

When a located optimal communication tree is not fault-free, it is necessary to reroute messages in such a way that the data transmission time can be reduced as much as possible. The *minimal cost* approach [1] is designed to achieve this goal, i.e., to find fault-tolerant communication trees in hypercubes. Its algorithm runs as follows: A node is first selected as the sink, the dimension order is determined according to the fault pattern, and then the selected sink node (the root) to the leaves is checked sequentially, passing as few link failures as possible to reduce the cost. The approach does not allow the adjacent links of the sink to be faulty; therefore, the number of faulty links in a hypercube must be kept under 2^{n-1} [1]. In addition, a communication tree so constructed may not be optimal in terms of the data transmission time.

Two random search techniques, the genetic algorithm (GA) [2, 3] and simulated annealing (SA) [4], are utilized in this paper to find a fault-tolerant communication tree in a hypercube with the shortest possible data transmission time. When we employ the GA to generate an optimal fault-tolerant communication tree in the hypercube, we first choose the node with the minimum number of faulty adjacent links as the sink and encode all the dimensions into a string. Then, some strings are randomly generated as the initial population, and the three genetic operations – selection, crossover, and mutation – are executed to locate an excellent communication tree. When we apply the SA approach, we let the solution be encoded in the same way as a chromosome in the GA approach. Each dimension number in the string is called an *element* of the encoded solution. The objective function is defined to calculate the data transmission time needed to collect data into the sink through the communication tree; its value needs to be minimized.

Simulation results show that, with reasonably low time complexity, both the GA and SA approaches are able to get better communication trees in terms of the data transmission time. A distributed approach is also proposed in the paper to further reduce the complexity. The approach first partitions an n -dimensional hypercube into several disjoint k -dimensional subcubes. The search for communication subtrees with the lowest data transmission time is then simultaneously activated in these k -dimensional subcubes by using the two random search techniques. Two algorithms, one that uses bit comparison (a *top-down* approach) and another that uses dimension screening (a *bottom-up* approach), are presented to find the maximal fault-free subcube in a faulty hypercube. Once the objective communication subtrees are found, they can be easily combined to form a final communication tree. Because parallel search (for communication subtrees) is applied in disjoint subcubes, the time complexity of attaining the final communication tree can be effectively reduced.

This paper is organized as follows. Section 2 provides background information. Section 3 presents the proposed GA and SA approaches for locating the optimal communication tree. Experimental performance evaluation of various approaches is given in

section 4. Aided by two algorithms, a distributed approach devised to further trim the search time complexity of the proposed approaches is introduced in section 5. Section 6 concludes the paper.

2. BACKGROUND

An n -dimensional hypercube, called an n -cube here, comprises 2^n nodes, each with n adjacent links serving as direct communication channels to n neighbors. Nodes in an n -cube are numbered from 0 to $2^n - 1$ and denoted using n -bit binary numbers ($x_{n-1} \dots x_i \dots x_0$) as their addresses. For convenience, bit i (x_i) is referred to as dimension i of the binary representation of an address. A node and any of its neighbors are exactly one bit different in their addresses. Two nodes with address bits that are different in only one dimension, say j , are called dimension- j neighbors. For example, in the 4-cube illustrated in Fig. 1, nodes 0100 and 0110 are dimension-1 neighbors, and the link between them is denoted as 01-0 (with a dashed bit in bit 1). A k -dimensional subcube in an n -cube, called a k -subcube here, comprises 2^k nodes such that all the nodes are exactly $n - k$ bit identical in addresses, and the remaining k bits are *don't care*. For example, nodes 1001, 1011, 1101, and 1111 shown in Fig. 1 form a 2-subcube denoted as 1**1, with two identical bits (1's) and two don't care bits (*'s).

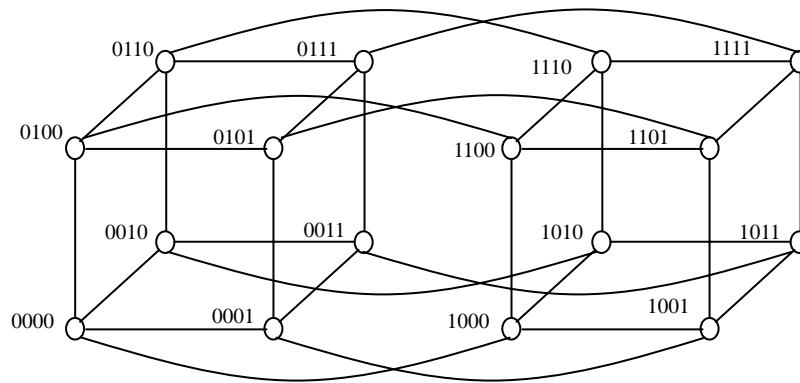


Fig. 1. A 4-cube.

A tree communication scheme was introduced in [5] to solve a problem that can be divided into independent subproblems. The scheme first engages a hypercube processor to solve a subproblem and then merges all the subresults into the final results through tree communication. A communication tree constructed by the scheme for a 4-cube with 4 communication stages is shown in Fig. 2. The lines with arrows indicate the directions of data flow. The node at the starting point is a data sending node, and the one at the arrow-head is a receiving node. For instance, nodes 0001 and 0000 form a sender-receiver pair at stage 0. After receiving data, the receiving node performs computation on the received data and its own data. Through $\log_2 N$ ($N = 16$ in this case) stages of operation, the final

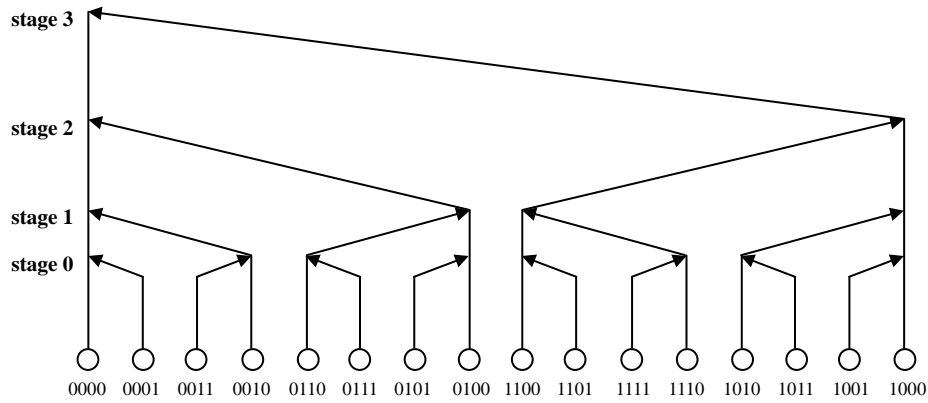


Fig. 2. A tree communication scheme for a 4-cube.

results are gathered into node 0000. An n -level binomial tree BT_n can be constructed recursively by connecting the roots of two BT_{n-1} 's and assigning one of them as the root of BT_n . The fault-free nodes and links involved in a broadcasting session also form a binomial tree with reversed communication directions, i.e., from the root down to the leaves. Such a tree communication scheme can be applied to various types of computations.

There are different ways to generate new communication trees. With the receiver of the last stage defined as the sink, we can generate a communication tree by selecting a node as the sink or by deciding on a dimension for messages at a stage to be traversed. That is, we can generate a new tree either by choosing another node as the sink or by changing the order of dimensions; thus, totally there will be $n! \times 2^n$ feasible communication trees for an n -cube. A simple communication tree with node 0^n as the sink and $(0, 1, 2, \dots, n-1)$ as the dimension order can function efficiently and correctly in a fault-free system, but when hardware failures (such as faulty links) occur, the tree communication scheme becomes defective. It is indeed quite complex to find a fault-free communication tree in a faulty hypercube since all $n! \times 2^n$ possibilities (trees) and also the $2^n - 1$ links used in each tree must be checked. The time complexity will be as much as $O(n! \times 2^n) \times O(2^n - 1) = O(n! \times 4^n)$. Moreover, if a fault-free communication tree does not exist, it becomes necessary to find a communication tree with the fewest possible faulty links and to reroute messages dynamically according to the fault pattern in order to cut the data transmission time.

3. THE PROPOSED APPROACHES

As mentioned previously, the major challenge in constructing communication trees is to achieve the shortest data transmission time. Due to the fact that communication trees found using the exhaustive search (EX) approach and the minimal cost (MC) approach are not necessarily the best ones in terms of the data transmission time, this paper aims to find more desirable communication trees – trees with the shortest data transmission time possible – by using two random search techniques, the genetic algorithm and simulated annealing.

3.1 The Genetic Algorithm (GA) Approach

The genetic algorithm (GA) is an adaptive search technique that is able to explore a large search space. It provides an alternative to traditional optimization techniques by using directed random searches to locate optimal or near optimal solutions of complex problems and is based on the mechanisms of evolution and natural genetics. The GA operates on a pool of chromosomes which represent candidate solutions to the problem under investigation. Chromosomes are selected based on “survival of the fittest” and are passed down to the next generation in a process called “reproduction.” Reproduction is realized using such genetic operations as selection, crossover, and mutation to generate new points in the search space. An objective function is supplied and used to weigh the fitness values of the chromosomes.

To apply the GA to generate an optimal fault-tolerant communication tree in a hypercube, we first choose the node with the fewest faulty adjacent links as the sink and encode all the dimensions into a string. (One possible string in an 8-cube, for example, is “20146357.”) Then, some strings (assume that the number of strings – the so-called *population size* – is η) are randomly generated as the initial population, and the three genetic operations are employed to locate an excellent communication tree. That is, from the initial randomly generated strings (dimension orders of trees) to the formation of the final desired (optimal or near optimal) communication tree, the entire search process is conducted using the genetic algorithm described below.

Selection: Three strings from the pool of strings (population) are arbitrarily selected, and the best string (the one with the minimum data transmission time) is selected for the next generation. A new generation can be produced by repeating this process η times.

Crossover: Two strings are arbitrarily chosen from the population generated in the above selection step. The exchange is carried out as follows. First, the crossover points are found. (For instance, if the two chosen strings are 012435768 and 014352678, then the crossover points will be positions 2 and 5 with the leftmost position counted as position 0.) Then, two new strings are produced by exchanging the dimensions of the two original strings between the two crossover points. (Continuing with the above example, we get two new strings, 014352768 and 012435678). From the above 4 strings, the best one is selected for the next generation. The process is repeated η times to produce a new generation.

Mutation: A string, say Str_i , is selected (in order from string 1 to string η) from the population generated in the crossover step and a decision is made whether to change Str_i or not with a probability of Pr_m . If Str_i is to be changed, then two positions in Str_i are randomly chosen, and the two dimensions at the two positions are exchanged to get a new string; otherwise, the original string is maintained in the population.

Note that in the selection and crossover processes, the data transmission time is used as the fitness value to select the strings (chromosomes) for the next generation. Recall that the data transmission time is defined as the time needed to collect data into the sink, based on the dimension order of the communication tree. Assume that the generated data

to be collected from each node is 1 unit, and that the time needed to transmit 1 unit of data through a fault-free link is counted as 1 unit of data transmission time. When the data to be transmitted from a node (say, node x) through a certain dimension (say, dimension j) at a given stage encounter a faulty link, they will bypass the faulty link and be transmitted by way of all the fault-free neighbors of node x . Assuming that the number of available neighbors is 3, node x will divide data messages into three equal partitions and send one partition to each neighbor to balance the load, as discussed in [1]. The incurred extra transmission time is then added to the data transmission time.

Take the 3-cube shown in Fig. 3 (a) as an example. Assume that there are 4 faulty links, -00, 01-, 10-, and 1-1, in the cube, and that node 001 is chosen as the sink. As shown in the figure, the generated data to be collected from each node before transmission is 1 unit in size. To calculate the data transmission time for string 012 (i.e., the dimension order of a tree for a 3-stage data collection), we can follow the steps described below.

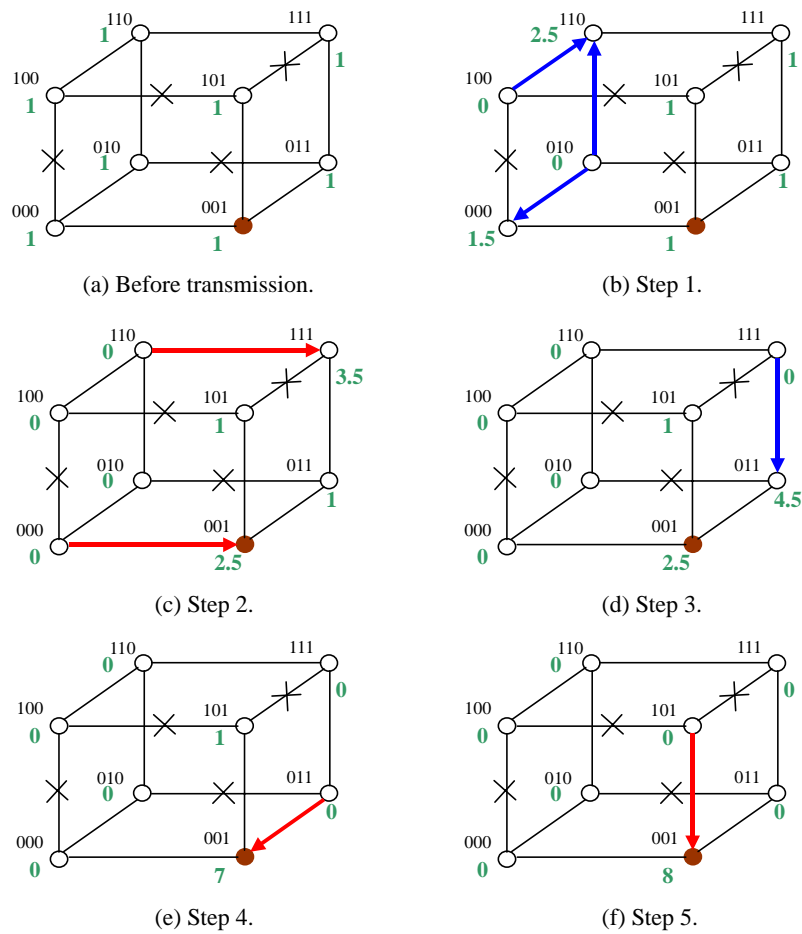


Fig. 3. Steps to illustrate the calculation of the data transmission time for a communication tree with sink 001 and dimension order 012.

- Step 1:** According to dimension order 012, the data are transmitted through dimension 0 at stage 0, i.e., from 000 to 001, 010 to 011, 110 to 111, and 100 to 101. Since dimension-0 links 01- and 10- are faulty, the data will bypass them and be transmitted by way of fault-free neighbors 000 and 110. The 1 unit of data generated in node 010 will be divided into 2 equal partitions, which will then be, respectively, transmitted to 000 and 110, and the 1 unit of data generated in node 100 will be transmitted to 110. Note that transmitting these data to the 2 neighbors will take only 1 unit of time due to parallel transmission. As a result, node 000 will have 1.5 units of data (1 unit generated plus 0.5 unit received), while node 110 will have 2.5 units (1 unit generated plus 1.5 units received) as shown in Fig. 3 (b).
- Step 2:** Since transmitting data through dimension 0 from node 000 to 001 and from 110 to 111, respectively, takes 1.5 and 2.5 units of time, the data transmission time at this step will be 2.5 units in all. The receiving nodes, 001 and 111, will now each possess 2.5 units and 3.5 units of data (Fig. 3 (c)).
- Step 3:** To transmit data through dimension 1 at stage 1 along the tree, it is necessary to bypass faulty link 1-1 and to move forward by way of neighboring node 011. The data transmission time needed for data to go from node 111 to 011 will, thus, be 3.5 units, and node 011 will contain 4.5 units of data after this step (Fig. 3 (d)).
- Step 4:** 4.5 units of time is needed at this step to transmit data through dimension 1 from node 011 to 001. As a result, node 001 will possess 7 units of data (Fig. 3 (e)).
- Step 5:** From Fig. 3 (f), we can see that transmitting data through dimension 2 at stage 2 from node 101 to 001 takes 1 unit of time. Now node 001 finally collects all 8 units of data, and the data transmission time for string 012 is, thus, $1 + 2.5 + 3.5 + 4.5 + 1 = 12.5$ units in total.

Based on the above steps, the possible minimum data transmission time (DT) for an n -cube can be derived as $2^n - 1 = (2^0 + 2^1 + 2^2 + \dots + 2^{n-1})$ units. Two conditions for terminating the algorithm are set as follows:

1. When the best possible solution (the solution with $DT = 2^n - 1$) is found, terminate the algorithm.
2. When DT_i is larger than $0.999 \times DT_{i-1}$, terminate the algorithm. (DT_i is the data transmission time of the best solution obtained from every 5 generations, where i denotes the sequence of the 5-generation groups. For example, DT_1 results from the first 5 generations, DT_2 from the second 5 generations, and so on.)

3.2 The Simulated Annealing (SA) Approach

The simulated annealing approach is another random search technique considered in this paper for forming a communication tree. The SA approach is basically an iterative random search procedure with adaptive moves used to locate the optimal or near optimal solutions of complex problems. As the name indicates, it needs an *annealing* schedule of the temperatures, besides a random generator of “moves” and an objective function. By permitting “uphill moves” under the control of a probabilistic criterion (a Boltzmann machine-like mechanism), the temperature is able to keep the algorithm from getting

stuck. Since the higher the temperatures are, the greater the probability of performing “uphill moves” is, this approach tends to avoid the first local minima encountered and has been successfully applied in different combinatorial optimizations, such as the Travel Salesman Problem [4].

The SA approach randomly generates one initial solution, which in turn generates a new solution based on the neighborhood structure. The two solutions will then compete by using the Boltzmann machine-like mechanism. During the process of minimization, if the objective function value of the new solution is lower than that of the initial one, then the new solution is selected. If the new solution has a higher value, it can still be selected under some probability that is usually assumed to result from the Boltzmann probability distribution function of the objective function value difference between the two “competing” solutions. The selected solution will generate another new solution, and the competing process will be repeated again. The iterative process will continue until convergence is achieved or for a specified length of times.

To employ the SA approach to generate an optimal communication tree in a hypercube, we let the solution be encoded in the same way as the chromosome in the GA approach (that is, the dimension-encoded string). Each dimension number in the string is called an *element* of the encoded solution. The objective function is defined in order to calculate the data transmission time needed to collect data into the sink through the communication tree (as described in section 3.1); therefore, its value needs to be minimized. The objective function considered is denoted by $\rho(x)$ for solution x .

The iterative process of the SA approach for searching a desired communication tree:

- (1) **Initialization:** Initialize the iteration count and the temperature. Generate one initial solution randomly. Set the initial solution as the selected solution (x).
- (2) **Iterative steps:**
 - (a) **Generation:** Generate a new solution (x') from the selected solution (x) based on the neighborhood structure; that is, randomly choose two elements from the selected encoded solution and exchange them to generate a new solution. (For instance, if the initial solution in an 8-cube is 20146357, a newly generated solution may be 20546317, obtained by exchanging 1 and 5 in the string.) Calculate the difference between the objective function values (the data transmission times) of the two solutions: say $\Delta\rho = \rho(x') - \rho(x)$. If $\Delta\rho \leq 0$, i.e., the objective function value of the new solution is not higher than that of the selected one, then the new solution becomes the selected solution. Otherwise, the new solution is selected with the probability $\exp(-\Delta\rho / T) > r$, where T is the temperature at the iteration, and r is a random number uniformly distributed between 0 and 1. Note that the above probability results from the Boltzmann probability distribution function and is used to permit “uphill moves.”
 - (b) **Cooling** (lower the temperature to reduce the probability of “uphill moves”).
 - (c) **Convergence check** (terminal checking according to the same conditions used in the GA approach).

4. EXPERIMENTAL EVALUATION

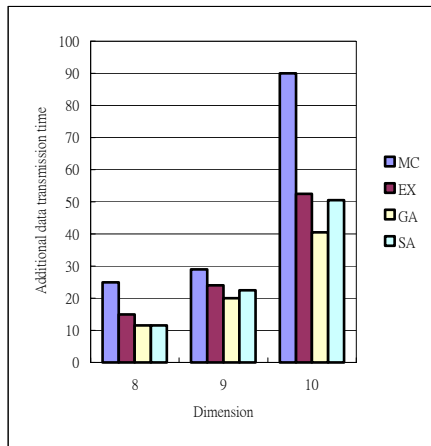
Extensive simulations were carried out to evaluate the performance of the exhaustive (EX) approach, the minimal cost (MC) approach, and our proposed GA and SA approaches in terms of the search time and solution quality. The search time is the time taken to find an objective communication tree; the solution quality is decided based on the data transmission time of the found communication tree: less data transmission time indicates better solution quality.

Simulation Model

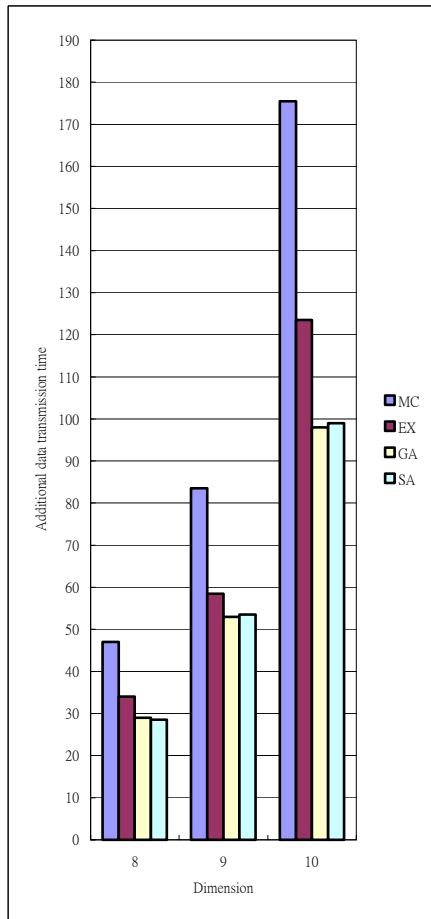
Simulations were conducted in hypercubes with different dimensions (8, 9 and 10) and different percentages of faulty links (10%, 20% and 30%). Link failures were initialized and distributed evenly in hypercubes; no new failures occurred during run-time. A total of 10 randomly generated hypercubes (each with a fixed number of faulty links) were used to evaluate the performance of the above approaches in finding objective communication trees. Each approach was run ten times in each of the 10 faulty hypercubes, and the results were used to calculate an average value. The 10 averaged values obtained from the 10 hypercubes were then averaged to obtain a final result. (The population size and the probability of mutation for the GA approach were randomly chosen to be 20 and 0.2. The choices of these values did not affect the resultant communication tree.)

Additional Data Transmission Time

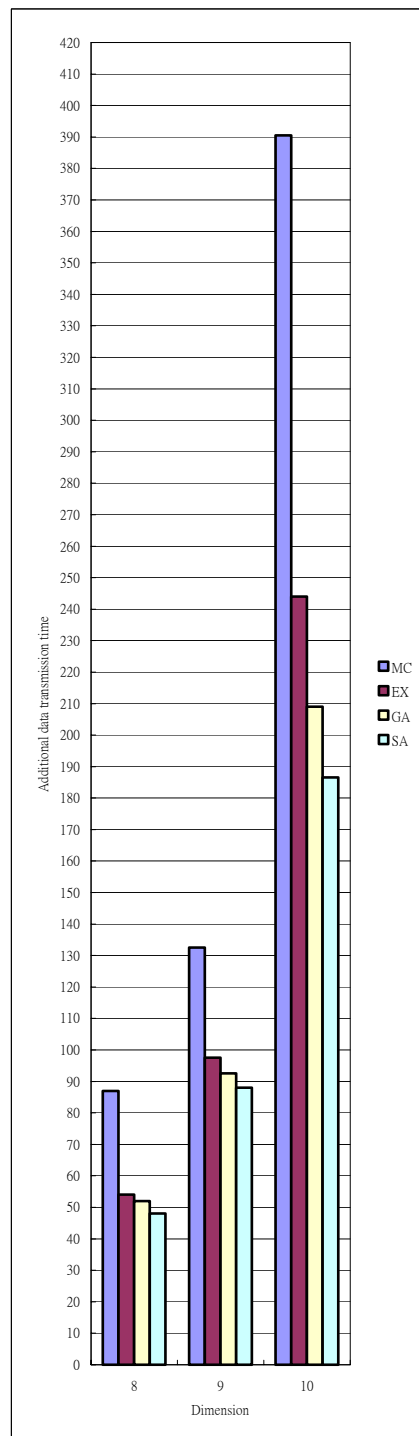
As the amount of data generated and collected from each node is 1 unit and the time for 1 unit of data in a node to be transmitted through a fault-free link to another node is counted as 1 unit of data transmission time, the possible minimum data transmission time for an n -cube is $\xi = 2^n - 1$. In the presence of faults, however, the incurred extra amount of transmission time due to rerouting of messages should also be added in. Therefore, the additional data transmission times (data transmission time – ξ) for the four approaches were collected so that we could conduct more explicit performance evaluation. The additional data transmission times were collected from hypercubes of different dimensions and different numbers of faulty links. As Fig. 4 shows, the proposed GA and SA approaches consistently took less additional data transmission time than the MC approach, and the difference grew as the number of faulty links and dimensions increased. For instance, the additional data transmission times for the MC, GA, and SA approaches were, indeed, quite similar in hypercubes of 8 or 9 dimensions with 10% faulty links (see Fig. 4 (a), where the GA and SA approaches display slightly lower values), but when the percentage of faulty links increased to 30%, the additional data transmission times for the GA and SA approaches apparently dropped (Fig. 4 (c)). The same trend was also observed for hypercubes of larger dimensions. That is, compared with the MC approach, the proposed GA and SA approaches were able to find better communication trees using less additional data transmission time, especially when the number of faulty links and the size of the hypercubes grew, which is the more practical situation. (The experimental performance of the EX approach is listed here for reference only because the search time involved in locating an optimal communication tree is conspicuous. In our simulation,



(a) 10% faulty links.



(b) 20% faulty links.



(c) 30% faulty links.

Fig. 4. Additional data transmission time for various approaches applied to faulty hypercubes.

the search conducted using the EX approach was terminated when a communication tree better than that obtained using the MC approach was obtained. The resulting data transmission time is shown in Fig. 4.)

Search Time

The GA and SA approaches took almost the same amount or slightly more search time to find a better communication tree than the MC approach. For example, the GA/SA approaches, respectively, took 0.11/0.09 and 0.97/0.48 second more search time to locate a better communication tree in 8- and 10-dimensional hypercubes with 10% faulty links. In fact, the MC approach was found to take the least search time to find an objective tree, but the tree was not as optimal as those found by the GA and SA approaches. In contrast, the EX approach took remarkably more search time (than all the other approaches) to locate a communication tree that was only better than that which the MC approach found, and its search time grew even more as the number of dimensions and faulty links increased. The GA and SA approaches also took more search time when the number of dimensions of the hypercubes and the number of faulty links grew, but the increase was small and insignificant. For example, in a 10-dimensional hypercube with 20% faulty links, the EX approach took 153 seconds more search time than the MC approach to find a better communication tree, while the GA and SA approaches took only 4.67 and 1.77 more seconds to locate even better communication trees. Then, when the hypercube had 30% faulty links, the EX, GA, and SA approaches, respectively, took 242.6, 1.92, and 6.57 seconds more search time than the MC approach. Note that the GA and SA approaches may take slightly more search time than the MC approach to locate objective communication trees, but the amount of extra time is negligible in comparison with the better quality of the located trees, i.e., trees with the shorter data transmission time.

5. A DISTRIBUTED APPROACH

A distributed approach is introduced in this section to further reduce the search time for the proposed GA and SA approaches. The idea behind the distributed approach is simple but effective: The n -cube is first partitioned into several disjoint k -subcubes, and the search for objective communication subtrees with the lowest data transmission time is simultaneously activated in these k -subcubes using the GA or SA approach. All the objective communication subtrees found in this way are then combined into a final communication tree in the n -cube. As the search for the subtrees in the disjoint subcubes is parallel in nature, the total search time for attaining a final communication tree can be effectively reduced.

To partition the n -cube into disjoint k -subcubes, a fault-free $(n - k)$ -subcube of the highest possible dimension should be found first. Each node of the $(n - k)$ -subcube is viewed as a subsink for each of the 2^{n-k} disjoint k -subcubes, i.e., as the root of the communication tree in each k -subcube. The search for communication subtrees with the lowest data transmission time is then simultaneously activated in these disjoint k -subcubes. To attain final results, the subresults in each k -subcube are collected into the subsink through each subtree, and the results in all the subsinks are collected into the sink through the tree in the fault-free $(n - k)$ -subcube. Any tree (with any sink and dimension

order) can be selected to collect the subresults (in the subsinks) into the final results (in the sink) because the $(n - k)$ -subcube is fault-free.

Finding the maximal fault-free subcube, i.e., the subcube with the highest dimension, in a faulty hypercube has been investigated in the literature [6-8] (in [6, 7], only node failures were considered). Algorithm I, derived from the *bit comparison* approach in [8], is presented in the following to serve the above purpose.

Algorithm I

```

if (there exist no faulty links) return ( $S_n$ ); /*  $S_n$  is the entire hypercube */
Choose a node  $m$  with minimal number of faulty adjacent links;
/* Search for an  $(n - i)$ -subcube which contains  $m$  and is fault-free */
 $i = 0$ ;
do
{
   $i = i + 1$ ;
  for (any of  $C(n, i)$  distinct  $(n - i)$ -subcube  $S_{n-i}$  which contains node  $m$ )
    if (non don't care notation bits of  $S_{n-i}$  are not all the same as the corresponding
        notation bits of any faulty link)
      return ( $S_{n-i}$ );
}
until ( $i = n - 1$ );
return ( $S_0$ ); /*  $S_0$  contains only  $m$  */

```

Inside the *do loop*, each $(n - i)$ -subcube containing node m is checked to see if it is fault-free through bit comparison with the faulty links. It can be easily derived that a subcube will be fault-free if its non don't care notation bits of the subcube are different from the corresponding notation bits of any faulty link. That is, if a subcube contains a faulty link, its non don't care notation bits of the subcube should be exactly the same as the corresponding notation bits of the faulty link. It is also clear that there are $C(n, i)$ distinct $(n - i)$ -subcubes which contain node m since the notation of a subcube containing a particular node can be obtained by changing any $n - i$ address bits of the node into don't care bits. To give an example, Algorithm I is presented here to find the maximal fault-free subcube in a 4-cube with 8 faulty links: 010-, 01-1, 0-11, 11-0, 11-1, 111-, 1-01, and -110 (as depicted in Fig. 5).

Step 1: Node 0001 is randomly chosen from nodes with zero (minimal) faulty adjacent links.

Step 2: Subcube 0*** is a 3-cube containing node 0001. Compared with the notation of the faulty links, it is found that the only non don't care notation bit of 0*** (0 in bit 3) is exactly the same as the corresponding bit (bit 3) of 010-, indicating that subcube 0*** at least contains faulty link 010-. Randomly select another subcube *0**.

Step 3: Through bit comparison, the only non don't care notation bit of *0** (0 in bit 2) is found to be different from the corresponding bit (bit 2) of any link (with 1 or - only in bit 2), meaning that subcube *0** contains no faulty links. Subcube *0** is, thus, returned as the required maximal fault-free subcube.

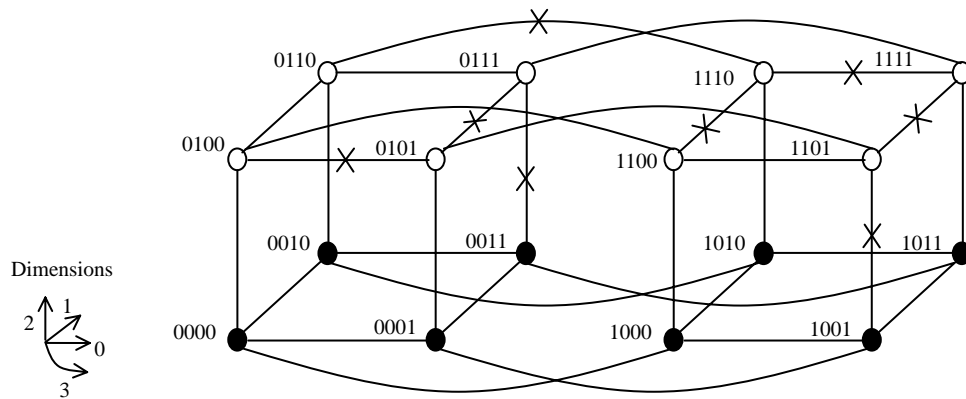


Fig. 5. A 4-cube with 8 faulty links.

As observed previously, Algorithm I checks all possible fault-free subcubes one by one through bit comparison, from the large sized subcubes $((n - 1)$ -subcubes) to the small sized subcubes (0-subcubes), to see if they contain faulty links. This can be considered a *top-down* approach, which works more desirably for hypercubes with larger fault-free subcubes, i.e., hypercubes with fewer faulty links. To be complete, Algorithm II, a bottom-up approach which locates the required fault-free subcube through *dimension screening*, is derived and provided next.

Algorithm II

```

if (there exist no faulty links) return ( $S_n$ ); /*  $S_n$  is the entire hypercube */
Choose a node  $m$  with the minimal number of faulty adjacent links as a fault free
0-subcube  $S_0$ ;
 $i = 0$ ;
 $dim = \varnothing$ ; /* empty set */
do
{
   $i = i + 1$ ;
   $candidate\_dim = \varnothing$ ;
  for (dimension  $j$  not in  $dim$ )
    if ( $S_{i-1}$  and all dimension- $j$  neighbors of its nodes form a fault-free  $i$ -subcube  $S_i$ )
    {
       $t =$  total number of faulty adjacent links of all the dimension- $j$  neighbors;
       $candidate\_dim = candidate\_dim \cup \{(j, t)\}$ ;
    }
  if ( $candidate\_dim \neq \varnothing$ )
  {
    select an element  $(j, t)$  from  $candidate\_dim$  where  $t$  is the minimum value in the set;
     $dim = dim \cup \{j\}$ ;
     $S_i =$  the  $i$ -subcube with all notation bits  $i$  (in  $dim$ ) that are don't care and the other
    bits that are the same as the corresponding address bits of node  $m$ ;
  }
}

```

```

    }
  else
    return ( $S_{i-1}$ );
  }
until ( $i = n$ );

```

The algorithm grows in the direction of the new dimension neighbors with the minimum total number of faulty adjacent links. The following steps can be taken to find the required fault-free subcube in the 4-cube depicted in Fig. 5.

- Step 1:** Node 0001 is randomly chosen as a fault-free 0-subcube (S_0) from nodes with zero (minimal) faulty adjacent links.
- Step 2:** S_0 forms a fault-free 1-subcube (S_1), respectively, with neighbors in dimensions 0, 1, 2, and 3, i.e., 0000, 0011, 0101, and 1001. t is calculated for each dimension, resulting in $candidate_dim = \{(0, 0), (1, 1), (2, 2), (3, 1)\}$. (0, 0) is selected, $dim = \{0\}$, and $S_1 = 000^*$.
- Step 3:** S_1 forms a fault-free 2-subcube (S_2), respectively, with dimension-1 neighbors 0010, 0011 and dimension-3 neighbors 1000, 1001. t is calculated for each dimension, resulting in $candidate_dim = \{(1, 1), (3, 1)\}$. (3, 1) is randomly selected since for both dimensions, $t = 1$. $dim = \{0, 3\}$ and $S_2 = *00^*$.
- Step 4:** S_2 forms a fault-free 3-subcube (S_3) only with dimension-1 neighbors 0010, 0011, 1010, and 1011. $candidate_dim = \{(1, 1)\}$. (1, 1) is selected, $dim = \{0, 3, 1\}$ and $S_3 = *0**$.
- Step 5:** S_4 is not fault-free, so return S_3 .

Algorithm II starts searching for fault-free subcubes from small sized subcubes. It works better than Algorithm I in terms of search complexity when the maximal fault-free subcube is smaller, i.e., when the hypercube has more faulty links. As for Algorithm II, the new dimension neighbors always have the minimum number of faulty adjacent links. If there is more than one candidate direction (i.e., a dimension with the minimum t in $candidate_dim$, as in Step 3 above), then one will be randomly chosen. Since the unchosen candidate directions will not be further checked, Algorithm II involves less complexity but is likely to run into and obtain a “local” maximal fault-free subcube in some rare situations.

It turns out that the maximal fault-free subcube in the faulty 4-cube for both approaches is S_3 (i.e., subcube $*0**$). Each node in S_3 (each black node shown in Fig. 5) can be viewed as a subsink for each of the 8 disjoint 1-subcubes (0*01, 1*01, 1*11, 0*11, 0*10, 1*10, 1*00, and 0*00). The 4-cube is partitioned in this way into 8 disjoint 1-subcubes, and the search for the communication subtrees with the shortest data transmission time can be initiated in parallel in these 1-subcubes using the GA or SA approach. The objective communication subtrees found in this way can be handily combined into a final communication tree in the 4-cube. The subresults in each 1-subcube are then collected into the subsink through each subtree, and the results in all the subsinks are then collected to the sink through any tree in the fault-free 3-subcube $*0**$. As a result of this parallel search for communication subtrees in the disjoint subcubes, the search time (for attaining the final communication tree) can be further reduced. The reduction is even more obvious for hypercubes of higher dimension.

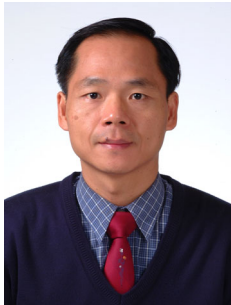
6. CONCLUSIONS

The search for a fault-tolerant communication tree with the minimum data transmission time in a faulty hypercube is desirable as it can accelerate the collection of subresults and, as a result, enhance the overall performance of the system. Various approaches, such as the exhaustive (EX) approach and the minimal cost (MC) approach, can be employed to construct such purposed communication trees. The EX approach finds an optimal communication tree (with the fewest link failures) by checking all possible sinks and dimension orders, one by one, but the operations involved are quite complicated and time consuming. The MC approach, when used to construct a communication tree, requires that all adjacent links of the sink be fault-free. To meet this requirement, the number of faulty links in a hypercube must be kept under 2^{n-1} , and the constructed communication tree may not be an optimal one in terms of the data transmission time. Thus, we have proposed using two random search techniques – the genetic algorithm (GA) and simulated annealing (SA) – to search for more desirable communication trees in hypercubes. When we employ the GA to generate an optimal fault-tolerant communication tree in a hypercube, we first choose the node with the fewest faulty adjacent links as the sink and encode all the dimensions into a string. Then, some strings are randomly generated as the initial population, and the three genetic operations – selection, crossover and mutation – are performed to locate an excellent communication tree. To employ the SA approach to generate an optimal communication tree in a hypercube, we let the solution be encoded in the same way as the chromosome in the GA approach (that is, a dimension-encoded string). Each dimension number in the string is called an *element* of the encoded solution. The objective function is defined to calculate the data transmission time needed to collect data into the sink through the communication tree; therefore, its value needs to be minimized. Extended simulation runs have been conducted to evaluate the performance of the EX, MC, GA, and SA approaches in locating a desired communication tree. The simulation results show that, with negligibly more search time, the proposed GA and SA approaches are able to locate better communication trees than the MC approach. When the sizes of the hypercubes and the number of faulty links increase, the GA and SA approaches can work even better; that is, the data transmission time of the constructed communication trees turns out to be even shorter. To reduce the search time for the GA and SA approaches, we have also presented a distributed approach which partitions an n -cube into several disjoint k -subcubes so that the search for objective communication subtrees with the shortest data transmission time can be simultaneously activated in these k -subcubes to bring down the time complexity. In order to partition the n -cube into disjoint k -subcubes, two algorithms (one using bit comparison and the other using dimension screening) have been presented and employed to find a fault-free $(n - k)$ -subcube of the highest possible dimension. As a result of this parallel search for communication subtrees in disjoint subcubes, the search time for attaining the final communication tree can, consequently, be shortened.

REFERENCES

1. Y. R. Leu and S. Y. Kuo, "A fault-tolerant tree communication scheme for hypercube systems," *IEEE Transactions on Computers*, Vol. 45, 1996, pp. 641-650.

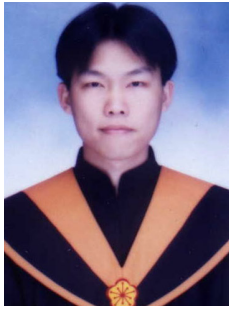
2. J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich., 1975.
3. M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," *IEEE Computer*, Vol. 27, 1994, pp. 17-26.
4. S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, Vol. 220, 1983, pp. 671-680.
5. A. C. Elster and A. P. Reeves, "Block-matrix operations using orthogonal trees," in *Proceedings of SIAM 3rd International Conference on Hypercube Multiprocessors*, 1988, pp. 1554-1561.
6. M. A. Sridar and C. S. Raghavendra, "On finding maximal subcubes in residual hypercubes," in *Proceedings of 2nd IEEE Symposium on Parallel and Distributed Processing*, 1990, pp. 870-873.
7. H. L. Chen and N. F. Tzeng, "Quick determination of subcubes in a faulty hypercube," in *Proceedings of 21st International Conference on Parallel Processing*, 1992, pp. 338-345.
8. F. Ozguner and C. Aykanat, "A reconfiguration algorithm for fault tolerance in a hypercube multiprocessor," *Information Processing Letters*, Vol. 29, 1988, pp. 247-254.



Po-Jen Chuang (莊博任) received the B.S. degree from National Chiao-Tung University, Taiwan, Republic of China, in 1978, the M.S. degree in Computer Science from the University of Missouri at Columbia, U.S.A., in 1988, and the Ph.D. degree in Computer Science from the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, U.S.A. (now the University of Louisiana at Lafayette), in 1992. Since 1992, he has been with Tamkang University, Taiwan, where he is currently a professor in the Department of Electrical Engineering. He was the department chairman from 1996 to 2000. His main areas of interest include parallel and distributed processing, fault-tolerant computing, computer architecture, and mobile computing. Dr. Chuang is a member of the IEEE, the IEEE Computer Society, the ACM, and the IICM.



Shih-Yuan Chen (陳世元) received the B.S. and M.S. degrees in Electrical Engineering respectively from Da-Yeh University and Tamkang University, Taiwan, in 1999 and 2001. He is currently with Winmate Communication Inc., Taiwan, where he involves the development of the high-performance embedded system integrating the WLAN and GPS communication technologies. His research interests include parallel processing, computer architecture, and mobile communication.



Juei-Tang Chen (陳瑞堂) received the B.S. and M.S. degrees in Electrical Engineering respectively from Chinese Culture University and Tamkang University, Taiwan, in 1997 and 1999. He is currently with Winbond Electronics Corp., Taiwan. His research interests include parallel processing and computer architecture.