Pergamon

Computers Math. Applic. Vol. 29, No. 10, pp. 85–104, 1995

0898-1221/95 $9.50 + 0.00

0898-1221(95)00049-6

A Heuristic Algorithm for the Reliability-Oriented File Assignment in a Distributed Computing System

D.-J. CHEN, W. C. HOL AND R.-S. CHEN†
Institute of Computer Science and Information Engineering
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

D. T. K. CHEN
Department of Computer & Information Science
Fordham University, Bronx, NY, U.S.A.

(Received and accepted February 1994)

Abstract—Distributed Computing Systems (DCS) have become a major trend in today's computer system design because of their high speed and high reliability. Reliability is an important performance parameter in DCS design. Usually, designers add redundant copies of software and/or hardware to increase the system's reliability. Thus, the distribution of data files can affect the program reliability and system reliability. The reliability-oriented file assignment problem is to find a file distribution such that the program reliability or system reliability is maximized.

In this paper, we develop a heuristic algorithm for the reliability-oriented file assignment problem (HROFA), which uses a careful reduction method to reduce the problem space. Our numerical results indicate that the HROFA algorithm obtains the exact solution in most cases and the computation time is significantly shorter than that needed for an exact method. When HROFA fails to give an exact solution, the derivation from the exact solution is very small.

Keywords—File assignment, Distributed computer system (DCS), Memory capacity constraint, Heuristic, Program reliability.

1. INTRODUCTION

Distributed computing systems (DCS) have become increasingly popular in recent years, for the advent of VLSI technology and low-cost microprocessors has made distributed computing economically practical in today's computing environment. The DCS provides potential increases in reliability, throughput, fault tolerance, resource sharing and extendibility [1–5]. To improve these performance characteristics, we require a careful design of the DCS. To increase reliability, we can add redundant copies of hardware and software, such as processing elements (PE), programs, and data files in different processors. The distribution of data files can affect the program reliability and overall reliability in the DCS. Hence, an important problem in DCS design is to find a data file distribution that maximizes a certain reliability measure.

Several network reliability measures have been defined and associated evaluation methods have been developed. Two of them, Distributed Program Reliability (DPR) [6,7] and Distributed System Reliability (DSR) [8–10], are adopted in this paper. For a given distribution of programs and data files in a DCS, DPR is the probability that a given program can be run successfully and will be able to access all the files it requires from remote sites in spite of faults occurring among

†Author to whom all correspondence should be addressed.

the processing elements and communication links. The second measure, DSR, is defined to be the probability that all the programs in the system can be run successfully.

The file assignment problem and related problems such as task assignment and job scheduling have been studied for many years [11–14]. They have been studied by using techniques from graphic theory, queuing theory, mathematical programming, and various heuristic and algorithmic techniques.

The file assignment problem is a special case of the task assignment problem. The problem we are concerned with in this paper is to assign files in a DCS so that all the programs are allocated to reading data or outputting data. The file assignment problem is inherently NP-complete [15] in complexity. This implies that optimum solutions can be found only for small problems. For larger problems, it is necessary to introduce heuristics to produce algorithms which generate near-optimum solutions. Several techniques, such as dynamic programming [16], branch-and-bound [17], backtracking [18] and heuristic programming [19], can be used to avoid the complete enumeration of the problem space. The choice of a particular technique depends on the structure of the problem.

The reliability-oriented file assignment problem has been studied for many years. In [18], a reliability-oriented file assignment algorithm (ROFA) was proposed to solve the optimal file assignment problem under a memory space constraint. That method first generates all the maximum feasible file combinations (MFFC) of each node, then constructs a space state tree according to each nodes' MFFCs and travels the state space tree in a depth-first manner by applying a back-tracking algorithm.

The back-tracking algorithm first finds a feasible solution as a lower bound and then back tracks to level $n-1$ of the space state tree. If the reliability upper bound (let the unvisited child nodes contain all required files) is smaller than the lower bound, the downward searching in the space state tree is fathomed. The reliability was measured by the SYREL algorithm [20]. Although this method is capable of finding the optimum solution, it is not efficient, so it is probably not a practical approach.

In this paper, we present a heuristic algorithm for the reliability-oriented file assignment problem (HROFA) which uses a careful heuristic pruning method to reduce the solution space. Unlike other assignment strategies, the proposed reduction method is a reasonable one. The HROFA algorithm works in a manner similar to that used to find a minimal file spanning tree; the complete algorithm and the justification for our reduction techniques are described in this paper. Numerical results show that the HROFA algorithm obtains the exact solution in most cases, and when it fails to give an exact solution, the deviation from the exact solution is quite small.

The organization of the rest of this paper is as follows. In Section 2, the problem statement, notation, and definitions that will be used throughout this paper are given. Section 3 states the reliability oriented file assignment problem in distributed computing systems. The derivation, correctness, and some examples of application of the HROFA algorithm are described in Section 4. Section 5 concludes the paper.

2. PROBLEM STATEMENT, NOTATION, AND DEFINITIONS

PROBLEM STATEMENT. The reliability-oriented file-assignment problem can be characterized as follows:

GIVEN.

Network topology
Distribution of programs in the network
Files required by programs for execution
The size of each file
The available memory space of each processing element
The reliability of each communication link

CONSTRAINT. The limitation of memory space of each processing element

VARIABLE. File assignment

GOAL. Maximize DPR of a given program (or Maximize DSR of the system)

NOTATION AND DEFINITIONS.

$G(V, E)$	An undirected graph in which V represents the node set of processing elements and E represents the edge set of communication links for the network under consideration
N_i	a node i in V
PRG	the set of programs allocated in the network for execution
F_s	the set of files required by PRG
PRG_p	a program p in PRG
DPR_p	the reliability of distributed program p
F_i	the file i in F_s
FN_p	the set of files required by PRG_p for execution
FST	a spanning tree that connects the root node (processing elements that runs the program under consideration) to other nodes such that its vertices hold all the needed files
MFST	an FST such that there exists no other FST which is a subset of it
MFFC	a feasible file combination such that there exists no other feasible file combination which is a superset of it
n	the number of nodes in G; $n = \|V\|$
k	the numbers of files in F_s
$p(q)$	the probability that the communication link works (fails)
I_p	the index set of FN_p
s_i	the size of F_i
C_i	the available memory space of N_i
$x_{i,j}$	the indicator of file assignment $x_{i,j} = 1$ if F_j is assigned to N_i, else 0
$X_i^{(t)}$	a feasible file combination of N_i; $X_i^{(t)} = (x_{i1}^{(t)}, x_{i2}^{(t)}, \ldots, x_{ik}^{(t)})$
FA_i	a set of MFFCs for N_i
$E(\text{PRG}_p)$	event that PRG_p can successfully run and files in FN_p can be successfully accessed by PRG_p
$\Pr(E)$	probability of event E
$P(i, k)$	a two-dimensional array such that if there is a solution to the file combination problem with the first i elements and size k

3. THE RELIABILITY-ORIENTED FILE ASSIGNMENT PROBLEM IN DISTRIBUTED COMPUTING SYSTEM

We shall now formally define the reliability-oriented file assignment problem. The reliability oriented file assignment problem can be stated mathematically as follows:

PROBLEM 1. *(Maximizing* DPR *subject to memory space constraint)*

$$\text{Maximize } \text{DPR}_p = \Pr\left[E\left(\text{PRG}_p\right)\right]$$

$$\text{subject to } \begin{cases} \sum_{j=1}^{k} s_j x_{ij} \le C_i, & i = 1, 2, \ldots, n. \\ \sum_{i=1}^{n} X_{ij} \ge 1, & \\ x_{ij} = 0 \text{ or } 1, \quad i = 1, \ldots, n. & \end{cases}$$

PROBLEM 2. *(Maximizing DSR subject to memory space constraint)*

$$\text{Maximize DSR} = \Pr\left[\bigcap_{i=1}^{l} E(\text{PRG}_i)\right]$$

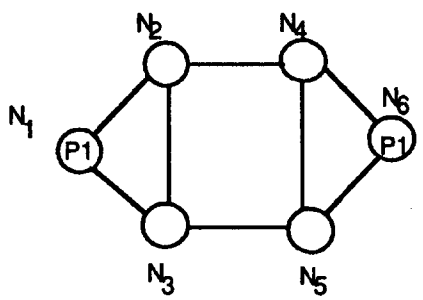
$$\text{subject to} \begin{cases} \sum_{j=1}^{k} s_j x_{ij} \le C_i, & i = 1, 2, \ldots, n. \\ \sum_{i=1}^{n} X_{ij} \ge 1, & j = 1, 2, \ldots, k. \\ x_{ij} = 0 \text{ or } 1, \quad i = 1, \ldots, n, \quad j = 1, 2, \ldots, k. \end{cases}$$

First, we present a back-tracking algorithm [18] for solving Problem 1. The algorithm has two steps:

(1) For each node N_i, find all of the maximal feasible file combinations (MFFC).

(2) Apply a back-tracking algorithm to find the optimal file assignment.

The following numerical example illustrates the operation of the back-tracking algorithm. Consider the distributed processing system shown in Figure 1, which consists of six nodes and the PRG_1.



P1 needs F1, F2, F3
File size: 2, 3, 5
Node capacity: 2, 3, 5, 4, 5, 2
for N1,N2,N3,N4.N5 and N6 respectively
Edge reliability: 0.9

File Combinations:

FA1= { (1,0,0) }
FA2= { (0,1,0), (1,0,0) }
FA3= { (1,1,0), (0,0,1) }
FA4= { (0,1,0), (1,0,0) }
FA5= { (1,1,0), (0,0,1) }
FA6= { (1,0,0) }

Figure 1. A simple DCS for illustration of the back-tracking algorithm.

Two copies of program PRG_1 are allocated in node N1 and N6, respectively. The files required for executing program PRG_1 are F1, F2, and F3. The file sizes of F1, F2, and F3 are 2, 3, and 5. Assume that all the communication links have the same reliability, 0.9. The available memory space for N1 to N6 is $M_1 = 2$, $M_2 = 3$, $M_3 = 5$, $M_4 = 4$, $M_5 = 5$, and $M_6 = 2$. In Step 1, we generate all MFFCs for each node. These are $FA_1 = \{(1,0,0)\}$, $FA_2 = \{(0,1,0), (1,0,0)\}$, $FA_3 = \{(1,1,0), (0,0,1)\}$, $FA_4 = \{(0,1,0), (1,0,0)\}$, $FA_5 = \{(1,1,0), (0,0,1)\}$, and $FA_6 = \{(1,0,0)\}$. The nodes in Figure 2 have been numbered according to the sequence of the back-tracking procedure. The bounding function is applied at number 9 of the state space tree. The reliability upper bound 0.982 is less than the lower bound 0.988 found so far, so then node 9 is fathomed. The more precise upper bound will be estimated at the lower level of the state space tree. The optimal solution is $[(x_{11}, x_{12}, x_{13}), (x_{21}, x_{22}, x_{23}), (x_{31}, x_{32}, x_{33}), (x_{41}, x_{42}, x_{43}), (x_{51}, x_{52}, x_{53}), (x_{61}, x_{62}, x_{63})] = [(1,0,0), (0,1,0), (1,1,0), (0,1,0), (0,0,1), (1,0,0)]$. The optimal value of DPR_1 is equal to 0.988.

The back-tracking process is an elegant method. In real cases, however, the reliability difference between all feasible solutions is not so clear. A small reliability difference is easily obtained by the reliability contribution of the Fullfile nodes (the unvisited nodes) which are assumed in the back-tracking algorithm. The more Fullfile nodes a state assumes, the higher its reliability. So

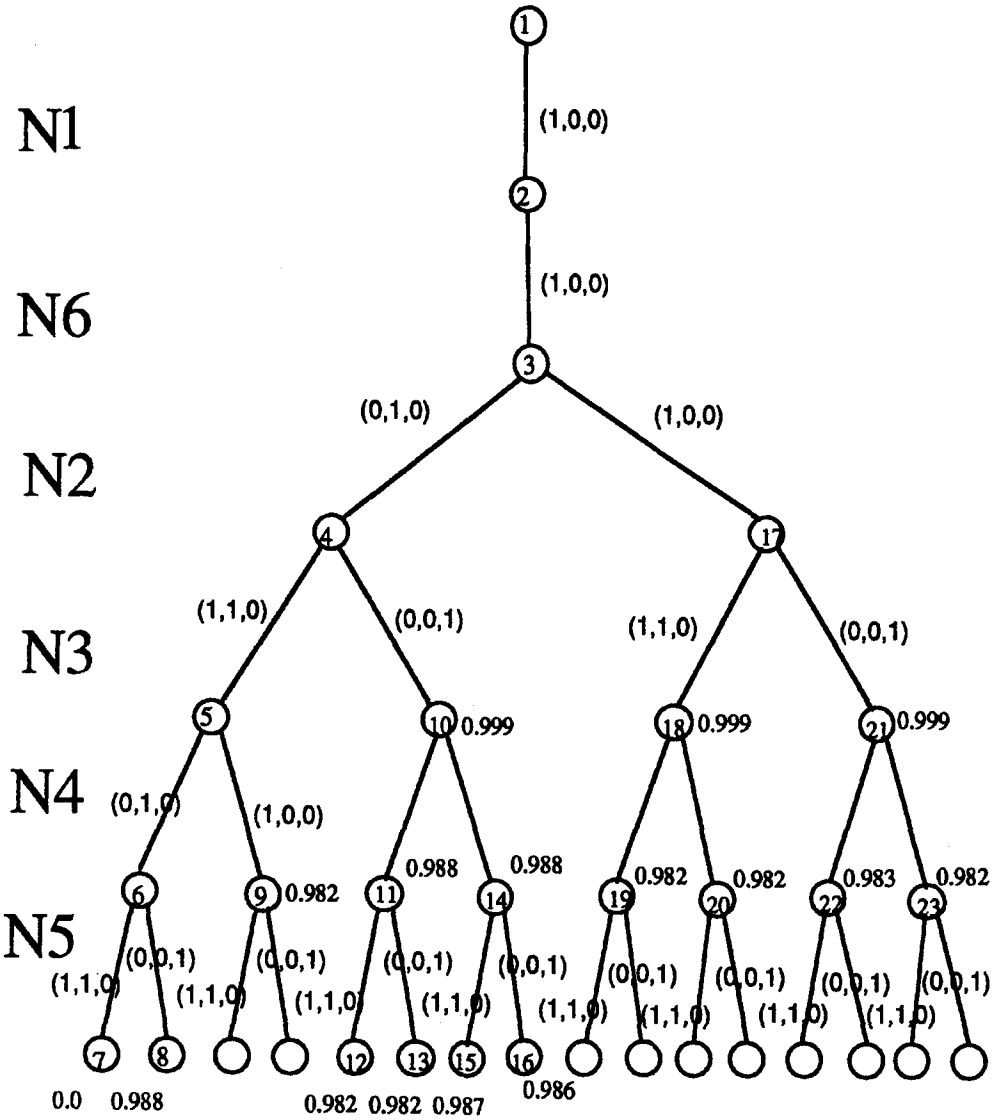


Figure 2. Generation of the state space tree for Figure 1.

the pruning will rarely occur in the high levels of the state space tree; most of the pruning will occur in the last few levels of the state space tree (because there are fewer Fullfile nodes in that pathset). Hence, this method may take more time than an exhaustive search. The example shown in Figure 2 provides the evidence for this conclusion (there are only 16 pathsets, but the back-tracking algorithm travels 23 states). So the branch-and-bound method is not well suited for the ROFA problem.

Also, the connection of a network has an important influence on the certain reliability. This fact motivates us to develop a heuristic algorithm, which we call HROFA, that uses information on network connections and analyzes DPR formulas to avoid exhaustive enumeration of the state space tree.

4. DERIVATION OF THE HROFA ALGORITHM

Nair [12] proposed a heuristic method for choosing the pathset with the highest possible reliability (under some assignment). Let the path be assigned according to this method. Then choose the pathset with the next highest reliability and assign the path to it, and so on. The maximal error rate of this method is under 4.8%, and in most cases, it successfully derives the correct answer.

We can start spanning our state space tree by constructing the most reliable MFST. That is, spanning the state space tree is just like finding its MFST. Using this basic idea for analyzing the DPR formula, we propose a heuristic algorithm for the reliability oriented file assignment problem. We call the algorithm HROFA (Heuristic algorithm for ROFA).

4.1. The Proposed Heuristic Algorithm (HROFA)

In the HROFA algorithm, we span the state space tree like ROFA does. We reduce the state space tree in a top-down manner by checking its file combination, and the nodes are spanned in different order. The following is an outline of HROFA:

(1) Generate the spanning order of each node.
(2) Perform HROFA algorithm to span the state tree.

STEP 1.

(a) Calculate the spanning order of each node.
The order is measured by the node reliable degree (RD).
$\mathrm{RD}_i = \overline{X_{s,i}} * \overline{F_{f_n \cap f_i}}$
where $\overline{X_{s,i}}$ is the average link reliability from Node i to Node s (the starting nodes).
$\overline{F_{f_n \cap f_i}}$ is the average number of needed files contained in Node i.
Example: If the MFFCs of N_i are (1,1,0), (0,0,1) and the file needed is (1,1,1), then $\overline{F_{f_n \cap f_i}} = (2+1)/2 = 3/2$.
The $\overline{X_{s,i}} = 0.9$ (suppose the reliability of all links is 0.9), and $\mathrm{RD}_i = 0.9 * 3/2 = 1.35$.

(b) Find the node that has the highest RD and add this node to StartNode. Repeat the process until the spanning order of all nodes is found.

For example, consider the network below:

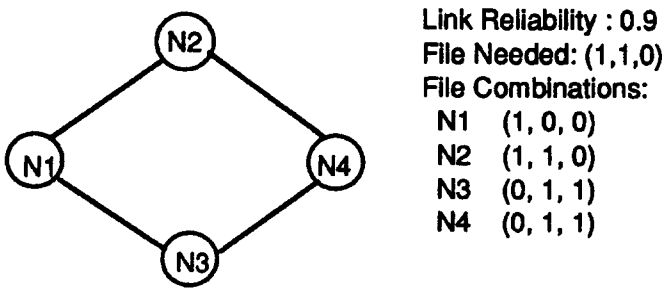


Figure 3. A simple DCS.

Since the link reliability is 0.9, the $\overline{X_{s,i}}$ for each node is 0.9, and the $\overline{F_{f_n \cap f_i}}$ for each node is {1,2,1,1}.

Table 1. The generation of spanning order for Figure 3.

StartNode	1	2	3	4
1	–	0.9 * 2	0.9 * 1	–
1, 2	–	–	0.9 * 1	0.9 * 1
1, 2, 3	–	–	–	0.9 * 1
1, 2, 3, 4	–	–	–	–

So the spanning order is N1, N2, N3, N4. Let the node sequence be $N_{i1}, N_{i2}, N_{i3}, \ldots N_{in}$, and span the state space tree in that sequence. Spanning to some node, if we find paths that contain all the needed files, then we delete their brother paths which do not contain all the needed files, i.e., we mask the $F_{N_{i1}}$. We then continue spanning, and if we find pathsets that can be reduced, then we mask $F_{N_{i2}}$. We repeat the above reduction method until no pathset can be reduced or all nodes have been masked.

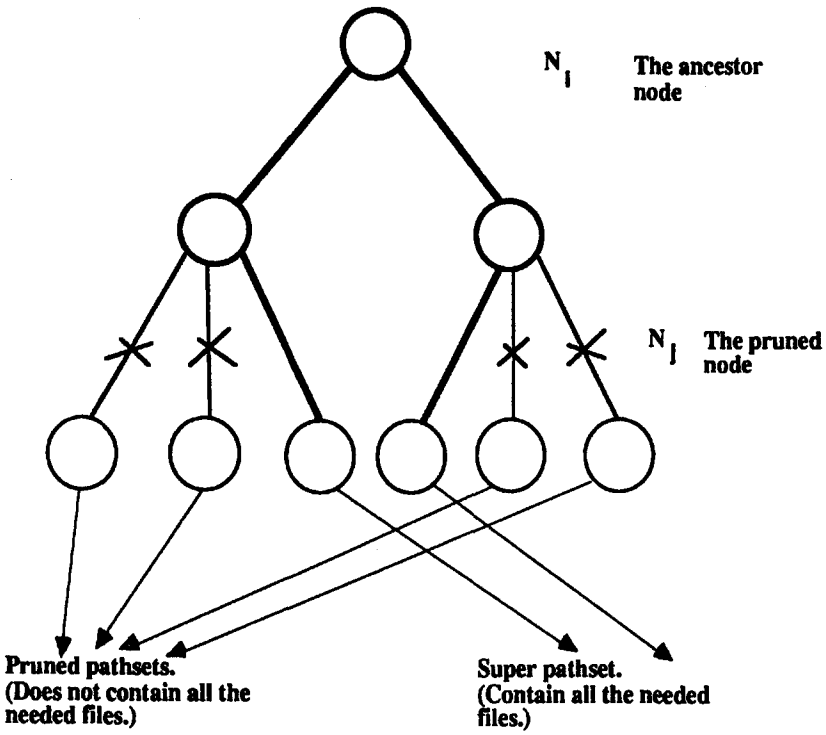


Figure 4. The pruned pathsets and the super pathsets.

4.2. Algorithm

Now we present a heuristic algorithm for computing the ROFA problem under memory space constraints. This is an enumerative algorithm which uses heuristic reduction to reduce the state space tree. The algorithm consists of four steps, as follows:

Step 0. Initialization.
Step 1. Generate all MFFCs of all nodes.
Step 2. Generate the spanning order of each node.
Step 3. Span the state space tree by the spanning order and perform heuristic reduction.
Step 4. Compute the reliability of each pathset spanned in Step 3 and output the best file assignment.

Step 0. Initialization

We read the data from the file which contains the system parameters to obtaining the following information:

N = number of nodes
L = number of links
F = number of files
P = number of programs
$C(i)$ = capacity of node i
$S(i)$ = size of file i
$PN(i)$ = file needed for program i to be executed

Step 1. Generate all MFFCs of all nodes

We solve this problem using a dynamic programming technique. Since the problem constraints are

$$X_1S(1) + X_2S(2) + \cdots + X_FS(F) \leq C(i),$$

where $X_j = 1$, if file j is contained in node, else 0.

It can be divided into several subproblems as follows:

$$\begin{aligned} X_1S(1) + X_2S(2) + \cdots + X_FS(F) &= C(i) \\ X_1S(1) + X_2S(2) + \cdots + X_FS(F) &= C(i) - 1 \\ &\vdots \\ X_1S(1) + X_2S(2) + \cdots + X_FS(F) &= C(i)\text{-MaxFileSize}. \end{aligned}$$

Each of these problems is just the Knapsack problem and the feasible solution is a file assignment of it.

($C(i)$-MaxFileSize) is set as a bound because, if there exists an MFFC in which

$$X_1S(1) + X_2S(2) + \cdots + X_FS(F) < C(i)\text{-MaxFileSize},$$

then we can add another file not included in this MFFC and the total file size will still be smaller than $N(i)$, which is a contradiction. So there is no MFFC with total size smaller than ($C(i)$-MaxFileSize).

We use a dynamic programming technique to generate a table which indicates if there exists a solution in the size of (see Table 2).

Table 2. Table generated by Knapsack algorithm.

	0	1	2	3	4	5
S_1	O	–	I	–	–	–
S_2	O	–	O	I	–	I
S_3	O	–	O	O	–	O

'I': a solution containing this item has been found
'O': a solution without this item has been found
'–': no solution of this size T

If there is a solution in the entry $P(i,k)$, then we will check whether $P(i-1, k-S_i)$ has a solution in it. If so, we continue checking until we check $P(0,0)$. When we find a feasible file combination, we will check whether it is covered by or covers the FFCs found before. We then delete the FFC if it is covered by other FFCs, or add the FFC to FOUND, if it is not covered. The following is a formal description of Step 1:

MFFC(K).

/*S(the array that stores the file size). K, the node capacity, P (a two-dimensional array such that $p[i,k]E$ = true if there exists a solution to the file combination problem with the first i elements and size k, and $P[i,k]B$ = true if the i^{th} element belongs to that solution*/

```
begin
    if ∑ S[i] ≤ K then
    begin
        MFFC = (1,1,1,...)
        return
    end
    Knapsack(K)
    FFC = 0
```

```
        for t = K downto K-MaxFileSize do
                check(K, t, FFC)
end
function Knapsack(K)
begin
        P[0,0]E = true
        for k = 1 to K do
                P[0,k]E = false
        for i = 1 to F do
                for k = 0 to K do
                        P[i,k]E = false
                        if P[i-1,k]E then
                        begin
                                P[i,k]E = true
                                P[i,k]B = false
                        end
                        else if k - S[i] ≥ 0 then
                                if P[i-1,k-S[i]]E then
                                begin
                                        P[i,k]E = true
                                        P[i,k]B = true
                                end
                od
end
function CHECK(K, t, FFC)
begin
        if K = 0 then
                check if FFC is covered or covers other elements in FOUND, add
                to FOUND if the FFC is not covered
        for i = t to N do
                if P[t,K]E then
                begin
                        FFC| = 1 ≪ i
                        check(K - S[t], t - 1, FFC)
                end
                else if K - S[t] ≥ 0 then
                begin
                        FFC| = 1 ≪ i
                        check(K - S[t]), t, FFC)
                end
        od
end
```

Step 2. Generate the spanning order of each node

In Step 2, we choose a starting node which contains the program to be executed and add that starting node to StartNode(0). We then compute the Reliable degree (RD) of each node to the StartNode and choose the most reliable node to add to StartNode(1). According to the new StartNode, we find the most reliable node from among the rest of the nodes and add it to StartNode(2). We repeat the process until all nodes have been added to StartNode. The StartNode records the spanning order of each node.

Step 3. Perform HROFA algorithm

The HROFA algorithm acts as follows: According to the spanning order, we span the state space tree in DFS manner. While spanning to a node, if we find there exist pathsets which contain all the needed files, we eliminate the other pathsets that do not contain all the needed files. Then we mask the MFFCs of the node which is in StartNode(1) and continue spanning to other nodes not yet spanned. If we find there exist pathsets that contain all the needed files, we cut the other pathsets which do not contain all the needed files. We then mask the MFFCs in StartNode(2), and continue the spanning and cutting process described above. We repeat the process until all MFFCs of each node have been spanned. The formal HROFA algorithm is given below.

```
HROFA (SPANNODE, MASKNODE, PATHSET).
/* The SpanNode is the node to be spanned. The MaskNode is the node whose MFFCs are
masked */
begin
    if SpanNode = 0 then
    begin
        add the Pathset to FOUND
        return
    end
    for all the MFFCs of the SpanNode do
        if the file included in the pathset contains all the needed files then
        begin
            temp = Pathset | the MFFC
            HROFA(the next node of the SpanNode, the
                next node of the MaskNode, temp)
        end
    od
    if there does not exist a pathset which contains all the needed files then
    begin
        for all MFFC in SpanNode do
            temp = Pathset | the MFFC
            HROFA(the next node of SpanNode, MaskNode, temp)
        od
    end
end
```

Step 4. Compute the reliability of each pathset

Since the system parameters have been read in Step 0, we know the network topology, the program distribution, the files needed for each program to be executed, and the link reliability. We still need to know the file distribution of each node in order to compute the pathset reliability. In this step, we simply pass the file distribution of each node according to the pathset in the FOUND. Then we call the reliability evaluating program FREA [21] to compute the pathset reliability. After computing all the reliabilities, we will output the file assignment with the highest reliability.

The complete algorithm

A formal description of the HROFA algorithm is given below.

HROFA ALGORITHM.

STEP 0. Initialization.

read system parameters
FOUND = $\varnothing$
$FN = \bigcup_{P_j \in P_N} F_j$
find a node x_i in the DCS that contains the program to be executed
StartNode = x_i
MaskNode = 0

STEP 1. Generate MFFCs of all nodes.

for all nodes do
    MFFC(node capacity)

STEP 2. Generate the spanning order of each node.

NextNode(StartNode)

STEP 3. Perform HROFA algorithm.

pathset = 0
HROFA(StartNode, MaskNode, pathset)

STEP 4. Compute pathset reliability.

for each pathset in FOUND do
    call FREA to evaluate the pathset reliability
output the assignment which has the highest reliability

4.3. Examples

We use the DCS shown in Figure 1 as an example to show how the HROFA works.

STEP 0. Initialization.

The system parameters are known. The node capacity for node 1 to node 6 is 2,3,5,4,5,2. The file sizes for F1 to F3 are 2, 3, and 5. All the link reliabilities are the same and equal 0.9. Two copies of program PRG_1 are allocated in node 1 and node 6. The files needed for PRG_1 are F1, F2, and F3.

STEP 1. Generate the MFFCs of each node.

The Knapsack generates a table as follows:

Table 3. Table generated for the example.

NodeSize / FileSize	0	1	2	3	4	5
S_1	O	–	I	–	–	–
S_2	O	–	O	I	–	I
S_3	O	–	O	O	–	O

Node[1]. MFFC = {(1,0,0)}
Node[2]. MFFC = {(0,1,0),(1,0,0)}
Node[3]. MFFC = {(1,1,0),(0,0,1)}
Node[4]. MFFC = {(0,1,0),(1,0,0)}
Node[5]. MFFC = {(1,1,0),(0,0,1)}
Node[6]. MFFC = {(1,0,0)}

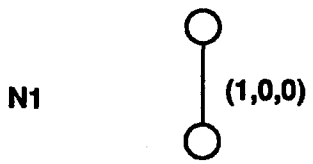
STEP 2. Generate the spanning order of each node.

Table 4. Table generated for the example in Step 2.

StartNodeSet	RD1	RD2	RD3	RD4	RD5	RD6
1	–	0.9	1.35	0	0	0
1,3	–	0.9	–	0	1.35	0
1,3,5	–	0.9	–	0.9	–	0.9
1,3,5,2	–	–	–	0.9	–	0.9
1,3,5,2,4	–	–	–	–	–	0.9
1,3,5,2,4,6	–	–	–	–	–	–

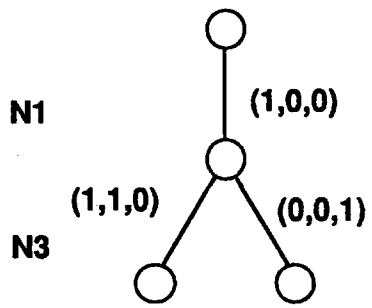
The spanning order is {1, 3, 5, 2, 4, 6}.

STEP 3. Perform heuristic reduction.



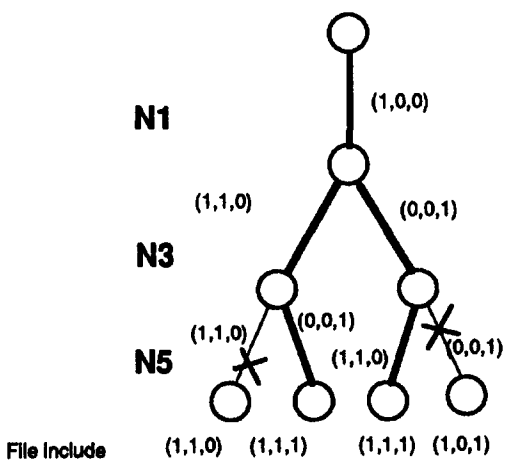
(a) SpanNode = N1, MaskNode = 0.

Figure 5. The result in HROFA Step 3a.



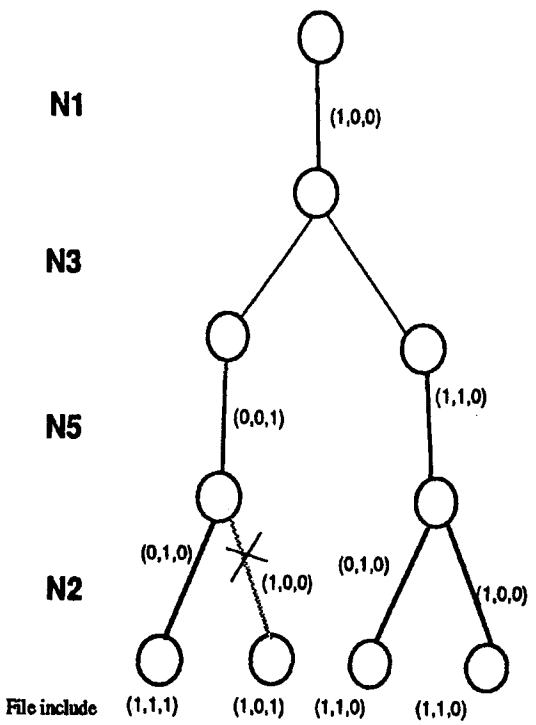
(b) SpanNode = N3, MaskNode = 0.

Figure 6. The result in HROFA Step 3b.



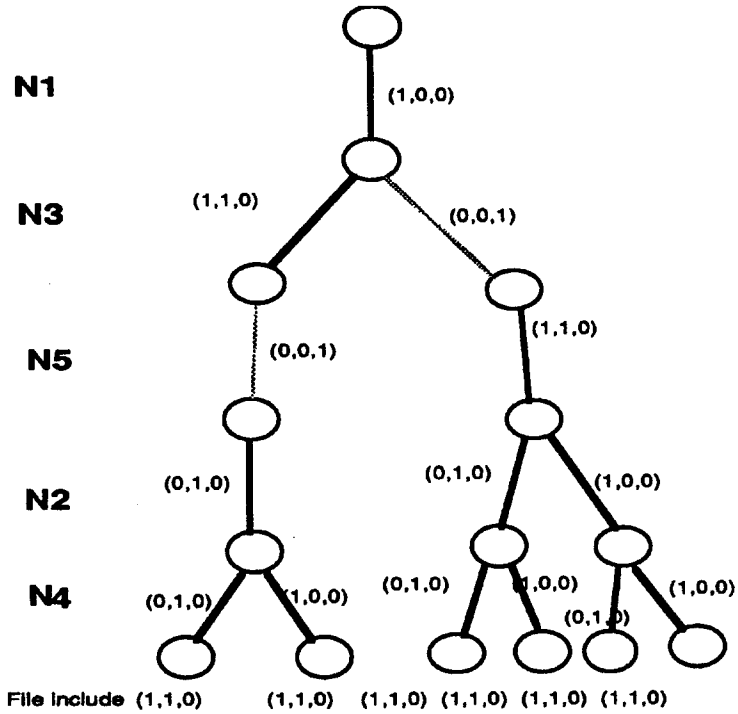
(c) SpanNode = N5, MaskNode = 0.

Figure 7. The result in HROFA Step 3c.



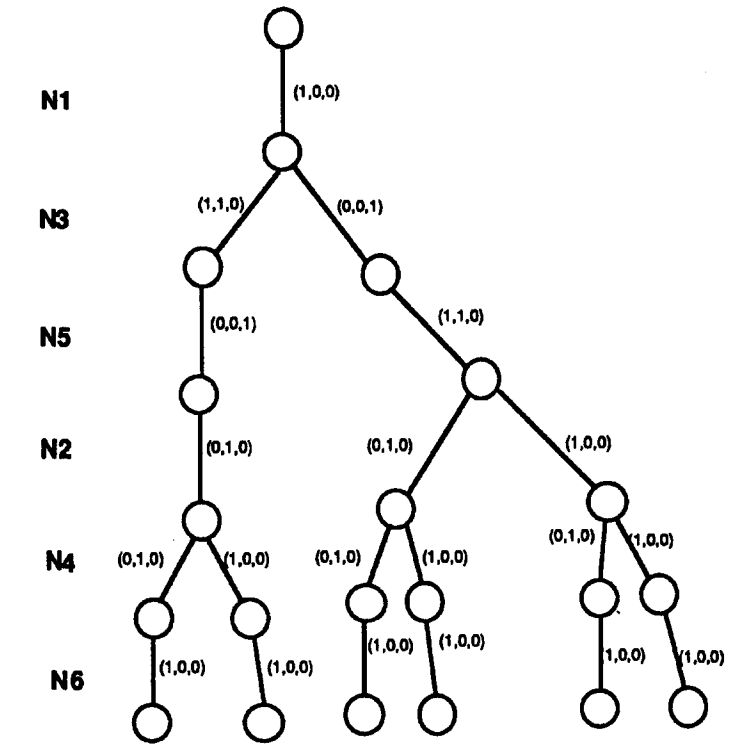
(d) SpanNode = N2, MaskNode = N3.

Figure 8. The result in HROFA Step 3d.



(e) SpanNode = N4, MaskNode = N3, N5.

Figure 9. The result in HROFA Step 3e.



(f) SpanNode = N6, MaskNode = N3, N5.

Figure 10. The result after HROFA Step 3.

STEP 4. Compute the pathset reliability.

The best assignments for N1, N2, N3, N4, N5, N6 are $\{(1,0,0),(0,1,0),(1,1,0),(0,1,0),(0,0,1),(1,0,0)\}$ (see Figure 11).

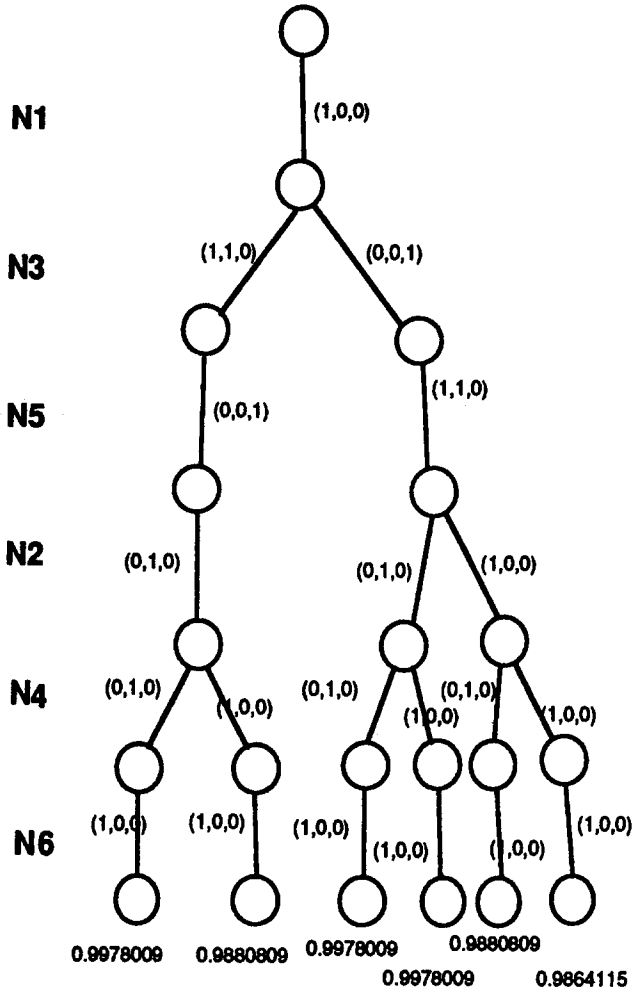


Figure 11. The result in HROFA Step 4.

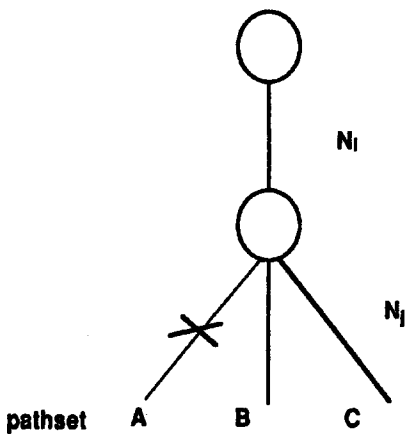


Figure 12. Example showing corresponding pathsets.

4.4. The Correctness of the HROFA Algorithm

To perform the pruning described above, the reliability of each pathset of the pruned subtree must be less than or equal to the corresponding super pathsets.

What are the corresponding pathsets? Consider Figure 12.

Pathsets A, B, and C are corresponding pathsets, i.e., these pathsets differ in only one MFFC, which is on the pruned level (node N_j). This means all these file assignments of the corresponding pathsets are different in only one node and that node is on the level where pruning occurred.

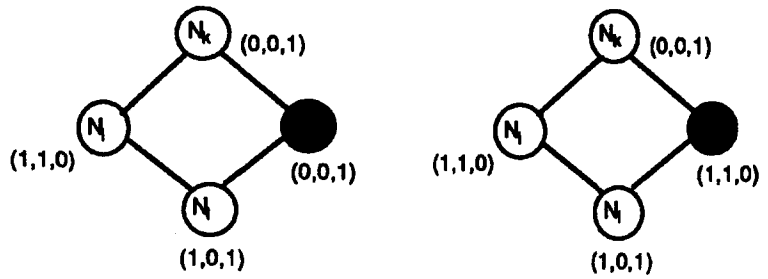


Figure 13. Example showing the different file assignments between corresponding pathsets.

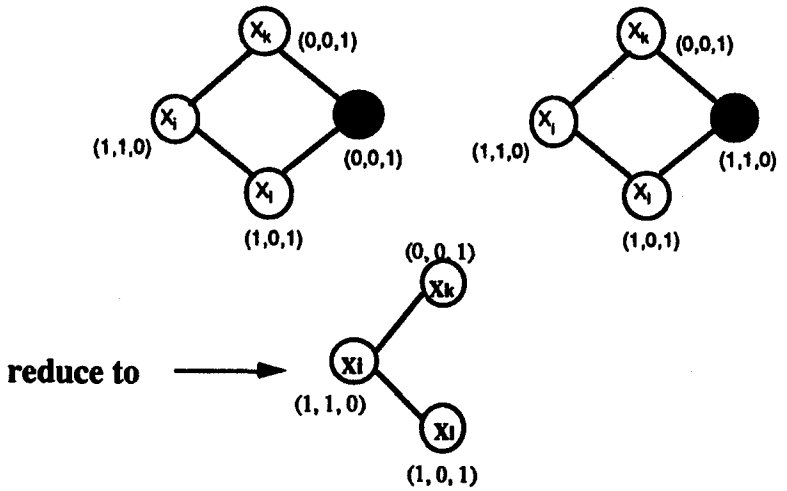


Figure 14. Example showing different file assignments between corresponding pathsets.

In Figure 13, the two networks are different in the file-assignment of N_j. Since there is only one node difference, we divide all the MFSTs of a pathset into three cases. We then observe the following facts:

CASE 1. The MFST does not include the pruned node, and the reliability of the corresponding pathsets is the same.

We can regard the network as if it were reduced to a subgraph (Figure 14). The MFSTs of the corresponding pathsets are generated from the same file assignment, and the pathsets have the same MFSTs. If we span the MFSTs' probability in the same path order, the same MFST should have the same probability. So the total reliability of the MFSTs which do not include the pruned node is the same. We denote the reliability difference between the super pathset and the pruned pathset in Case 1 by D_1.

CASE 2. The MFST contains the pruned node.

We divided the problem into two parts.

CASE 2A. The MFST contains all the ancestor nodes. The reliability of the Super pathset is superior.

Since it spans the state space tree from the start node to all of the ancestor nodes and the pruned node, the super pathset already has all the needed files, but the pruned pathset must connect to other nodes so as to contain all the needed files. So the probability of the MFSTs of a pruned pathset must be smaller than that of the super pathset. For example, from the file assignment of the super pathset, one can generate as MFST like that shown in Figure 15.

As shown in Figure 16, however, the pruned pathset should have more nodes to generate an MFST. Even if the pruned pathset has more MFSTs, its total probability is still smaller than that of the super pathset. No matter how many MFSTs the pruned pathset has (even if it has

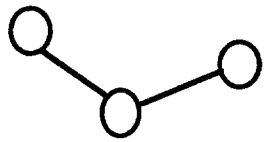


Figure 15. An MFST of the super pathset.

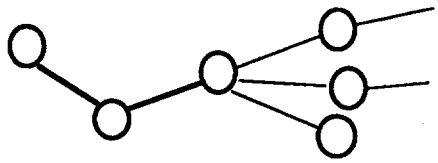


Figure 16. Possible MFSTs of the pruned pathset.

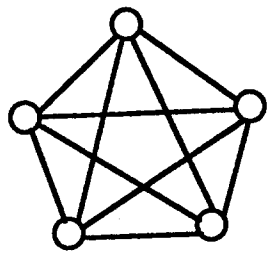


Figure 17. A five-node fully connected network.

infinitely many MFSTs), the total probability is

$$0.9^3 + 0.1 * 0.9^3 + 0.1^2 * 0.9^3 + \cdots = 0.9^2(0.9 + 0.09 + 0.009 + \cdots),$$

which is still smaller than the 0.9^2 of the super pathset (by measuring Figure 16 and Figure 17 and supposing the reliability of all links is 0.9).

We let the reliability difference between the corresponding pathsets in Case 2a be D_a.

CASE 2B. At least one ancestor node is not included in the MFST. The reliability difference between the pathset is small.

The probability of Case 2b is given by a term of the form $\Pi q_i \Pi p_j$. Because each MFST in Case 2b probability derivation is multiplied by more than one term q, the reliability difference between the super pathset and the pruned pathset in Case 2b is very small. Let the difference between the corresponding pathsets be D_b.

Today, link reliability of more than 0.9 is quite common, and the term q is typically smaller than 0.1. Since the probability of each MFST in Case 2b is multiplied by more than one such q, Case 2b contributes less to the probability than Case 2a does. We cannot say the total reliability contribution of Case 2b must be smaller than that of Case 1, for there may be many variations in the network topology, file distribution, and program distribution in a DCS, so there could be exceptions. However, we can say that in most cases the total probability contribution is approximately equal to Case 2b and is about 5% of a system's reliability. So D_b is typically quite small.

The worst case for our reduction could be that the super pathsets of Case 2b have no MFST and the pruned pathsets have many MFSTs. Even in this case, however, the error rate is still quite small (because D_a reduce the error, and D_b is inherently very small, about 5% of system reliability).

The restriction of Case 1 is that one node (the done node) is not included, but in Case 2b, at least one ancestor node is not included and this node is connected to the pruned node. This means that in Case 2b, the DCS is reduced to a subgraph that is smaller than that of Case 1. Each condition in Case 2b is likely to have fewer MFSTs than Case 1, and the probability of

their MFSTs is multiplied by more q's. If the link reliability is 0.9, ten similar MFSTs may be needed to save a q term. In Case 1, the MFST's probability expression could be $0.1^2 * 0.9^3$, but in Case 2b, it is $0.1^3 * 0.9^3$. So ten such MFSTs may be needed. In other words, the reduction process is just like

$$\begin{aligned} \text{DPR} = {} & p^i + q * p^j + q^2 * p^k + \cdots + && \text{HROFA (no mask node)} \\ & q^l * (p^i + q * p^j + q^2 * p^k + \cdots) + && \text{HROFA (mask StartNode[1])} \\ & q^m * (\cdots) + && \text{HROFA (mask StartNode[2])}, \end{aligned}$$

where $l < m < n \ldots$.

We cut the pathsets which are less reliable in each span level. In HROFA (no mask node), we cut the pathsets which have the smaller product for the term $(p^i + q * p^j + q^2 * p^k + \cdots +)$. In HROFA, (mask StartNode[1]), we cut the pathsets which have the same value for term 1 $(p^i + q * p^j + q^2 * p^k + \cdots)$, but have a smaller value for term 2 $\left(q^l * (p^i + q * p^j + q^2 * p^k + \cdots)\right)$. The HROFA (mask StartNode[2]) cuts the pathsets which have the same value for term 1 and term 2 but a smaller value for term 3, and so on. If an exception occurs, it must be that the D_a is quite small and the pruned pathset has many more MFSTs than the super pathset in Case 2b. Such a condition will occur in a fully connected network.

In a fully connected network, there are many MFSTs in each possible file assignment for a certain program. Thus, when we execute the HROFA algorithm, the reliability difference between the corresponding pathsets in Case 2a will be very small. That is, D_a will probably be reduced to be a value like 0.0000081, or else D_a will no longer be a great advantage to D_b, for the pruned pathset would probably have many more MFSTs than the super pathset in such a topology.

In such a topology, the most important parameter influencing the reliability is the load balanced. Because there are many paths that connect two different nodes, under the limitation of memory space constraints, the more load-balanced the network is, the more choices there are for a program to access the needed files, and thus the more MFSTs exist.

To sum up, our method is to reduce the pathsets whose reliability in Case 1 (about 90% of system reliability) is smaller than that of the other super pathsets, and thus for which the difference D_a is bigger than D_b. It is highly unlikely that the reliability of the reduced pathset will be greater than that of the super pathsets. This is the main justification for this reduction method.

4.5. Some Numerical Results

We now compare the reliability given by the HROFA algorithm with the optimal reliability obtained by complete enumeration. For a given network topology, we compare the difference between the optimal solution for the reliability and the reliability obtained by the HROFA algorithm under variations in the program distribution and link reliability.

EXAMPLE 1.

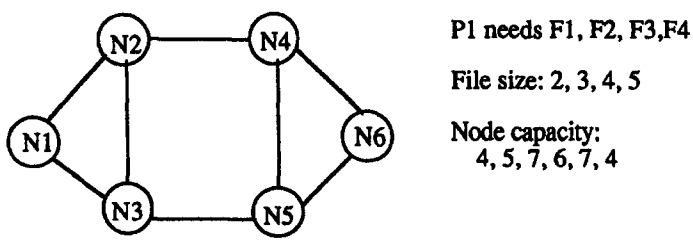


Figure 18. A six node network topology.

Table 5. The numerical result for Example 1.

P1 in	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
$p = 0.9$ reliability difference	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
$p = 0.8$ reliability difference	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
$p = 0.7$ reliability difference	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
OPT reliability						
$p = 0.9$	0.9882000	0.9987948	0.9988119	0.9999118	0.9999579	0.9998462
$p = 0.8$	0.9472000	0.9887744	0.9892352	0.9975398	0.9987379	0.9968845
$p = 0.7$	0.8722000	0.9575776	0.9604693	0.9839450	0.9820002	0.9820002

Total pathsets: 1296
After HROFA: 108

EXAMPLE 2.

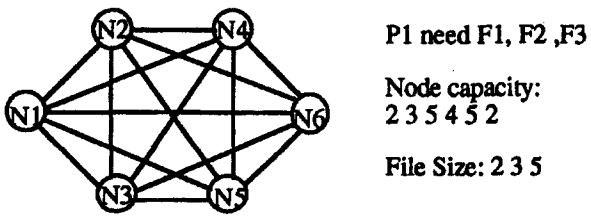


Figure 19. A six node fully connected network topology.

Table 6. The result for Example 2.

P1 in	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)
$p = 0.9$ reliability difference	0.0000100	0.0000100	0.0000010	0.0000100	0.0000010	0.0000100
$p = 0.8$ reliability difference	0.0003228	0.0003219	0.0003040	0.0002040	0.0003160	0.0003240
$p = 0.7$ reliability difference	0.0024593	0.0024351	0.0022844	0.0024351	0.0022844	0.0024847
OPT REL						
$p = 0.9$	0.9999899	0.9999999	0.9999999	0.9999999	0.9999999	0.9999999
$p = 0.8$	0.9996621	0.9999869	0.9999869	0.9999869	0.9999869	0.9999895
$p = 0.7$	0.9971307	0.9996398	0.9996398	0.9996398	0.9996398	0.9997050

Total pathsets: 1296
After HROFA: 108

EXAMPLE 3.

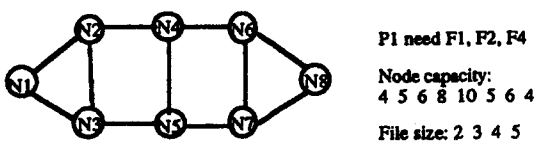


Figure 20. The system parameter for Example 4.

Table 7. The result for Example 3.

P1 in	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)	(1,8)
$p = 0.9$ reliability difference	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
$p = 0.8$ reliability difference	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
$p = 0.7$ reliability difference	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000	0.0000000
OPT REL							
$p = 0.9$	0.9989100	0.9989100	0.99989100	0.9999999	0.9999776	0.9999776	0.9998608
$p = 0.8$	0.9972122	0.9907200	0.99072000	0.9999999	0.9992397	0.9995397	0.9972122
$p = 0.7$	0.9836672	0.9673300	0.96733000	0.9941340	0.9941340	0.9941340	0.9836672s

The number of total pathsets: 10935
After HROFA: 3329

When we apply the HROFA algorithm, an exception will occur when the network topology is fully connected. This is because the superior reliability part (Case 2a contains the pruned node and all of the ancestor nodes) for Super pathsets becomes very small, and the reliability difference from Case 2b (containing the pruned node and at least one ancestor node not included) becomes significant for the many MFSTs in it. Even so, the deviation is no more than 0.25% (at a link reliability of 0.7).

5. CONCLUSION

Distributed Computing Systems (DCS) have become a major trend in today's computer system design for their high fault-tolerance, potential for parallel processing, and better reliability performance. One important characteristic of a DCS is that it offers redundant copies of software and/or hardware to improve the reliability of the system. One important problem in DCS design is the file assignment problem. This problem has been proved to be an NP-complete problem. Traditional solution techniques such as the back-tracking algorithm and mathematical programming can give the optimal solution, but they cannot effectively reduce the problem space. Sometimes, an application requires a fast way to compute reliability because of resource considerations. In this situation, deriving the optimal reliability may not be a wise idea. Instead, a fast method yielding near optimal reliability is preferable.

In this paper, we develop a heuristic algorithm (HROFA) for the reliability-oriented file assignment problem that uses a careful reduction method to reduce the problem space. Our numeral results show that the HROFA algorithm obtains the exact solution in most case and the computation time is significantly shorter than that needed for an exact method. When HROFA fails to give an exact solution, the deviation from the exact solution is very small.

REFERENCES

1. D.P. Agrawal, *Advanced Computer Architecture*, 376 pages, Computer Society of the IEEE, (1988).
2. T.C.K. Chou and J.A. Abraham, Load redistribution under failure in distributed systems, *IEEE Trans. Comput.* **32**, 799–808 (1983).
3. D.W. Davies *et al.*, Distributed systems architecture and implementation, *Lecture Notes in Computer Science*, p. 105, Springer-Verlag, Berlin, Germany, (1981).
4. P. Enslow, What is a distributed data processing system, *IEEE Computer* **11** (1978).

5. J. Garcia-Molina, Reliability issues for fully replicated distributed database, *IEEE Computer* **16**, 34–42 (1982).
6. V.K. Prasnna Kumar, S. Hariri and C.S. Raghavendra, Distributed program reliability analysis, *IEEE Trans. Software Eng.* **12** (1), 42–50 (1986).
7. A. Kumar, S. Rai and D.P. Agrawal, On computer communication network reliability under program execution constraints, *IEEE Journal on Selevted Areas in Communication* **6** (8), 1393–1399 (1988).
8. S. Hariri, C.S. Raghavendra and V.K. Kumar, Reliability analysis in distributed systems, In *IEEE '86 Distributed System Conf.*, pp. 564–571, (1986).
9. V.K. Kumar, S. Hariri and C.S. Raghavendra, Distributed program reliability analysis, *IEEE Trans. Software Engineering* **12**, 42–50 (1986).
10. C.S. Raghavendra, V.K. Kumar and S. Hariri, Reliability analysis in distributed systems, *IEEE Trans. Computer* **37**, 352–358 (1988).
11. W.W. Chu, L.J. Holloway, M.T. Lan and K. Efe, Task allocation in distributed data processing, *IEEE Computer Magazine*, 57–69 (1980).
12. V. Rajendra Prasad, Y.P. Aneja and K.P.K. Nair, A heuristic approach to optimal assignment of components to a parallel-series network, *IEEE Trans. Reliability* **40** (5) (1991).
13. C.V. Ramamoorthy, The isomorphism of simple file allocation, *IEEE Trans. Computer* **32** (1983).
14. W.W. Chu, Optimal file allocation in a multiple computer system, *IEEE Trans. Computer* **18** (10) (1969).
15. K.P. Eswaran, Placement of records in a file and file allocation in a computer network, In *Information Processing 74, IFIPS*, North-Holland, New York, (1974).
16. S.H. Bohai, Dual processor scheduling with dymanic reassignment, *IEEE Trans. on Software Enginnering* **5** (4), 341–349 (1979).
17. J. Akoka, Bounded branch and bound method for mixed integer non-linear programming, Sloune-School of Management, p. 77, MIT, (1977).
18. C.P. Wang, On the study of the file assignment in distributed system, NCTU Technical Report, (1990).
19. G. Hwang, A heuristic task assignment algorithm to maximize reliability of a distributed system, *IEEE Trans. on Reliability* **42** (3), 408–415 (1993).
20. S. Hariri and C.S. Raghavendra, SYREL: A symbolic reliability algorithm based on path and cut set methods, In *Proceeding of INFOCOM. '86*, pp. 293–302, (1986).
21. M.S. Lin and D.J. Chen, New reliability evaluation algorithms for distributed computing systems, *Journal of Information Science and Engineering* **8** (3) (1992).