Distributed Program Reliability Analysis
Min-Sheng Lin and Deng-Jyi Chen
Institute of Computer Science and Information Engineering
National Chiao-Tung University
Hsin Chu, Taiwan, R.O.C. 30050

Abstract

This paper presents an algorithm for computing the distributed program reliability in Distributed Computing Systems (DCS). The algorithm, called FREA (Fast Reliability Evaluation Algorithm), is based on the generalized factoring theorem with several incorporated reliability-preserving reductions to speedup the reliability evaluation. The effect of file distributions, program distributions, and various topologies on reliability of the DCS is studied in detail using the proposed algorithm. Compared with existing algorithms on various network topologies, file distributions, and program distributions, the proposed algorithm is much more economic in both time and space. To compute the Distributed Program Reliability, the ARPA network is studied to illustrate the feasibility of the proposed algorithm.

1 Introduction

Recently, Distributed Computing System (DCS) has become increasingly popular because it offers higher fault tolerance, potential for parallel processing, and better reliability in comparison with other processing systems [1-5]. A typical DCS consists of Processing Elements (PEs), memory units, data files and programs as its resources. These resources are interconnected via a communication network that dictates how information could flow between PEs. Programs residing on some PEs can run using data files at other PEs as well. For successful execution of a program, it is essential that the PE containing the program and other PEs that have the required data files, and communication links between them must be operational. In [6], a notion of Minimum File Spanning Tree (MFST) is proposed to represent the multiterminal connection required for executing a distributed program and a two-pass method for the reliability analysis of DCS is developed. In their method, all MFSTs are obtained by using graph traversal rather than applying path enumeration technique among the pairs of PEs. After finding out the MFSTs, for they are not disjoint with each other, the algorithm requires other reliability evaluation algorithms such as SYREL [12] to generate the reliability expression. Although the method is elegant, it does generate some replicated trees during the processing and thus will be inefficient. Instead of generating MFSTs, one algorithms, called FARE, has been proposed in [13-14] to compute DSR directly by using connection matrix. Based on the assumption that the PEs (nodes) in the DCS are perfect, it does not require additional reliability evaluation algorithms to convert multiterminal connection into reliability expression. The shortcoming of this algorithm is that they are not applicable for distributed programs running on more than one node. The proposed FREA algorithm employs a different concept to compute the reliability of DSR and DMR for node perfect case. It is based on the generalized factoring theorem with several incorporated reliability-preserving reductions to speedup the computation. The factoring theorem for exact computation of K-terminal reliability in undirected networks have existed since at least 1958, viz., Moskowitz [15]. Recently, several papers have addressed worst-case computational complexity and the optimality of classes of factoring algorithms and related algorithms, for example, Bail [16], Chang [17], Satyanarayana & Chang [18], and Wood [19] to name a few. However, these papers only address reliability problems or performance issues on various computer networks with are considered to be static-oriented problems. Distributed system reliability (DSR) and distributed program reliability (DPR), on the other hand, are more dynamic-oriented since the reliability is very sensitive to the way of data file distributions, program distributions, and network topologies. Naturally, this type of problem is considered to be more complex and difficult than computer network reliability problems.

2 Notations and Definitions

The notations and definitions used in the rest of the paper are summarized here.

\[ G=(V, E) \] an undirected graph in which the vertices (nodes) represent the PEs and the links (edges) represent the communication links.

\( x_i \) a node representing a processing element \( i \).

\( x_{ij} \) a link between processing elements \( i \) and \( j \).

\( G \) a graph with a node \( x_i \) called starting node, specified from which the FARE-NP algorithm begins to generate subgraphs.

\( p_i (q_j) \) probability that the node \( x_i \) works (fails).

\( p_{ij} (q_{ij}) \) probability that the link \( x_{ij} \) works (fails).

\( F_i \) the data file \( i \).

\( P_i \) the program \( i \).

\( P_A \) the set of programs can be run at processing element \( x_i \).

\( F \) the set of data files available at processing element \( x_i \).

\( F_A \) the set of data files needed to execute \( P_i \).

\( F_A \) the set of data files needed to execute all programs in \( P \) (i.e. \( F = \bigcup F_A \)).

\( F \) a spanning tree that connects the root node (the processing element that runs the program under consideration) to some other nodes such that its vertices hold all the needed files for the program under consideration.

\( MFST \) it is a FST such that there exists no other FST which is subset of it.

\( G \) the graph \( G \) with edge \( x_{ij} \) deleted.

\( G_B \) the graph \( G \) with edge \( x_{ij} \) contracted so that the endpoints are identified as a single node. This new node includes the data files and programs that the original endpoints have.

3 The Distributed Program Reliability Analysis

This research work is supported in part by the Chung-Shan Institute of Science and Technology under the contract # CS 80-0210-D-009-33 and in part by the National Science Council, Taiwan, R.O.C. under the contract # NSC 81-0408-E-009-18.
Considering the distributed processing system in Figure 1, there are four processing elements (x₁, x₂, x₃, x₄) connected by links x₁,₂, x₁,₃, x₂,₃, x₂,₄, and x₃,₄. Processing element x₁ contains two data files (F₁ and F₂) and can run P₁ directly from here to complete the execution of P₁. The detail information of each node is communicated with other nodes for accessing data files required to run P₁. Thus the distributed program reliability for a given program P₁ requires F₁, F₂, and P₁ to work properly. Let P₁ require F₁, F₂, and P₁. Two data files (F₁ and F₂) and can run P₁ directly from here to complete its execution. The detail information of each node is communicated with other nodes for accessing data files required to run P₁. Thus the distributed program reliability for a given program P₁ requires F₁, F₂, and P₁ to work properly.

**Fig. 1.** A simple distributed computing system

Let P₁ require F₁, F₂, and F₃ to complete its execution in the DCS and can be run on both node x₁ and x₂ (Figure 1). We can identify some File Spanning Trees (FSTs) rooted on x₁ from the DCS graph:

1. x₁ x₂ x₁,₂
2. x₁ x₂ x₁,₂ x₁,₂ x₂,₃
3. x₂ x₂ x₂,₃ x₂,₃ x₁,₂
4. x₁ x₂ x₁,₂ x₁,₂ x₂,₃
5. x₁ x₂ x₂ x₁,₂ x₂,₃ x₃,₄
6. x₁ x₂ x₂ x₃ x₂,₃ x₁,₂
7. x₁ x₂ x₂ x₂,₃ x₁,₂ x₂,₃
8. x₁ x₂ x₁,₂ x₁,₂ x₂,₃ x₃,₄
9. x₁ x₂ x₂ x₁,₂ x₂,₃ x₃,₄
10. x₁ x₂ x₁,₂ x₁,₂ x₂,₃ x₃,₄

Thus the distributed program reliability for a given program j can be defined as the probability of at least one MFST of program j is connected by links x₁,₂, x₁,₃, x₂,₃, x₂,₄, and x₃,₄. Processing element x₁ contains two data files (F₁ and F₂) and can run P₁ directly from here to complete its execution of P₁. The detail information of each node is summarized in P₁, P₂, and P₃ (i=1,2,4) in Figure 1.

**Fig. 2.** A simple distributed computing system with different file distribution.

Because the MFSTs generated by the algorithm in [6] are not disjoint with each other, other reliability computation programs such as SYREL [12] are required to generate the reliability expression. For node perfect case, one algorithm, called FARE, which can evaluate DPR in one pass is reported in [13]. Since a matrix is used to represent the subgraphs in the FARE algorithm, the reliability analysis methods can not be used to evaluate the reliability of a program running on more than one node.

**4. Derivation of The FREA Algorithm**

In this section, we present a new algorithm, called FREA (Fast Reliability Evaluation Algorithm), for the reliability evaluation of DCS. The FREA algorithm is based on the generalized factoring theorem employing several reliability-preserving reductions to reduce the computation trees. To illustrate our approach, we begin by presenting the concept of a generalized factoring theorem and then several reliability-preserving reductions.

**4.1 The Generalized Factoring Theorem for Distributed Program Reliability**

The factoring theorem of network reliability [18] is the basis for a class of algorithms for computing K-terminal reliability. This theorem establishes the validity of the following conditional reliability formula:

\[ R(G) = p_{ij} R(G \setminus x_{ij}) + q_{ij} R(x_{ij}) \]  

(4.1)

The theorem can be used to interpret topologically the following conditional reliability formula for a general binary system S with component x₁,₂:

\[ R(S) = p_{ij} R(S \setminus x_{ij}, \text{ works}) + q_{ij} R(S \setminus x_{ij}, \text{ fails}) \]  

(4.2)

Thus, equation (4.1) can be generalized in the following manner. Suppose that node x₃ is the starting node of graph G₂, and x₁,₂, x₂,₃, ..., x₂,₃,k are the edges incident on x₃. We can obtain the following generalized equation:

\[ R(G₂) = p_{ij} R(G \setminus x_{ij}, \text{ works}) + q_{ij} R(x_{ij}) \]  

(4.3)

**Theorem 1:** Equation (4.3) is correct.

**Proof:** Let events E₁ be the event of x₁,₂ works, E₂ be the event of x₁,₂ fail, E₃ be the event of x₂,₃ works, E₄ be the event of x₂,₃ fail, ..., Eₖ be the event of x₂,₃,k works, and Eₖ⁺ be the event of x₂,₃,k fail.
Then, $E_1, E_2, \ldots, E_{k+1}$ are mutually exclusive events such that

$$k+1 \quad S = \bigcup_{i=1}^{k+1} E_i$$

In other words, exactly only one of the events $E_1, E_2, \ldots, E_{k+1}$ can occur. By writing

$$k+1 \quad S = \bigcup_{i=1}^{k+1} SE_i$$

and using the fact that the events $SE_{i+1}, i = 1, \ldots, k+1$, are mutually exclusive, we obtain that

$$k+1 \quad \Pr(S) = \bigcup_{i=1}^{k+1} \Pr(SE_i) = \sum_{i=1}^{k+1} \Pr(\text{Event } E_i) \Pr(\text{Event } E_i)$$

Since the link states are independent and the failure of one link does not affect the probability of other links, so $Pr(E_i) = Pr(x_i,2) \times Pr(x_2,2)$ for $i = 1, 2, \ldots, k$. By equation (4.4) we obtain

$$Pr(S) = \prod_{i=1}^{k+1} \Pr(x_i,2) \times \Pr(x_2,2)$$

Replacing $x_i,2$ by $x_i,k$ in equation (4.5) by $G_2$ and rewrite terms in equation (4.5) as that in equation (4.1), we get

$$Pr(G_2) = \prod_{i=1}^{k+1} \Pr(x_i,k) \times \Pr(x_2,k)$$

Thus equation (4.5) is correct.

4.2 Reliability-preserving Reductions for the DCS Reliability Evaluation

In order to reduce the size of graph $G$ and therefore reduce the state space of the associated reliability problem, reliability-preserving reductions can be applied. Some reductions are designed and developed to speed up the reliability evaluation.

**Def: Degree-1 Reduction**

Degree-1 reduction is removing nodes and their incident edges which contain no needed data files and programs under consideration. Considering the DCS in figure 4 for computing DPR, since node $x_1$ does not contain $P_1$ and any needed data files and programs under consideration, the degree-1 reduction is applied to remove node $x_1$ and its incident edge $x_1,j$. The resulting graph is shown in figure 4. To prove degree-1 reduction is correct is trivial, thus it is omitted here.

**Def: Irrelevant Component Deletion**

Let $G_0 = (V_0, E_0)$ be a connected component of $G$, and it is not connected to the rest of components of $G$. If there are no FSTs in $G_0$ then the component $G_0$ is irrelevant and a reduction is applied to delete the component $G_0$. For the example in figure 5, to compute the DPR, one needs only data files $F_1$, $F_2$, and $F_3$. Since $G_0$ does not contain data files $F_2$ and $F_3$, there are no FSTs in it. Thus, it can be deleted from the graph without affecting the reliability evaluation. To prove irrelevant component deletion reduction is correct is trivial, thus it is omitted here.

**Def: Parallel Reduction**

Let $x_i,j$ and $x_i,k$ be two parallel edges in $G$. Then, $G'$ is obtained by replacing $x_i,j$ and $x_i,k$ with a single edge $x_i,j'$. Thus, equation (4.3) is correct. Q.E.D.

**Def: Series Reduction**

There are some differences in series reduction between the DCS reliability problem and the $K$-terminal network reliability problem. To prove series reduction for the $K$-terminal network reliability problem is the same as that in equation (4.1), we get

$$Pr(G') = \prod_{i=1}^{k+1} \Pr(x_i,k) \times \Pr(x,2,k)$$

Thus equation (4.3) is correct.

**Def: Degree-2 Reduction**

Degree-2 reduction is removing nodes and their incident edges which contain no needed data files and programs under consideration. Considering the DCS in figure 4 for computing DPR, since node $x_1$ does not contain $P_1$ and any needed data files and programs under consideration, the degree-2 reduction is applied to remove node $x_1$ and its incident edge $x_1,j$. The resulting graph is shown in figure 4. To prove degree-2 reduction is correct is trivial, thus it is omitted here.

**Def: Reducible Node**

A node $x_i$ is called a reducible node for distributed program $P_j$ in graph $G$ if and only if: (1) the degree of node $x_i$ is two in graph $G$, and (2) the degree of node $x_i$ in the MSTs of $P_j$ that contains node $x_i$ must be two.
Theorem 2: Node $x_i$ is a reducible node for distributed programs $P_g$ if it satisfies

1) node degree is two, and
2) $PA \cap FN = \emptyset$ and $PA \cap FN = \emptyset$

where node $x_i$ and $x_j$ are the two adjacent nodes of $x_i$.

Proof:
Case 1: Some Minimal File Spanning Tree $MFST_1$ generated for $DPR_g$ contains node $x_i$. Suppose $x_i$ satisfies the properties of theorem 2 and $x_i$ is not a reducible node, then it implies either 1) $x_i$'s node degree is not two, or 2) $x_i$'s node degree in the $MFST_1$ is not two according to the definition of reducible node. In 1), $x_i$'s node degree is not two is violated properties of theorem 2 that degree of node $x_i$ is two (since we assume $x_i$ satisfies the properties of theorem 2). Thus, it must be the case of 2), i.e., the $x_i$'s node degree in the $MFST_1$ is two. Since the first given property in theorem 2 states that the degree of node $x_i$ is two, the $MFST_1$ that contains node $x_i$ can only have the degree of node $x_i$ less than or equal to two. Furthermore, in 2), we assume that the degree of node $x_i$ in the $MFST_1$ is two, then it must be one. This implies that node $x_i$ is a leaf node in the $MFST_1$. Based on the second given property in theorem 2, it implies that node $x_i$ contains a subset of needed data files in node $x_j$ or $x_k$ and a subset of programs to be executed in node $x_i$. From these facts, we conclude that $x_i$ is the one of the nodes in $MFST_1$ incorrect. In other word, $MFST_1$ is not a Minimal File Spanning Tree. Thus, the assumption that node $x_i$ is not a reducible node is not true. Therefore, node $x_i$ must be a reducible node.

Case 2: No $MFST_1$s contain node $x_i$. Theorem 2 is obviously true for this case. Q.E.D.

Corollary 1: If a node $x_i$ satisfies the following properties

1) the degree is two, and
2) $PA \cap FN = \emptyset$ and $PA \cap FN = \emptyset$

then node $x_i$ is a reducible node.

Def: Degree-2 Reduction
Suppose node $x_i$ is a reducible node, then one can apply series reduction on node $x_i$ and move data files and programs within node $x_i$ to one of its adjacent nodes $x_j$ or $x_k$. This reduction case is called degree-2 reduction. Figure 9 presents an example of such reduction.

Theorem 3: Degree-2 reduction is correct for DPR analysis.
Proof: Let $G$ be the original graph while $G'$ be the one after degree-2 reduction. We need to show the computation of DPR from $G'$ is the same as that from $G$. To compute DPR from $G'$, we need to show that all the reliability properties due to the changes of data file distribution, program distribution, and topologies are all preserved from $G$. Since $G'$ is a graph from $G$ by applying degree-2 reduction, we know that two edges $(x_{ij}$ and $x_{ik})$ incident on $x_i$ must be working simultaneously for those $MFST_1$s that contain node $x_i$ since node $x_i$ is a reducible node. Thus, we replace edges $x_{ij}$ and $x_{ik}$ with a single edge $x_{jk}$ such that $p_{jik} = p_{ijk} = p_{jki}$ is used during the reliability evaluation. In this way we preserve its topological change during the reliability analysis. Furthermore, we have moved data files and programs within node $x_i$ to node $x_j$ or $x_k$. This will preserve both the data files and programs distribution during the reliability analysis. Therefore, all the reliability properties within $G'$ after degree-2 reduction are preserved in graph $G'$. This implies that the computation of DPR from $G'$ is the same as that from $G$. Q.E.D.

By theorem 3, the series reduction is just a special case of degree-2 reduction that meets the properties of corollary 1.

4.3 The Identification of Reducible Nodes
In this section, we propose an algorithm to identify all reducible nodes in a DCS graph.

Proof of Theorem 3:

1. If $G=GI$, $x_i$ is a reducible node.
2. If $G=GI$, $x_i$ is a reducible node.
3. If $G=GI$, $x_i$ is a reducible node.
4. If $G=GI$, $x_i$ is a reducible node.
5. If $G=GI$, $x_i$ is a reducible node.
6. If $G=GI$, $x_i$ is a reducible node.

Fig. 10. An example of DCS and all MFSTs for program 1 under consideration.

Let us consider the DCS shown in figure 10. Although $x_1$ and $x_2$ are reducible nodes by the definition of reducible node, only $x_1$ can be identified based on the reliability analysis. Thus, the problem is how to find all the reducible nodes in the DCS graph. The most straightforward solution is to find all the MFSTs, and then validate the nodes of those MFSTs that contains the reducible nodes. However, such a solution inherits the problem in Kumar 86 [6] which will generate several replicated trees and therefore is not a good approach.

In the following, we present a new algorithm, called REDUCIBLE_NODE, to identify all the reducible nodes without the generation of all MFSTs. The basic concept of the algorithm can be explained from the following statements.

Step 1: $G=GI$. $x_1$ is a reducible node.
Step 2: delete all nodes in $GI$ that contain data file $Fa$ except node $x_1$.
Step 3: check if there are some $FSTs$ in the component of $GI$ that contains $x_1$.
Step 4: $GI=GI-x_1$.
Step 5: the same as step 2.
Step 6: the same as step 3.1.

We repeat the above steps to check the other needed data files and programs under consideration that are also in $x_1$. If the checking procedure can not identify $x_1$ as a reducible node (step 3.1 or step
REDUCIBLE-NODE algorithm is given below.

**REDUCIBLE-NODE** (G)

begin
  for all node \( x \in G \) do
    if degree\((x)\) = 2 then
      /* assume that the two edges incident on node \( x \) are
      \( x_i \) and \( x_j \) */
      \( G' = G - x_i \) /* delete \( x_i \) from \( G \) */
      for all \( f \in (P_{A_1} \cap FN) \) and all
      program \( p \in (PA_1 \cap PN) \) do
        delete all nodes in \( G \) that contain file
        \( f \) or program \( p \) from \( G \) except node \( x \).
      end
      \( G' = \) the component that contains node \( x \) in \( G' \)
      if there are some FSTs in \( G' \) then
        go to check_next_node
      end
      /* the \( x \) is a reducible node, apply degree-2 reduction */
      \( G = G - x_i - x_j \) /* delete \( x_i \) and \( x_j \) from \( G \) */
      \( P_j = P_j \cap P_k \)
      \( \forall \) \( i \) such that \( \forall p \in PA_i \) do
        \( FA = FA \cup PA_i \) in \( G \)
      end
      \( G = G - x_i \) /* delete \( x_i \) from \( G \) */
      if there are some FSTs in \( G \) then
        go to check_next_node
      end
    end
  end

end /* REDUCIBLE-NODE */

4.4 The FREA Algorithm

Once the way of finding all the reducible nodes is understood, we can use equation (4.3) and the reliability-preserving reductions discussed in section 4.2 to compute the DPR and DSR. The complete FREA algorithm is listed below.

**FREA Algorithm**

begin
  \( G = \) the original DCS graph
  \( FN = \cup FN_j \)
  \( P_j \in PN \)
  /* all the needed data files for program \( P_j \) in \( PN \) */
  \( R = 0 \) /* the reliability set to 0 */
  search a node \( x \) that contains program \( P_j \) in \( PN \)
  if node \( x \) is not found then
    begin
      output \((R)\)
      step
    end
    \( s = i \) /* starting node's number */
    \( R = REL(G_s) \)
    output \((R)\)
  step
end /* FREA */

function REL(G)
begin
  Step 1: The checking step
  if \( FA_1 \supseteq FN \) and \( PA_1 \supseteq PN \) then
    begin
      REL = 1
      return
    end
  end

If there are no FSTs in \( G_s \) then
/* using DFS algorithm to check this */
begin
  REL = 0 /* no FSTs in \( G_s \) */
  return
end
Step 2: The reduction step for \( G_s \)
repeat
  Perform degree-1 reduction
  Perform series reduction
  Perform parallel reduction
  Perform degree-2 reduction
  /* using REDUCIBLE-NODE algorithm */
end
Until no reductions can be made
Step 3: The formulating step for equation (4.3)
3.1: \( G'_s \) = the new graph after the above reduction
3.2: \( G_s \) and \( G'_s \) are temporary variables for graph \( G_s \) */
3.3: \( R = R + C \cdot REL(G'_s \oplus x_s) \)
3.4: \( G_s = G'_s - x_s \)
3.5: \( G''_s \) = the new graph after deleting
    irrelevant components from \( G''_s \)
if \( x_s \) is deleted then
  go to step 4
end
Step 4: The choosing step to find the new staring
node
if finding a node \( x_k \) in \( G''_s \) that contains the
programs under consideration then
begin
  \( s = k \)
  \( R = R + C \cdot REL(G''_s) \)
end
REL = \( R \)
end /* REL */

4.5 Numeric Examples

The reliability analysis process of the FREA algorithm can be represented by a trace tree. A trace tree depicts the relationship among intermediate trees or subgraphs generated using the reduction concepts incorporated in the FREA algorithm. A trace tree node consists of four components, \( G, G', G'', \) and \( G''' \) as shown in figure 11, which represent the intermediate trees or subgraphs from the reduction process.

![Fig. 11. The basic node structure of trace tree](image)

The relationship of trees within a trace tree node, using notation defined in the FREA algorithm, can be explained by the following example.

Considering the trace tree in figure 12, suppose intermediate tree \( G_0'' \) in the trace node \( N_0 \) have starting node \( x_k \) with \( k \) incident edges then the maximal number of trace tree nodes that trace tree node \( N_0 \)
can derive is k+l (refer to equation (4.3)). Since only k+l terms (intermediate subgraphs) can be generated, components Gk+l" and Gk+l‴ within the trace tree node Nk+l are deleted. The operations to be applied from G' in trace tree node Nk+l are...operations to be applied from G' in trace tree node Nk+l to trace tree node Nk+l. The operations available for Sj can be deleting, merging, or combinations of merging and deleting. For example, Sj=x1,1 x1,2 means that edge x1,1 in component G0' is deleted and then G0' is merged with edge x1,2 to produce a new intermediate subgraph Gj within trace tree node Nk+l. The symbol → indicates which intermediate subgraph is generated by which intermediate subgraph. For example, G1 in trace tree node Nk+l is obtained from the G0' within trace tree node Nk+l by applying operation Sj (written as G1 = G0'@x1,1 using the notation defined in the FREA algorithm). The rest of the relations are listed below.

G1=G0'@x1,1
G1'=the reduction graph of G1
G1"=the reduction graph of G1'
G1‴=the reduction graph of G1"
G1‴=the reduction graph of G1‴

If the starting node x1 in component G within trace tree node Nk+l holds all data files required and programs to be executed then Nk+l is a leaf node of the trace tree. Figure 13 depicts the trace tree for program 4 under consideration in figure 1, and the DPR4 can be computed as

DPR4 = p1 p2 p3 p4 + q1 p2 p3 p4 + p1 q2 p3 p4 + q1 q2 p3 p4

where pi is the probability of link i, and qi=1-pi)

Let the probability of any link being operational be 0.9, then DPR4 is computed to 0.9766640.

5 Algorithm Comparison

In this section, comparisons among the algorithms proposed in this paper and existing algorithms [6,13,20] are given. The algorithms presented in [6,13,20], in the worst case, can generate as many as (n-1)(e-1) intermediate trees (or subgraphs) where n denotes the number of nodes and e is the maximum in-degree of a node in the graph. However, in practical conditions, it may not occur since once a MFST is found the tree expansion is stopped. Unlike the computer network reliability problems which are static-oriented, the distributed program reliability problems in DCS are dynamic-oriented since many factors (file distribution, program distribution, topology) can greatly affect the efficiency of the algorithm. Thus, it is very difficult to quantify exactly the time complexity. The FREA algorithm employs several reduction concepts which effectively speed up the whole reliability evaluation. A more appropriate and rational comparison for these different algorithms can be made based on the

Fig. 12 The trace tree structure

Fig. 13 The trace tree of FREA for the example of figure 1

Fig. 14 is another example of DCS and figure 15 is the trace tree for program 4 under consideration in figure 14, and the DPR4 can be computed as

DPR4 = p1 p2 p3 p4 + q1 p2 p3 p4 + p1 q2 p3 p4 + q1 q2 p3 p4 + p1 q2 p3 p4 + q1 q2 p3 p4 + p1 q2 p3 p4 + q1 q2 p3 p4

where pi is the probability of link i, and qi=1-pi)

Let the probability of any link being operational be 0.9, then DPR4 is computed to 0.9766640.
under the effects of different sizes of DCS, data file distributions, program distribution, and topologies. The following sections focus on these different comparisons.

5.1 The Effect of Different Sizes on the Performance of Different Algorithms

Figure 18 is a well-known example of a computer communication network- the ARPA computer network in which there are 21 nodes and 26 links. Suppose that there are 12 data files and 10 programs distributed in the ARPA computer network, and the file distribution, program distribution, and files needed for a program to be executed are given in tables 1, 2, and 3 respectively. The number of subgraphs generated for different programs under consideration are given in table 4.

Table 1. File distributions Table 2. Program distributions Table 3. Data files needed for execution a program $P_i$

Table 4. The number of subgraphs generated and the DPR for the example of ARPA computer network

It is clear that as the size of the DCS increases the number of intermediate subgraphs generated to compute the reliability increases for all algorithms. Also, the number of intermediate subgraphs generated by the FREA algorithm is thousands of times less than that of the existing algorithms in a large and complex distributed network such as the ARPA network.

6 Conclusion

Distributed Computing System (DCS) has become very popular for its high fault-tolerance, potential for parallel processing, and better reliability performance. One of the important issues in the design of the DCS is the reliability performance. Traditional reliability indexes such as source-to-terminal, survivability, multi-terminal reliability, K-terminal reliability and so on are not directly applicable for the analysis of the distributed reliability property in DCS. Thus, new approaches and algorithms for the distributed program reliability analysis of the DCS must be developed. In this paper we propose an algorithm, called FREA which is based on the generalized factoring theorem with several incorporated reliability-preserving reductions, to speed up the reliability evaluation process. These reliability-preserving reductions are the major contributions on speeding up the reliability evaluation process. Compared with existing algorithms on various network topologies, file distributions, and program distributions, the FREA algorithm is much more economic in both time and space. The feasibility of the proposed algorithm for DPR and DSR analysis can easily be confirmed through analysis on the ARPA computer network.

References