# 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(PE), 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 DPR 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 classed of factoring algorithms and related algorithms, for example, Ball[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

Notations and definitions used in the rest of 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 $i$. |
| $x_{i,j}$ | a link between processing elements $i$ and $j$. |
| $G_s$ | $G$ with a node $x_s$, called starting node, specified from which the FARE-NP algorithm begins to generate subgraphs. |
| $p_i(q_i)$ | probability that the node $x_i$ works (fails). |
| $p_{i,j}(q_{i,j})$ | probability that the link $x_{i,j}$ works (fails). |
| Fi | the data file $i$ |
| Pi | the program $i$ |
| $PA_i$ | the set of programs can be run at processing element $x_i$ |
| $FA_i$ | the set of data files available at processing element $x_i$. |
| $FN_i$ | the set of data files needed to execute Pi |
| $PN$ | the set of programs under consideration |
| $FN$ | the set of data files needed to execute all programs in $PN$ (i.e. $FN = \bigcup_{Pi \in PN} FN_i$) |
| FST | 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. |
| $G\text{-}x_{i,j}$ | the graph $G$ with edge $x_{i,j}$ deleted |
| $G \oplus x_{i,j}$ | the graph $G$ with edge $x_{i,j}$ 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. |
| R($G$) | the reliability of the DCS graph $G$ |

Since tree and subgraph are used to represented the underlying communication structure of the DCS, the terms *tree* and *subgraph* are used interchangeable in the rest part of this paper.

## 3 The Distributed Program Reliability Analysis

This research work is supported in part by the Chung-Shan Insitute 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.



0-8186-2755-7/92 $03.00 © 1992 IEEE

Considering the distributed processing system in figure 1, there are four processing elements $(x_1, x_2, x_3, x_4)$ connected by links $x_{1,2}, x_{1,3}, x_{2,3}, x_{2,4}$ and $x_{3,4}$. Processing element $x_1$ contains two data files (F1 and F2) and can run P1 directly from here to communicate with other nodes for accessing data files required to complete the execution of P1. The detail information of each nodes is summarized in $FA_j$, $PA_j$, and $FN_j$ ($j$=1...4) in figure 1.

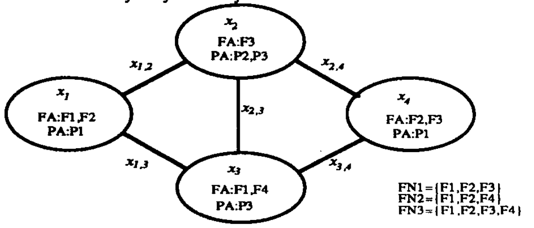


Fig. 1. A simple distributed computing system

Let P1 require F1, F2, and F3 to complete its execution in the DCS and can be run on both node $x_1$ and $x_4$ (figure 1). We can identify some File Spanning Trees (FSTs) rooted on $x_1$ from the DCS graph: 1) $x_1\, x_2\, x_{1,2}$, 2) $x_1\, x_2\, x_3\, x_{1,2}\, x_{2,3}$, 3) $x_1\, x_2\, x_4\, x_{1,2}\, x_{2,4}$, 4) $x_1\, x_2\, x_3\, x_{1,3}\, x_{2,3}$, 5) $x_1\, x_3\, x_4\, x_{1,3}\, x_{3,4}$, 6) $x_1\, x_2\, x_3\, x_4\, x_{1,2}\, x_{2,3}\, x_{3,4}$, 7) $x_1\, x_2\, x_3\, x_4\, x_{1,2}\, x_{2,4}\, x_{3,4}$, 8) $x_1\, x_2\, x_3\, x_4\, x_{1,3}\, x_{2,3}\, x_{2,4}$, *and* 9) $x_1\, x_2\, x_3\, x_4\, x_{1,3}\, x_{3,4}\, x_{2,4}$

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 working [6]. This can be written as

$$DPR_j = \text{Prob}\left( \bigcup_{k=1}^{n_{mfst}} MFST_k \right) \quad (3.1)$$

where $n_{mfst}$ is the number of MFST that run the given program.

For computing the reliability of the entire DCS, the concept of MFST has been extended to Minimal File Spanning Forest (MFSF)[14]. Then the reliability of a DCS can be given as

$$DSR = \text{Prob}\left( \bigcup_{i=1}^{n_{mfsf}} MFSF_i \right) \quad (3.2)$$

where $n_{mfsf}$ is the number of MFSF that run all programs.

Based on the concepts of the MFST and MFSF, Kumar and his colleagues developed algorithms to generate all MFSTs [6] and MFSFs[20] respectively. Once the MFSTs and MFSFs are obtained, SYREL[12] are called for evaluating the reliability. Although the concept of their algorithm is very straightforward, it generates many replicated trees during the MFST generating process. Considering the DCS in figure 2, we like to find all the MFSTs for P1. As we can see in figure 3, the replicated trees (e.g. tree B , d2 , and d4 are replicated) have been generated by their algorithm. Thus a procedure, called CLEAN, is required to remove these replicated trees.

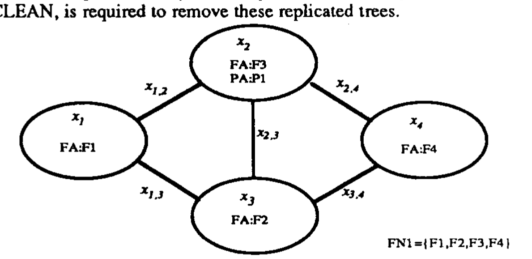


Fig. 2. A simple distributed computing system with different file distribution.

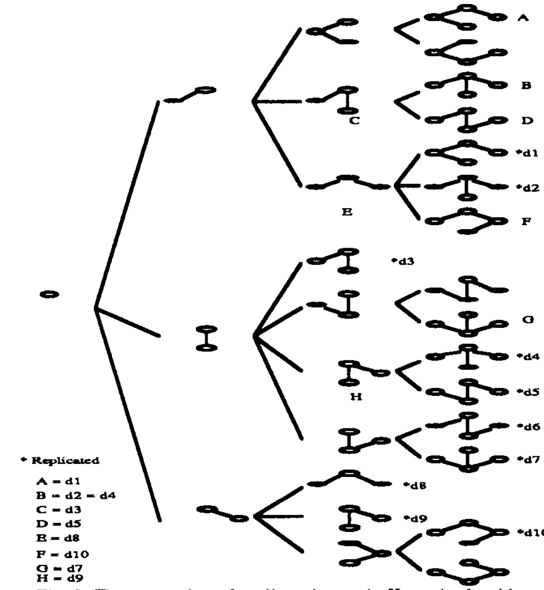


Fig. 3. The generation of replicated trees in Kumar's algorithm.

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 The 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 :

$$R(G) = p_{i,j}\, R(G \oplus x_{i,j}) + q_{i,j}\, R(G - x_{i,j}) \qquad (4.1)$$

The theorem can be used to interpret topologically the following conditional reliability formula for a general binary system $S$ with components $x_{i,j}$ :

$$R(S) = p_{i,j}\, R(S \mid x_{i,j} \text{ works}) + q_{i,j}\, R(S \mid x_{i,j} \text{ fails}) \qquad (4.2)$$

Thus, equation (4.1) can be generalized in the following manner. Suppose that node $x_s$ is the starting node of graph $G_s$, and $x_{s,1}$, $x_{s,2}$, ...., and $x_{s,k}$ are the edges incident on $x_s$. We can obtain the following generalized equation.

$$R(G_s) = p_{s,1} R(G \oplus x_{s,1}) + q_{s,1} p_{s,2} R(G - x_{s,1} \oplus x_{s,2}) + \dots + q_{s,1} q_{s,2} \cdots q_{s,k-1}\, p_{s,k} R(G - x_{s,1} - x_{s,2} - \dots - x_{s,k-1} \oplus x_{s,k}) + q_{s,1} q_{s,2} \cdots q_{s,k} R(G - x_{s,1} - x_{s,2} - \dots - x_{s,k}) \quad (4.3)$$

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

*Proof* : Let events

$E_1$ be the event of $x_{s,1}$ works,

$E_i$ be the event of $x_{s,1}$,...,and $x_{s,i-1}$ fail; $x_{s,i}$ works. ( for $i$=2, 3,...,$k$ ),

$E_{k+1}$ be the event of $x_{s,1}, x_{s,2}$, ....,and $x_{s,k}$ fail.

Then, $E_1, E_2, ...., $ and $E_{k+1}$ are mutually exclusive events such that

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

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

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

and using the fact that the events $SE_i$, $i = 1, ...., k+1$, are mutually exclusive, we obtain that

$$\Pr(S) = \sum_{i=1}^{k+1} \Pr(SE_i) = \sum_{i=1}^{k+1} \Pr(S/E_i)\Pr(E_i) \quad (4.4)$$

Since the link states are s-independent and the failure of one link does not affect the probability of other links, so $\Pr(E_i) = \Pr(x_{s,1}$ fails$)*\Pr(x_{s,2}$ fails$) .... * \Pr(x_{s,i-1}$ fails$)*\Pr(x_{s,i}$ works$) = q_{s,1}\, q_{s,2} \cdots q_{s,i-1}\, p_{s,i}$, for $i=1, 2, ..., k$; and $\Pr(E_{k+1}) = q_{s,1}\, q_{s,2} \cdots q_{s,k}$. By equation (4.4) we obtain

$\Pr(S) = p_{s,1}\Pr(S|x_{s,1}$ works$) + q_{s,1}p_{s,2}\Pr(S|x_{s,1}$ fails; $x_{s,2}$ works$) + .... + q_{s,1}q_{s,2} \cdots q_{s,k-1}p_{s,k}\Pr(S|x_{s,1}, x_{s,2}, ....,$ and $x_{s,k-1}$ fail; $x_{s,k}$ works$) + q_{s,1}q_{s,2} \cdots q_{s,k}\Pr(S|x_{s,1}, x_{s,2}, ....,$ and $x_{s,k}$ fail$)$. 4.5)

Replacing $S$ in equation (4.5) by $G_s$ and rewrite terms in equation (4.5) as that in equation (4.1), we get

$$R(G_s) = p_{s,1}R(G \oplus x_{s,1}) + q_{s,1}p_{s,2}R(G - x_{s,1} \oplus x_{s,2}) + .... + q_{s,1}q_{s,2} \cdots q_{s,k-1}p_{s,k}R(G - x_{s,1} - x_{s,2} - .... - x_{s,k-1} \oplus x_{s,k}) + q_{s,1}q_{s,2} \cdots q_{s,k}R(G - x_{s,1} - x_{s,2} - .... - x_{s,k})$$

Thus equation (4.3) is correct. **Q.E.D.**

Equation (4.3) can be recursively applied to the induced graph until either 1) the further induced graph with node $x_s$ containing all needed data files and all programs to be executed, or 2) the further induced graph with no FSTs is obtained. The induced graph of the former case represents a success while the latter case represents a failure.

### 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 $DPR_1$, since node $x_1$ does not contain $P_1$ and any needed data files ($F_1, F_2$, and $F_3$), the degree-1 reduction is applied to remove node $x_1$ and its incident edge $x_{1,3}$. The resulting graph is also shown in figure 4. To prove degree-1 reduction is correct is trivial, thus it is omitted here.

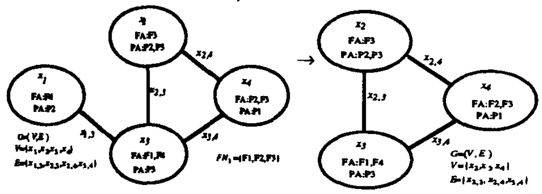

Fig. 4 The example of degree-1 reduction

**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 $DPR_1$, 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 effecting the reliability evaluation. To prove irrelevant component deletion reduction is correct is trivial, thus it is omitted here.

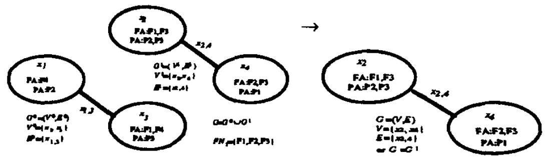

Fig. 5 The example of irrelevant component deletio

**Def: Parallel Reduction**

Let $x_{i,j}$ and $x_{i,j}'$ be two parallel edges in $G$. Then, $G'$ is obtained by replacing $x_{i,j}$ and $x_{i,j}'$ with a single edge $x_{i,j}''$ such that $p_{i,j}''=1-q_{i,j}*q_{i,j}'$ ( or $p_{i,j}''=p_{i,j}+p_{i,j}'-p_{i,j}*p_{i,j}'$ ). The parallel reduction for DPR and DSR problems is the same as the parallel reduction for $K$-terminal network reliability problem. To prove parallel reduction is correct is trivial, thus it is omitted here.

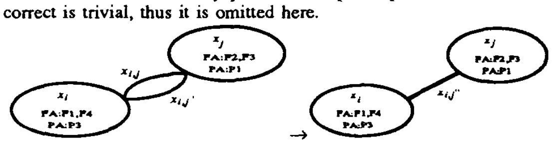

Fig. 6 The example of parallel reduction

**Def: Series Reduction**

There are some differences in series reduction between the DCS reliability problem and the $K$-terminal network reliability problem. The series reduction for the $K$-terminal network reliability problem is defined in [19] and is recalled here.

*Let $x_{i,j}$ and $x_{i,k}$ be two series edges in G such that degree($x_i$)=2 and $x_i \notin K$. Then, G' is obtained by replacing $x_{i,j}$ and $x_{i,k}$ with a single edge $x_{j,k}$ such that $p_{j,k}=p_{i,k}*p_{i,j}$*

The series reduction for DCS reliability problem is the same as above description except that the condition of $x_i \notin K$ is replaced by $FA_i \cap FN = \varnothing$ and $PA_i \cap PN = \varnothing$. In other words, if degree($x_i$)=2 and node $x_i$ contain no needed data files and programs to be executed, then we apply the series reduction on $G$. For example, figure 7 presents a case of series reduction for computing $DPR_1$. To prove series reduction is correct is trivial, thus it is omitted here.

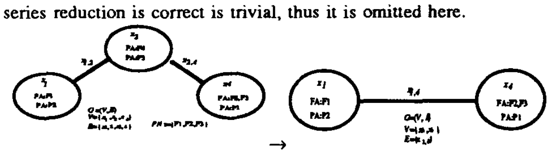

Fig. 7 The example of series reduction

For the case of degree($x_i$)=2 and node $x_i$ contains some needed data files or programs to be executed, the series reduction may be performed. The details of this case will be described latter in the degree-2 reduction.

**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 MFSTs of $P_j$ that contains node $x_i$ must also be two.

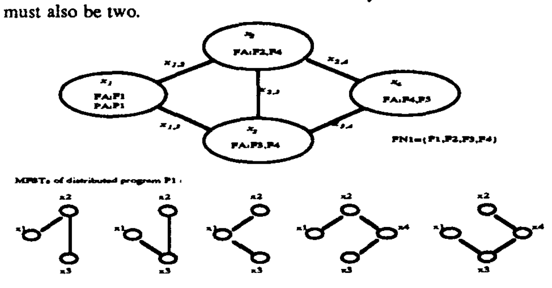

Fig. 8 An example of DCS and all MFSTs for program 1 under consideration

Figure 8 presents an example of the DCS and all the MFSTs generated for $DPR_1$ analysis. By the definition of the reducible node, node $x_4$ is a reducible node while node $x_1$, $x_2$, and $x_3$ are not.

***Theorem 2*** : Node $x_i$ is a reducible node for distributed program $P_g$ if it satisfies

1) node degree is two, and

2) $FA_j \supseteq (FA_i \cap FN)$ & $PA_j \supseteq (PA_i \cap PN)$ & $FA_k \supseteq (FA_i \cap FN)$ & $PA_k \supseteq (PA_i \cap PN)$ ( where node $x_k$ and $x_j$ are the two adjacent nodes of $x_i$ )

*Proof :*

Case 1: Some Minimal File Spanning Tree $MFST_t$ 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 $MFST_t$ is not two according to the definition of reducible node. In 1), $x_i$ 's node degree is not two is violated the first given property in theorem 2 that declares 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 $MFST_t$ is not two. Since the first given property in theorem 2 states that the degree of node $x_i$ is two, the $MFST_t$ that contain 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 $MFST_t$ is not two, then it must be one. This implies that node $x_i$ is a leaf node in the $MFST_t$. 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 $x_j$ or $x_k$. From these facts, we conclude that $x_i$ is one of the nodes in $MFST_t$ is incorrect. In other word, $MFST_t$ 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 MFSTs contains node $x_i$.

Theorem 2 is obviously true for this case. **Q.E.D.**

Using theorem 2, it is easy to verify the following corollary.

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

1) the degree is two, and

2) $FA_i \cap FN = \emptyset$ and $PA_i \cap PN = \emptyset$

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.

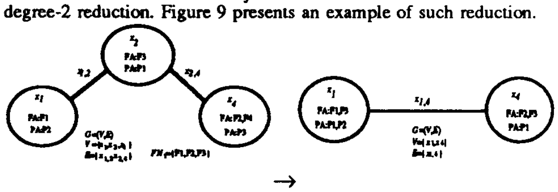

Fig. 9 The example of degree-2 reduction

**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 ( $x_{i,j}$ and $x_{i,k}$ ) incident on $x_i$ must be working simultaneously for those MFSTs that contain node $x_i$ since node $x_i$ is a reducible node. Thus, we replace edges $x_{i,j}$ and $x_{i,k}$ with a single edge $x_{j,k}'$ such that $p_{j,k}'=p_{i,j}*p_{i,k}$ 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 $x_k$ or $x_j$. 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.

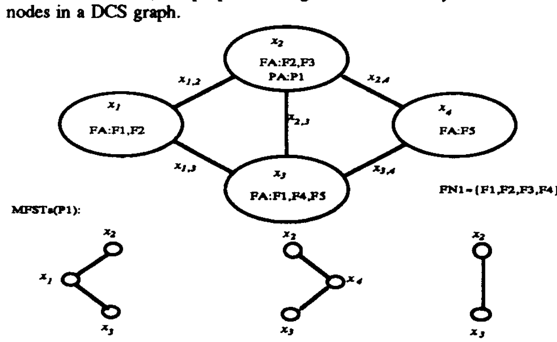


Fig. 10 An example of DCS and all MFSTs for program 1 under consideration

Let us consider the DCS shown in figure 10 . Although $x_1$ and $x_4$ are reducible nodes by the definition of the reducible node, only $x_4$ can be identified based on the corollary 1. 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 to 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.

Let $G$ be the original graph that contains node $x_i$ with node degree=2. Edges $x_{i,j}$ and $x_{i,k}$ be the two incident edges on $x_i$. Suppose node $x_i$ is not a reducible node, then it must be a leaf node of some $MFST_t$ ( also discussed in the proof of theorem 2 ). Thus, node $x_i$ must contain some needed data files or programs to be executed which are not resident at other nodes in the same $MFST_t$.

To test which data file causes the node $x_i$ that becomes a leaf node of the $MFST_t$, we can repeatedly check each needed data files, $F_a$, in node $x_i$ . The following procedures are used to check if needed data file $F_a$ in node $x_i$ is the one that causes $x_i$ not to be a reducible node.

Step 1: $G1=G-x_{i,j}$ /* $G1$ is $G$ with deleting edge $x_{i,j}$ */

Step 2: delete all nodes in G1 that contain data file $Fa$ except node $x_i$.

/* $x_i$ is the only node that contains data file $F_a$ in $G1$ */

Step 3: check if there are some FSTs in the component of $G1$ that contains $x_i$.

/* using the Depth-First-Search algorithm */

3.1: If there are some FSTs in this component then $x_i$ must be a leaf node of some MFSTs. Thus, $x_i$ is not a reducible node. Stop checking node $x_i$.

Step 4: $G1=G-x_{i,k}$

/* $G1$ is $G$ with deleting edge $x_{i,k}$ */

Step 5: the same as step 2.

Step 6: the same as step 3.

6.1: 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 $x_i$. If the checking procedure can not identify $x_i$ as not a reducible node (step 3.1 or step

6.1) then $x_i$ is a reducible node. The maximal number of the iteration of the checking procedure for node $x_i$ is equal to the number of elements in the set of $(FA_i \cap FN) \cup (PA_i \cap PN)$. The formal REDUCIBLE_NODE algorithm is given below.

**REDUCIBLE_NODE** ( $G$ )
**begin**
**for** all node $x_i \in G$ **do**
  **if** degree($x_i$) = 2 **then**
    **begin**
/* assume that the two edges incident on node $x_i$ are $x_{i,j}$ and $x_{i,k}$ */
      $G1 = G - x_{i,j}$ /* delete $x_{i,j}$ from $G$ */
      **for** all files $f \in (FA_i \cap FN)$ and all program $p \in (PA_i \cap PN)$ **do**
        delete all nodes in $G1$ that contain file $f$ or program $p$ from $G1$ except node $x_i$,
        $G2$ = the component that contains node $x_i$ in $G1$
        **if** there are some FSTs in $G2$ **then**
          go to check_next_node
      **od**
      $G1 = G - x_{i,k}$   /* delete $x_{i,k}$ from $G$ */
      ..........
      the same as the above *for-loop*
      ..........
/* the $x_i$ is a reducible node, apply degree-2 reduction */
      $G = G - x_i - x_{i,j} - x_{i,k} + x_{j,k}'$
      $p_{j,k}' = p_{i,j} * p_{i,k}$
      $FA_j = FA_j \cup FA_i$   ( or $FA_k = FA_k \cup FA_i$ )
      $PA_j = PA_j \cup PA_i$   ( or $PA_k = PA_k \cup PA_i$ )
    **end**
check_next_node:
  **od**
**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
  $FN = \bigcup_{P_j \in PN} FN_j$
/* all the needed data files for program $P_j$ in $PN$ */
  $R=0$ /* the reliability set to $0$ */
  search a node $x_i$ that contains program $P_j \in PN$
  **if** node $x_i$ is not found **then**
  **begin**
    **output**($R$)
    **stop**
  **end**
  $s = i$     /* starting node's number */
  $R = REL(G_s)$
  **output**($R$)
  **stop**
end (* FREA *)

*function* REL($G_s$)
**begin**
Step 1: The checking step
  **if** $FA_s \supseteq FN$ **and** $PA_s \supseteq PN$ **then**
    **begin**
      $REL=1$
      **return**
    **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 */
  **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:   $G_s''' = G_s'' = G_s'$
/* $G_s'''$ and $G_s''$ are temporary variables for graph $G_s'$ */
  $R=0$   /* set reliability to $0$ */
  $C=1$
  /* the constant terms, $\ldots q_{s,1} q_{s,2} \ldots p_{s,h}$, of equation (4.3) */
  **for all** $x_{s,j} \in$ the set of edges incident on starting node $x_s$ **do**
    $C = C * p_{s,j}$
3.3:   $R = R + C * REL(G_s''' \oplus x_{s,j})$
    $C = C * q_{s,j}$
3.4:   $G_s'' = G_s''' - x_{s,j}$
3.5:   $G_s'''$ = the new graph after deleting irrelevant components from $G_s''$
    **if** $x_s$ is deleted **then**
      **go to step 4**
  **od**
Step 4 : The choosing step to find the new staring node
  **if** finding a node $x_k$ in $G'''$ that contains the programs under consideration **then**
    **begin**
      $s = k$
      $R = R + C * REL(G_s''')$
    **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 reductions 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.

| | |
|---|---|
| G | G'' |
| G' | G''' |

Fig. 11 The basic node structure of trace tree

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 $G0'$ in the trace node $N_0$ have starting node $x_s$ with $k$ incident edges then the maximal number of trace tree nodes that trace tree node $N_0$

can derive is $k$+1 (refer to equation (4.3) ). Since only $k$+1 terms (intermediate subgraphs) can be generated, components $Gk+1''$ and $Gk+1'''$ within the trace tree node $N_{k+1}$ are nil. The $S_j$ represents the operations to be applied from $G'$ in trace tree node $N_0$ to trace tree node $N_j$. The operations available for $Sj$ can be deleting, merging, or combinations of merging and deleting. For example, $S_j=\overline{x_{s,1}}\ x_{s,2}$ means that edge $x_{s,1}$ in component $G0'$ is deleted and then $G0'$ is merged with edge $x_{s,2}$ to produce a new intermediate subgraph $Gj$ within trace tree node $N_j$. The symbol $\rightarrow$ indicates which intermediate subgraph is generated by which intermediate subgraph. For example, $G1$ in trace tree node $N_1$ is obtained from the $G0'$ within trace tree node $N_0$ by applying operation $S_1$( written as $G1 = G0'\oplus x_{s,1}$ using the notation defined in the FREA algorithm ). The rest of the relations are listed below.

$G1=G0' \oplus x_{s,1}$ /* step 3.2 and 3.3 */
$G1'$=the reduction graph of $G1$ /* step 3.1 */
$G1''=G1'-x_{s,1}$ /* step 3.2 and 3.4 */
$G1'''$=the reduction graph of $G1''$ /* step 3.5 */
$Gi=Gi-1''' \oplus x_{s,i}$ /* step 3.3 */
$Gi'$=the reduction graph of $Gi$ /* step 3.1 */
$Gi''=Gi-1'''-x_{s,i}$ /* step 3.4 */
$Gi'''$ = the reduction graph of $Gi''$ for $i$=2,3,....,$k$ /* step 3.5 */
$Gk+1=Gk'''$ with a new starting node /* step 4 */
$Gk+1'$=the reduction graph of $Gk+1$ /* step 3.1 */

If the starting node $x_s$ in component $G$ within trace tree node $N_j$ holds all data files required and programs to be executed then $N_j$ is a leaf node of the trace tree. Figure 13 depicts the trace tree for program 1 to be executed in figure 1, and the $DPR_1$ can be computed as

$DPR_1 = p_1+ q_1p_2(p_3+q_3p_5) + q_1q_2p_6$
$= p_1+ q_1p_2(p_3+q_3p_5) + q_1q_2(p_3p_4+p_5-p_3p_4p_5)$
$= p_1+ q_1p_2p_3 + q_1p_2q_3p_5 + q_1q_2p_3p_4 + q_1q_2p_5$ ( where $p_i$ is the probability of link i, and $q_i=1-p_i$ )

Let the probability of any link being operational be 0.9, then $DPR_1$ is computed to 0.99891.

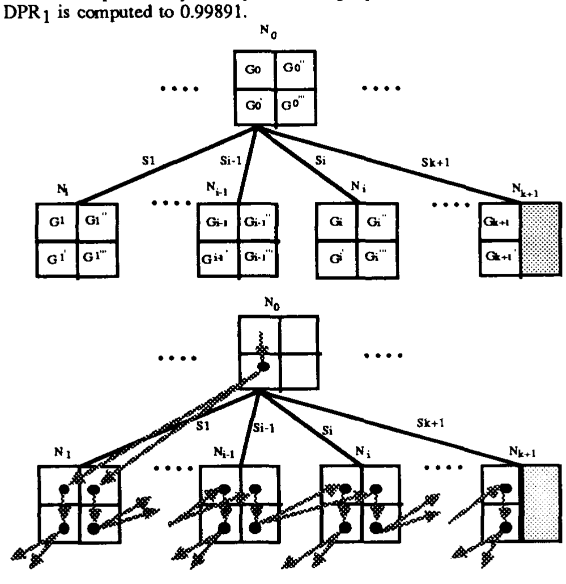

Fig. 12 The trace tree structure

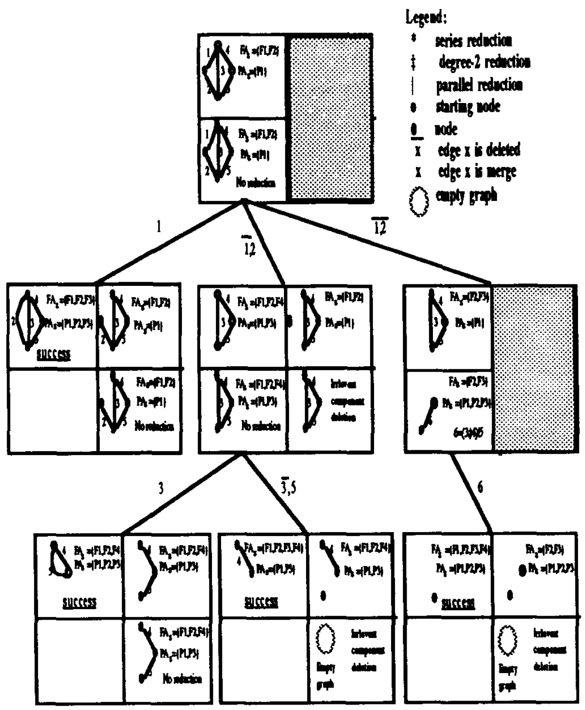


Fig. 13 The trace tree of FREA for the example of figure 1

Figure 14 is another example of DCS and figure 15 is the trace tree for program 4 under consideration in figure 14, and the $DPR_4$ can be computed as

$DPR_4 = p_4p_{11} + q_4p_9p_{12}$
$= p_4[1-q_7(1-p_8p_{10})] + q_4[(1-(1-p_1p_2)q_3)p_5]p_{12}$
$= p_4(1-q_7(1-p_8[1-q_6(1-p_9)])) + q_4((1-(1-p_1p_2)q_3)p_5)[1-q_8(1-p_6p_7)]$
$= p_4(1-q_7(1-p_8(1-q_6(1-[(1-(1-p_1p_2)q_3)p_5])))) + q_4((1-(1-p_1p_2)q_3)p_5)(1-q_8(1-p_6p_7))$
$= p_4 - p_4q_7+p_4q_7p_8 - p_4q_6q_7p_8 + p_4p_5q_6q_7p_8 - q_3p_4p_5q_6q_7p_8 + p_1p_2q_3p_4p_5q_6q_7p_8 + q_4p_5 - q_3q_4p_5 + p_1p_2q_3q_4p_5 - q_4p_5q_8 + q_3q_4p_5q_8 - p_1p_2q_3q_4p_5q_8 + q_4p_5p_6q_8 - q_3q_4p_5p_6p_7q_8 + p_1p_2q_3q_4p_5p_6p_7q_8$

( where $p_i$ is the probability of link i, and $q_i=1-p_i$ )

Let the probability of any link being operational be 0.9, then $DPR_4$ 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 counting approach which counts the number of intermediate trees or subgraphs generated during the whole reliability evaluation. From such a comparison, one can tell how much memory space and time units are required for their algorithms to run the distributed programs under the effects of different sizes of DCS, data file distributions, program distributions, 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.

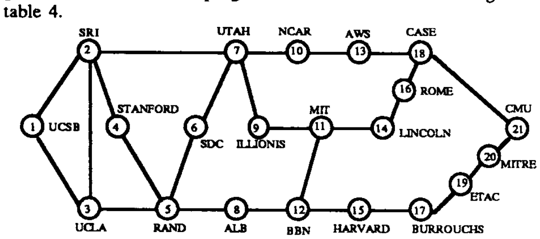

Fig. 18 ARPA computer network

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

| files | nodes |
|---|---|
| F1 | 11,14,19 |
| F2 | 1,14,21 |
| F3 | 2,5,17 |
| F4 | 9,15 |
| F5 | 6,12,20 |
| F6 | 1,5,18 |
| F7 | 3,11,15 |
| F8 | 9,16 |
| F9 | 10,18 |
| F10 | 4,10,13 |
| F11 | 2,7 |
| F12 | 8 |

| programs | nodes |
|---|---|
| P1 | 1 |
| P2 | 14 |
| P3 | 2 |
| P4 | 15 |
| P5 | 9 |
| P6 | 21 |
| P7 | 19 |
| P8 | 6 |
| P9 | 8 |
| P10 | 4 |

| programs | files required |
|---|---|
| P1 | F1,F3,F5,F7 |
| P2 | F2,F4,F6,F8 |
| P3 | F9,F10,F11 |
| P4 | F10,F11,F12 |
| P5 | F6,F7 |
| P6 | F1,F6,F7 |
| P7 | F1,F8,F12 |
| P8 | F3,F4,F5,F6 |
| P9 | F1,F11 |
| P10 | F4,F8,F12 |

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

| algorithm | $P_1$ | $P_2$ | $P_3$ | $P_4$ | $P_5$ |
|---|---|---|---|---|---|
| MFST[6] | 55700 | 70842 | 172907 | 197541 | 17292 |
| FARE[13] | 20007 | 13923 | 35515 | 38120 | 3300 |
| FREA | 412 | 57 | 70 | 184 | 75 |
| DPR | 0.97084 | 0.97393 | 0.97668 | 0.93457 | 0.98475 |
| algorithm | $P_6$ | $P_7$ | $P_8$ | $P_9$ | $P_{10}$ |
| MFST[6] | 39893 | 82759 | 44017 | 72005 | 257333 |
| FARE[13] | 13075 | 25135 | 11141 | 22436 | 66752 |
| FREA | 95 | 25 | 152 | 55 | 290 |
| DPR | 0.93348 | 0.91438 | 0.98217 | 0.97039 | 0.96954 |

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

[2] T. C. K. Chou and J. A. Abraham, "Load Redistribution under Failure in Distributed Systems", *IEEE Trans. Comput.*, Vol. C-32, pp. 799-808, Sep. 1983.

[3] D. W. Davies, E. Holler, E. D. Jensen, S. R. Kimbleton, B. W. Lampson, G. Lelann, K. J. Thurber, and R. W. Watson, "Distributed Systems Architecture and Implementation," in *Lecture Notes in Computer Science*, vol. 105. Berlin, Germany : Springer-Verlag, 1981.

[4] P. Enslow, "What Is a Distributed Data Processing System", *IEEE Computer*, Vol. 11, Jan. 1978.

[5] J. Garcia-Molina, "Reliability Issues for Fully Replicated Distributed Database", *IEEE Computer*, vol. 16, pp. 34-42, Sep. 1982.

[6] V. K. Prasnna Kumar, S. Hariri and C. S. Raghavendra, "Distributed Program Reliability Analysis", *IEEE Trans. Software Eng.*, Vol. SE-

[11] R. Kevin Wood, "Factoring Algorithms for Computing K-terminal Network Reliability", *IEEE Trans. Reliability*, Vol. R-35, pp.269-278, Aug. 1986.

[12] S. Hariri and C. S. Raghavendra, "SYREL: A Symbolic Reliability Algorithm based on Path and Cutset Methods", USC Tech. Rep., 1984.

[13] A. Kumar, S. Rai and D. P. Agrawal, "Reliability Evaluation Algorithms for Distributed Systems," in *Proc. IEEE INFOCOM 88*, pp.851-860, 1988.

[14] A. Kumar, S. Rai and D. P. Agrawal, "On Computer Communication Network Reliability Under Program Execution Constraints", *IEEE Journal on Selected Areas in Communication*, Vol. 6, No. 8, pp. 1393-1399, Oct. 1988.

[15] Fred Moskowitz, "The analysis of redundancy networks", *AIEE Trans. (Commun. Electron.)*, vol 29, 1958, pp 627-632.

[16] Michael O. Ball, "Computing network reliability", *Operations Research*, vol 27, pp 132-143.

[17] Mark K. Chang, "A graph theoretic appraisal of the complexity of network reliability algorithms", PhD Thesis, Dept. of IEOR, University of California Berkeley, 1981.

[18] A. Satyanarayana, M. K. Chang, "Network reliability and the factoring theorem", *Networks*, vol 13, 1983, pp 107-120.

[19] R. Kevin Wood, "A factoring algorithm using polygon-to-chain reductions for computing $K$-terminal network reliability", *Networks*, vol 15, 1985, pp 173-190.

[20] C. S. Raghavendra, V. K. Prasnna Kumar, and S. Hariri, "Reliability Analysis in Distributed System," *IEEE Trans. Comput.*, Vol. C-37, pp.352-358, Mar. 1988.