Brief Contributions

On Distributed Computing Systems Reliability Analysis Under Program Execution Constraints

Deng-Jyi Chen, *Member, IEEE*, and Min-Sheng Lin

***Abstract*—This correspondence presents an algorithm for computing the reliability of distributed computing systems (DCS). The algorithm, called the Fast Reliability Evaluation Algorithm, is based on the factoring theorem employing several reliability preserving reduction techniques. 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 economical in both time and space. To compute the distributed program reliability, the ARPA network is studied to illustrate the feasibility of the proposed algorithm.**

***Index Terms*—Distributed program, distributed system, factoring theorem, graph theory, reliability, reliability-preserving reduction, spanning tree.**

I. Introduction

Recently, the 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's), memory units, data files, and programs as its resources. These resources are interconnected via a communication network that dictates how information could flow between PE's. Programs residing on some PE's can run using data files at other PE's as well. For successful execution of a program, it is essential that the PE containing the program and other PE's that have the required data files, and communication links between them must be operational. Using this concept, distributed program reliability (DPR) is defined as the probability of successful execution of a distributed program that runs on some PE's and needs to communicate with other processing elements for remote files. Distributed system reliability (DSR) is defined as the probability that all programs with distributed files can run successfully despite some faults occurring in the PE's and/or in the communication links [6].

In [6], a 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 this method, all MFST's are obtained by using the breadth-search method. After finding the MFST's, since 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 a lot of replicated trees during the processing and thus will be inefficient. Instead of generating MFST's, one algorithm, called FARE, has been proposed in [13] and [14] to compute DPR directly by using a connection matrix. Based on the assumption that the PE's (nodes) in the DCS are perfect, it does not require additional reliability evaluation algorithms to convert a multiterminal connection into a reliability expression. The shortcoming of this algorithm is that it is not applicable for distributed programs running on more than one node.

In this correspondence, we propose a new algorithm called the Fast Reliability Evaluation Algorithm (FREA) that employs a different concept to compute the reliability of DSR and DPR. It is based on the generalized factoring theorem with several reliability preserving reductions to reduce the computation tree. The factoring theorem for the exact computation of K-terminal reliability in undirected networks has been proposed since 1958 by Moskowitz [15]. Recently, several papers have addressed worst case computational complexity and the optimality of classed factoring algorithms and related algorithms, for example, Ball [16], Chang [17], Satyanarayana and Chang [18], and Wood [19] to name a few. Unlike the K-terminal reliability problem, where K-terminal nodes are fixed and given, the distributed program reliability problem does not have fixed K -terminal nodes. The K-terminal nodes in the distributed program reliability analysis can be changed dynamically due to the effects of link or node failure, using data files and programs distribution, and the topology of the network. Therefore, we may say the network reliability problem is considered to be static-oriented, whereas the distributed program reliability problem is dynamic-oriented. Naturally, distributed program reliability problems are considerd to be more complex and difficult than computer network reliability problems.

II. Notations and Definitions

Notations and definitions used in the rest of correspondence are summarized here.

$G = (V, E)$	Undirected DSC graph in which the set of vertices (nodes) in V represents the PE's and the links (edges) in E represent the communication links.
x_i	Node representing a processing element i.
$x_{i,j}$	Link between processing elements i and j.
G_s	G with a node x_s, called starting node, indicates where the FREA algorithm begins to generate subgraphs.
$p_{i,j}(q_{i,j})$	Probability that link $x_{i,j}$ works (fails).
F_i	Data file i.
P_i	Distributed program i.
PA_i	Set of programs that can be run at processing element x_i.
FA_i	Set of data files available at processing element x_i.
FN_i	Set of data files needed to execute P_i.
PN	Set of programs to be executed.
FN	Set of data files needed to execute all programs in PN (i.e., $FN = \cup_{P_i \in PN} FN_i$).
FST	Spanning tree that connects the root node (the processing element that runs the program under consideration) to other nodes, such that its vertices hold all the needed files for the program under consideration.
MFST	An FST such that there exists no other FST that is a subset of it.

Manuscript received March 7, 1991; revised November 15, 1991. This work was supported in part by the National Science Council under Contract NSC-80-0408-E009-16 and in part by the Chung San Institute of Technology under Contract 7S79-0210-D009-03.

The authors are with the Institute of Computer Science and Information Engineering, National Chiao-Tung University, Hsin Chu, Taiwan, Republic of China.

IEEE Log Number 9204158.

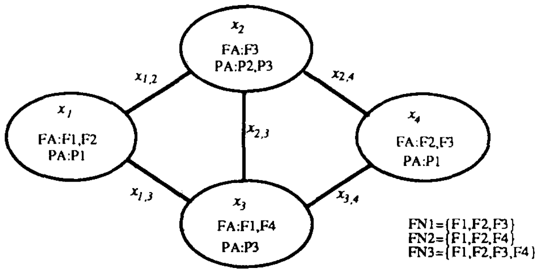

Fig. 1. Simple distributed computing system.

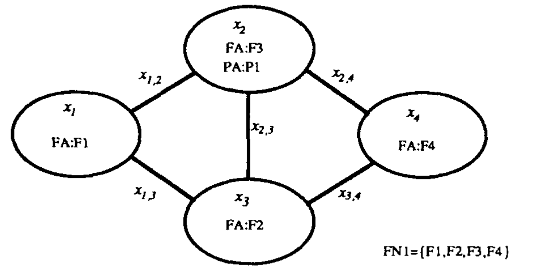

Fig. 2. Simple distributed computing system with different file distribution.

$G - x_{i,j}$ Graph G with edge $x_{i,j}$ deleted.

$G \oplus x_{i,j}$ Graph G with edge $x_{i,j}$ contracted such that nodes x_i and x_j are merged into a single node. This new merge node contains all data files and programs that were in nodes x_i and x_j.

$R(G)$ Reliability of the DCS graph G.

Since trees and subgraphs are used to represent the intermediate communication structure of the DCS, they are used interchangeably in the rest of this correspondence.

III. Distributed Program Reliability Analysis

Considering the distributed computing system in Fig. 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 (F_1 and F_2) and can run P_1 directly from here to communicate with other nodes for accessing data files required to complete the execution of P_1. Detailed information for each node is summarized in FA_j, PA_j, and FN_j $(j = 1, \cdots, 4)$ in Fig. 1.

Let program P_1 require F_1, F_2, and F_3 to complete its execution in the DCS. Also, P_1 can be run on both nodes x_1 and x_4 in the DCS (Fig. 1). We can identify some file spanning trees (FST's) rooted on x_1 from the DCS graph: 1) $x_1x_2x_{1,2}$, 2) $x_1x_2x_3x_{1,2}x_{2,3}$, 3) $x_1x_2x_4x_{1,2}x_{2,4}$, 4) $x_1x_2x_3x_{1,3}x_{2,3}$, 5) $x_1x_3x_4x_{1,3}x_{3,4}$, 6) $x_1x_2x_3x_4x_{1,2}x_{2,3}x_{3,4}$, 7) $x_1x_2x_3x_4x_{1,2}x_{2,4}x_{3,4}$, 8) $x_1x_2x_3x_4x_{1,3}x_{2,3}x_{2,4}$, and 9) $x_1x_2x_3x_4$ $x_{1,3}x_{3,4}x_{2,4}$.

If P_1 can be run only on node x_1, the MFST's are 1) $x_1x_2x_{1,2}$, 4) $x_1x_2x_3x_{1,3}x_{2,3}$, and 5) $x_1x_3x_4x_{1,3}x_{3,4}$.

If we also consider the FST rooted on x_4, then the MFST's for P_1 are 1) $x_1x_2x_{1,2}$, 4) $x_1x_2x_3x_{1,3}x_{2,3}$, *) $x_3x_4x_{3,4}$, and *) $x_2x_3x_4x_{2,3}x_{2,4}$. The last two MFST's marked by * are rooted on node x_4 instead of x_1.

Since the MFST's connect the root node (the PE that runs the program under consideration) to some other nodes such that its nodes hold all the needed files for the program under execution, the DPR can then be determined by computing the probability that at least one of these MFST's is working. Thus the distributed program reliability for a given program j can be defined as the probability that at least one MFST of program j is working [6]. The DPR measures the reliability of a particular distributed program. For the entire DCS to be operational, several such programs or a given set of distributed programs must be operational. A system-level reliability measure for all distributed programs to be operational is defined in [6] as the probability that at least one MFST of all distributed programs is working.

For computing the reliability of the entire DCS, the concept of MFST has been extended to the minimal file spanning forest (MFSF) [14]. Based on the concepts of the MFST and MFSF, Kumar and his colleagues developed algorithms to generate all MFST's [6] and MFSF's [20], respectively. Once the MFST's and MFSF's are obtained, SYREL [12] is 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 Fig. 2, for finding all the MFST's for P_1, let us use Kumar's algorithm [6] to generate the MFST's. The algorithm starts from finding the MFST's of size 0, and then size 1, $\cdots$ until size $n-1$. As we can see in Fig. 3, the replicated trees (e.g., trees B, d2, and d4 are replicated) have been generated by their algorithm. Thus a procedure, called CLEAN, is required to remove these replicated trees.

Because the MFST's 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 the 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 cannot be used to evaluate the reliability of a program running on more than one node.

IV. Derivation of FREA Algorithm

In this section, we present a new algorithm, called FREA, for the reliability evaluation of DCS. The FREA algorithm is based on the generalized factoring theorem employing several reliability preserving reductions to reduce the size of computed graphs and to simplify the reliability computation. To illustrate our approach, we begin by presenting the concept of a generalized factoring theorem and then several reliability preserving reductions.

A. 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:

$$R(G) = p_{i,j}R(G \oplus x_{i,j}) + q_{i,j}R(G - x_{i,j}). \quad (1)$$

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

$$R(S) = p_{i,j}R(S|x_{i,j} \text{ works}) + q_{i,j}R(S|x_{i,j} \text{ fails}). \quad (2)$$

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

$$\begin{aligned} R(G_s) = &p_{s,1}R(G \oplus x_{s,1}) \\ &+ q_{s,1}p_{s,2}R(G - x_{s,1} \oplus x_{s,2}) + \cdots \end{aligned}$$

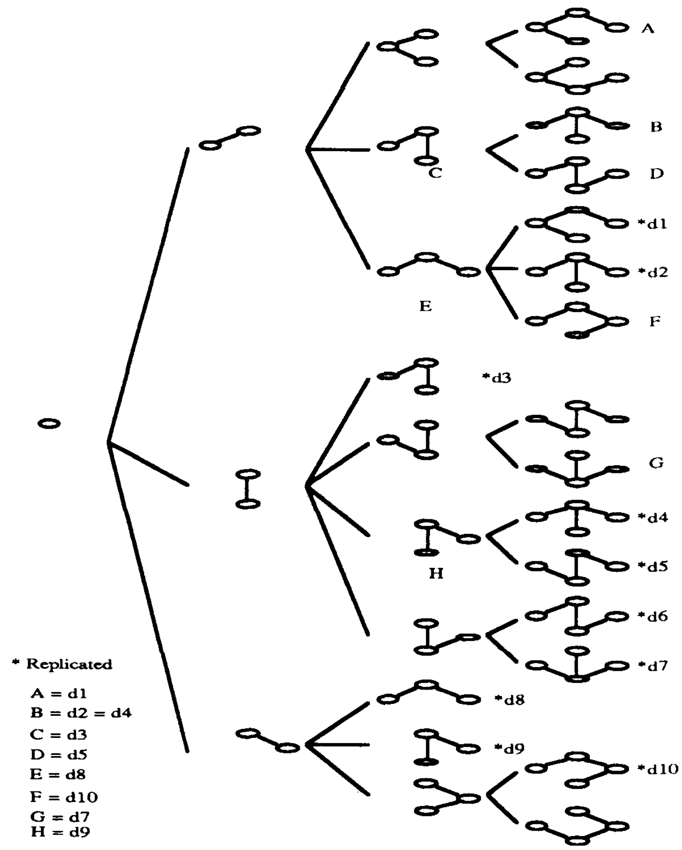

Fig. 3. Generation of replicated trees in MFST [6] algorithm.

$$+ q_{s,1}q_{s,2}\cdots q_{s,k-1}p_{s,k}R(G - x_{s,1} - x_{s,2} - \cdots - x_{s,k-1} \oplus x_{s,k})$$
$$+ q_{s,1}q_{s,2}\cdots q_{s,k}R(G - x_{s,1} - x_{s,2} - \cdots - x_{s,k}). \quad (3)$$

Equation (3) is obviously true. For the proof of its correctness, readers are referred to [21]. Equation (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 FST's is obtained. The induced graph of the former case represents a success, wheres the latter case represents a failure. It is easy to see that subtrees (or subgraphs) generation based on (3) will be completely disjoint. Since all of these disjoint terms represent either a success or a failure, one can simply sum all these disjoint terms together to produce the reliability expression of the system. Thus, the dominant factor for the reliability computation becomes the subgraph generation which is the process to produce these disjoint terms. Since the subgraph generation based on (3) will be completely disjoint, it guarantees no replicated trees will be generated during the expansion of the tree. This is one of the key reasons why the FREA algorithm will generate less subgraphs than existing algorithms. The other major reason will be the use of several reliability preserving reduction techniques, which will be discussed in the following section, to reduce the size of the graph.

B. Reliability Preserving Reductions for the DCS Reliability Evaluation

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.

Definition 1: Degree-1 Reduction Degree-1 reduction is to remove nodes and their incident edges that contain no needed data files and programs under consideration. Considering the DCS in Fig. 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 Fig. 4.

Definition 2: 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 the components of G. If there are no FST's in G^0 then the component G^0 is irrelevant and a reduction is applied to delete component G^0.

Definition 3: 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 the K-terminal network reliability problem.

Definition 4: 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 the DCS reliability problem is the same

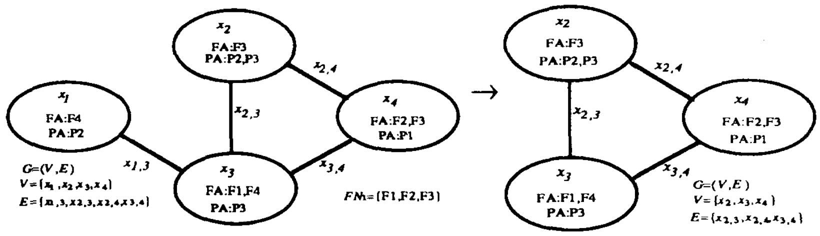

Fig. 4. Example of degree-1 reduction.

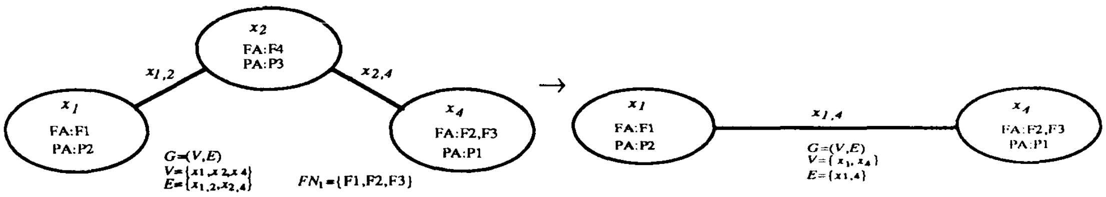

Fig. 5. Example of series reduction.

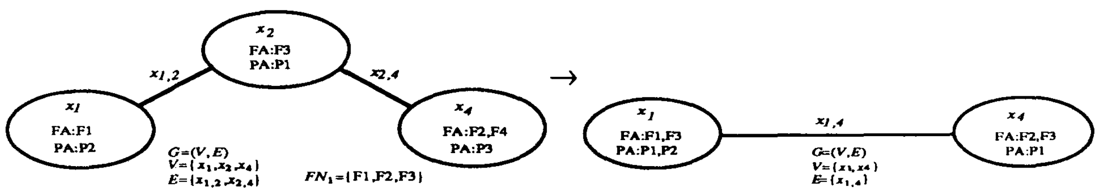

Fig. 6. Example of degree-2 reduction.

as the preceding description except that the condition of $x_i \notin K$ is replaced by $FA_i \cap FN = \emptyset$ and $PA_i \cap PN = \emptyset$. 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, Fig. 5 presents a case of series reduction for computing DPR_1.

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 later in the degree-2 reduction.

Definition 5: 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 MFST's of P_j that contains node x_i must also be two.

Theorem 1: Node x_i is a reducible node for distributed program P_g if it satisfies the following conditions:

a) Node degree is two, and
b) $FA_j \supseteq (FA_i \cap FN)$ and $PA_j \supseteq (PA_i \cap PN)$ and $FA_k \supseteq (FA_i \cap FN)$ and $PA_k \supseteq (PA_i \cap PN)$ (where node x_k and x_j are the two adjacent nodes of x_i).

Proof: Case 1: Some MFST$_t$ generated for DPR$_g$ contain node x_i. Suppose x_i satisfies the properties of Theorem 1 and x_i is not a reducible node, then it implies either i) x_i's node degree is not two, or ii) x_i's node degree in the MFST $_t$ is not two according to the definition of a reducible node. In the former case, that x_i's node degree is not two is violated in the first given property in Theorem 1 that declares the degree of node x_i is two (since we assume x_i satisfies the properties of Theorem 1). Thus, it must be the latter case, that is, x_i's node degree in the MFST$_t$ is not two. Since the first given property in Theorem 1 states that the degree of node x_i is two, the MFST$_t$ that contains node x_i can only have the degree of node x_i less than or equal to two. Furthermore, in the latter case, 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 1, 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 words, 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 MFST's contain node x_i. Theorem 1 is obviously true for this case.

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.

Definition 6: 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. Fig. 6 presents an example of such reduction.

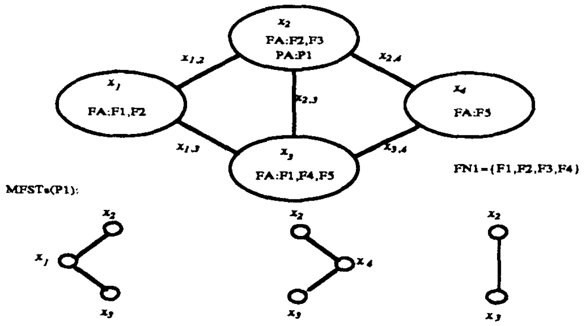

Fig. 7. Example of DCS and all MFST's for program 1 under consideration.

To prove degree-2 reduction is correct for DPR analysis is trivial; readers are referred to [21]. In fact, the series reduction is just a special case of degree-2 reduction that meets the properties of Corollary 1.

C. Identification of Reducible Nodes

In this subsection, we propose an algorithm to identify all reducible nodes in a DCS graph.

Let us consider the DCS shown in Fig. 7. Although x_1 and x_4 are reducible nodes by the definition of the reducible node, only x_4 can be identified based on 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 MFST's, and then to validate the nodes of those MFST's that contain the reducible nodes. However, such a solution inherits the problem in Kumar *et al.* [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 MFST's. 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 $x_{i,k}$ are 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 that 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 file, 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 FST's in the component of $G1$ that contains x_i.
/* using the Depth-First-Search algorithm */
3.1: If there are some FST's in this component then x_i must be a leaf node of some MFST's.
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.

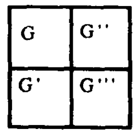

Fig. 8. Basic node structure of trace tree.

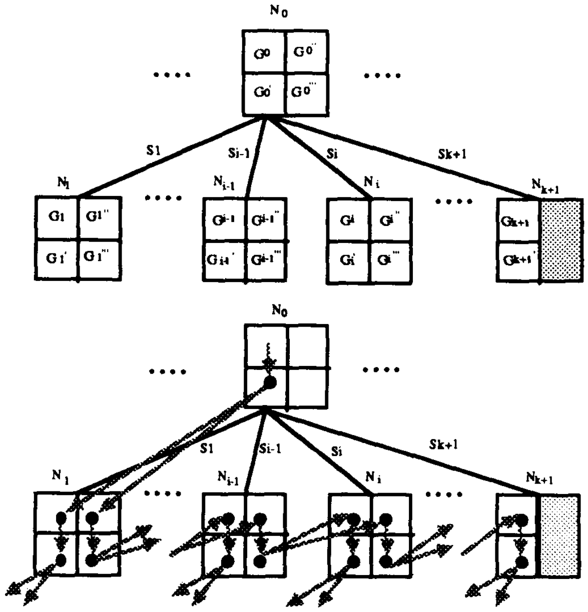

Fig. 9. Trace tree structure.

We repeat the preceding steps to check the other needed data files and programs under consideration that are also in x_i. If the checking procedure cannot identify x_i as an irreducible 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 at the bottom of the page.

D. FREA Algorithm

Once the way of finding all the reducible nodes is understood, we can use (3) and the reliability preserving reductions discussed in Section IV-B to compute the DPR and DSR. The complete FREA algorithm is listed on the next page.

E. 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 Fig. 8, which represents the intermediate trees or subgraphs from the reduction process.

The relationship of trees within a trace tree node, using notation defined in FREA, can be explained by the following example. A trace tree is given in Fig. 9.

Suppose intermediate tree G'_0 in the trace node N_0 has started 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 (3)).

```
FREA ALGORITHM
begin
    G = the original DCS graph
    FN = ∪_{Pj∈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 ∈ 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 ⊇ FN and PA_s ⊇ PN then
          begin
              REL = 1
              return
          end
      if there are no FST's in G_s then        /* using DFS algorithm to check this */
          begin
              REL = 0                          /* no FST's 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 (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, ... q_{s,1} q_{s,2} ... p_{s,h}, of equation (3) */
      for all x_{s,j} ∈ the set of edges incident on starting node x_s do
              C = C* p_{s,j}
3.3:          R = R + C* REL(G'''_s ⊕ 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 *)
```

Since only $k+1$ terms (intermediate subgraphs) can be generated, components G''_{k+1} and G'''_{k+1} within the trace tree node N_{k+1} are nil. 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 S_j 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 G'_0 is deleted and then G'_0 is merged with edge $x_{s,2}$ to produce a new intermediate subgraph G_j within trace tree node N_j. The symbol $\rightarrow$ indicates which intermediate subgraph is generated by which intermediate subgraph. For example, G_1 in trace tree node N_1 is obtained from the G'_0 within trace tree node N_0 by applying operation S_1 (written as $G_1 = G'_0 \oplus x_{s,1}$ using the notation defined in FREA). The rest of the relations are listed at the bottom of the page.

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. Fig. 10 depicts the trace tree for program 1 to be executed in Fig. 1, where link $x_{1,2}$ corresponds to link 1, link $x_{1,3}$ corresponds to link 2, $\cdots$, etc.

DPR_1 can be computed as

$$\begin{aligned}
\mathrm{DPR}_1 &= p_1 + q_1p_2(p_3 + q_3p_5) + q_1q_2p_6 \\
&= p_1 + q_1p_2(p_3 + q_3p_5) + q_1q_2(p_3p_4 + p_5 - p_3p_4p_5) \\
&= p_1 + q_1p_2p_3 + q_1p_2q_3p_5 + q_1q_2p_3p_4 \\
&\quad + q_1q_2p_5 - q_1q_2p_3p_4p_5.
\end{aligned}$$

where p_i is the probability of link i in Fig. 10, and $q_i = 1 - p_i$.

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

V. Algorithm Comparison

Unlike the K-terminal reliability problem, where K -terminal nodes are fixed and given, the distributed program reliability problem does not have fixed K-terminal nodes. The K -terminal nodes in the distributed program reliability analysis can dynamically be changed due to the effects of link or node failure, the ways of data files and program distribution, and the topology of the network. Therefore, we may say the network reliability problem is considered to be static-oriented while the distributed program reliability problem is dynamic-oriented. Naturally, the DPR problem is considered to be more complex and difficult than the computer network reliability problem. In fact, computing reliability of this type of problem has been known as a NP-hard problem.

In this section, comparisons with existing algorithms [6], [13], [14], [20] are given. The algorithms presented in [6], [13], [14], and [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 seldom occurs since once an MFST is found the tree expansion is stopped. The FREA algorithm employs the generalized factoring theorem with several reduction concepts to speed up the whole reliability evaluation. A 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 approximate 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. We also provide some actual execution results to support these analyses. The following subsections focus on these different comparisons.

A. Effect of Different Sizes on Performance of Different Algorithms

Fig. 11 is a well-known example of a computer communication network—the ARPA computer network in which there are 21 nodes

```
REDUCIBLE_NODE (G)
begin
  for all node x_i ∈ 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 ∈ (FA_i ∩ FN) and all program p ∈ (PA_i ∩ 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 FST's 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 ∪ FA_i   (or FA_k = FA_k ∪ FA_i)
        PA_j = PA_j ∪ PA_i   (or PA_k = PA_k ∪ PA_i)
      end
check_next_node:
  od
end (* REDUCIBLE_NODE *)
```

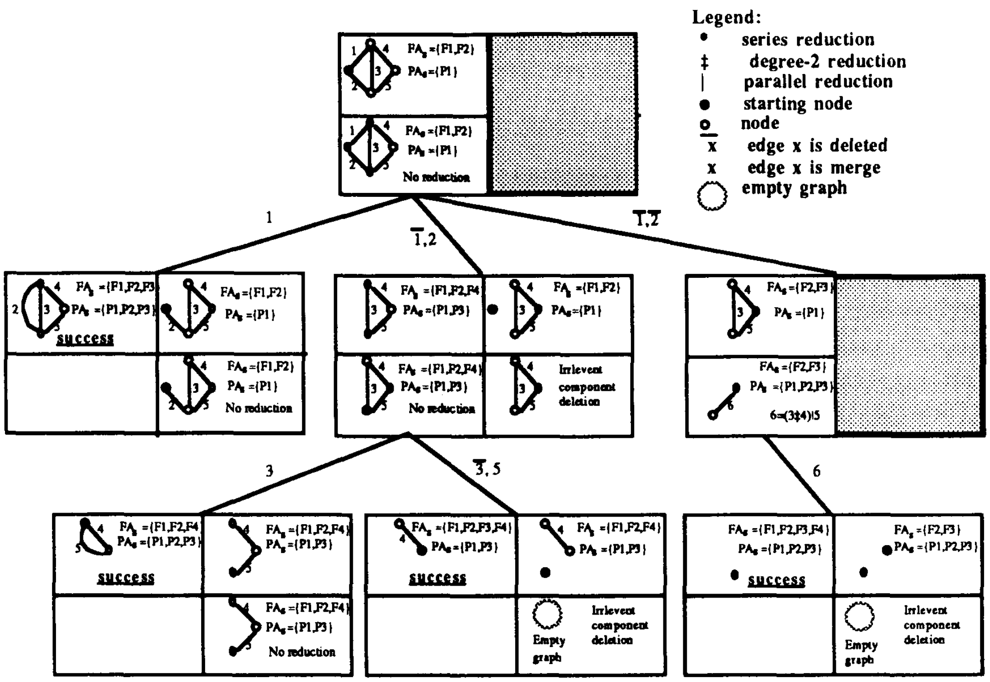

Fig. 10. Trace tree of FREA for example of Fig. 1.

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 I, II, and III, respectively. The number of subgraphs generated for different programs under consideration are given in Table IV.

It is clear that the FREA algorithm is thousands of times less than that of the existing algorithms in a large and complex distributed network such as ARPA.

B. *Effect of Topology on Performance of Different Algorithms*

In this study, we want to see the effect of topological configuration on the performance of different algorithms used. Thus, we run a different set of programs and file distributions over various topologies starting from a simple loop to a completely connected graph. These topologies are shown in Fig. 12, and the file distributions, program distributions, and data files needed for the program to be executed are given in Tables V, VI, and VII, respectively. These topologies, file distributions, and program distributions are the same as those used in [13]. Fig. 13 shows the number of subgraphs generated versus different topologies based on program 1 as executed at node 1.

TABLE I
FILE DISTRIBUTIONS

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

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

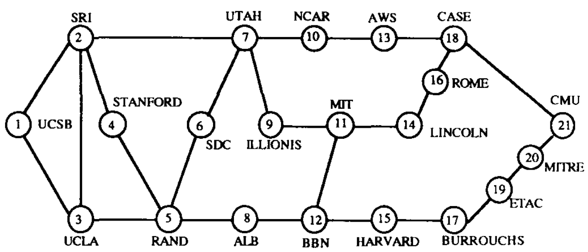

Fig. 11. ARPA computer network.

TABLE II
PROGRAM DISTRIBUTIONS

Programs	Nodes
P1	1
P2	14
P3	2
P4	15
P5	9
P6	21
P7	19
P8	6
P9	8
P10	4

TABLE III
DATA FILES NEEDED FOR EXECUTING A PROGRAM P_i

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

Other results also follow a similar curve and are reported in [21]. From these comparisons, it is clear that the FREA algorithm is the fastest (best) one, compared with the other algorithms, in any of these different topologies.

C. Effect of Data File Distributions on Performance of Different Algorithms

Eight different sets of data file distributions, generated randomly based on the topology in Fig. 14 for the comparison of three algorithms, are listed in Table VIII. The program distribution and data files needed for the program to be executed are referred to Tables VI and VII, respectively. Fig. 15 depicts that the number of subgraphs versus different data file distributions based on program 4 is executed at node 2. Other results also follow the similar curve and are reported in [21].

From the preceding comparisons, it is clear that the FREA algorithm has the best performance in these different data file distributions.

D. Effect of Program Distributions on Performance of Different Algorithms

Fig. 16 shows the effect of programs running on different nodes based on the DCS in Fig. 14. The data file distributions and data files

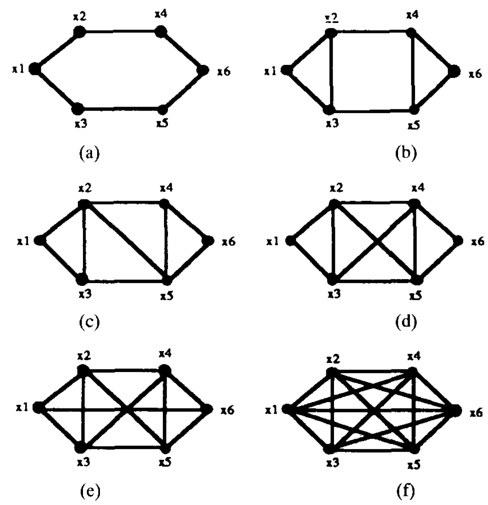

Fig. 12. Various topologies.

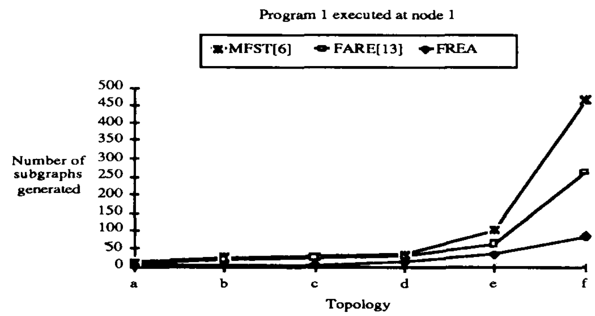

Fig. 13. Number of subgraphs generated versus different topologies.

TABLE IV
NUMBER OF SUBGRAPHS GENERATED AND DPR FOR EXAMPLE OF ARPA COMPUTER NETWORK

Program / 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	0.9708450	0.9739356	0.9766832	0.9345704	0.9847566

Program / 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	0.9334858	0.9143801	0.9821738	0.9703900	0.9695497

needed for each program to be executed are referred to in Tables V and VII, respectively. Other results also follow the similar curve and are reported in [21].

E. DPR Analysis of Running the Same Distributed Program from More than One Site

In this section, we compare the effect of the same program when executed from more than one site (node). From the example in Fig. 17, P_1 can be executed at node x_1 or x_6; P_2 can be executed at node

TABLE V
FILE DISTRIBUTIONS

Files	Nodes
F1	1, 2, 3
F2	2, 4
F3	3, 5
F4	3, 6
F5	1, 4
F6	5

TABLE VI
PROGRAM DISTRIBUTIONS

Nodes	Program
1	P1
2	P4
3	P2, P3
4	P2, P3
5	P4
6	P1

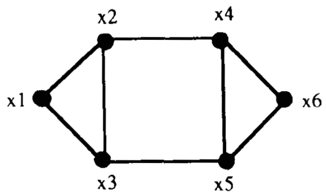

Fig. 14. Topology of DCS for 8-set of data file distributions.

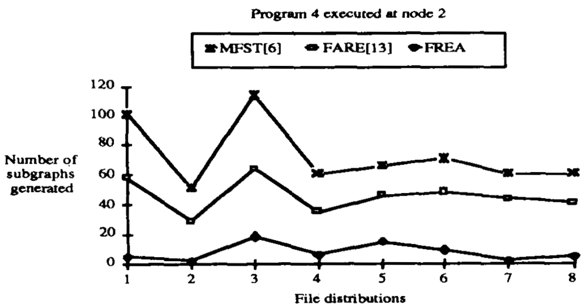

Fig. 15. Number of subgraphs versus different data file distributions.

TABLE VII
DATA FILES NEEDED FOR EXECUTING A PROGRAM P_i

Programs	Files Required
P1	F1, F2, F3
P2	F2, F4, F6
P3	F1, F3, F5
P4	F1, F2, F4, F6

x_3 or x_4; P_3 can be executed at node x_3 or x_4; P_4 can be executed at node x_2 or x_5. Table IX shows the number of subgraphs generated and the DPR of the same program to be executed from more than one node of the example in Fig. 17. FARE [13] is not applicable for distributed programs running at more than one node.

It should be noted that the current FARE algorithm [13] cannot compute DPR of the same program executed from more than one site.

TABLE VIII
DATA FILE DISTRIBUTIONS USED FOR COMPARISON

Set Files	Set 1 (nodes)	Set 2 (nodes)	Set 3 (nodes)	Set 4 (nodes)	Set 5 (nodes)	Set 6 (nodes)	Set 7 (nodes)	Set 8 (nodes)
F_1	2, 4, 5	2, 3, 6	4, 5, 6	1, 2, 3	1, 4, 6	1, 3, 6	3, 4, 5	2, 3, 6
F_2	4, 5	3, 5	2, 3	4, 5	2, 5	3, 6	1, 2	3, 5
F_3	5, 6	3, 4	4, 5	1, 6	3, 4	1, 2	5, 6	1, 6
F_4	3, 4	2, 3	1, 3	2, 4	2, 5	4, 5	5, 6	2, 6
F_5	4, 6	4, 5	4, 5	2, 4	3, 5	4, 6	1, 6	3, 6
F_6	6	3	6	3	5	5	5	4

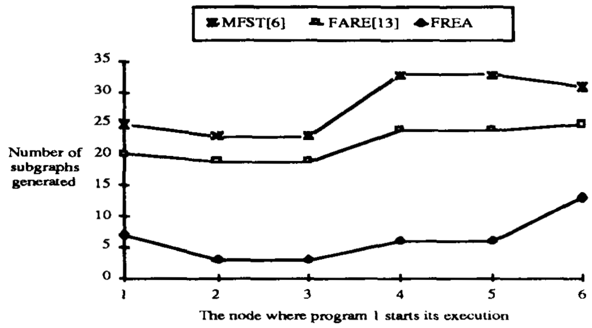

Fig. 16. Number of subgraphs versus different program distributions.

TABLE IX
NUMBER OF SUBGRAPHS GENERATED AND DPR FOR EXAMPLE OF FIG. 17

Program / Algorithm	P_1	P_2	P_3	P_4
MFST [6]	42	98	58	103
FARE [13]	—	—	—	—
FREA	30	22	27	58
DPR	0.9995076	0.9976697	0.9997831	0.9976616

F. Actual Execution Time Comparison

Generally, an algorithm with less subgraphs generated during the DPR analysis will have better execution efficiency since the execution time required for the algorithm to analyze the reliability is dominated by the expanding steps (the recursive part) to generate subgraphs. When fewer subgraphs are generated during the analysis, it implies that the size of the original graph has been reduced before subgraph generation. Certainly, we expect that it will take less time to analyze a smaller graph. The time spent by reliability preserving reduction routines incorporated in the FREA algorithm is less significant than the subgraph expansion (the recursive part) which could grow exponentially. To support this observation, we provide some actual execution time comparisons among these algorithms. The compared algorithms are all implemented using the C program under the same hardware and software environments. The following execution results are the analysis of the distributed programs 1 to 10 in the ARPA network (Fig. 11) under the IBM RISC/6000 workstation. It is clear that the proposed FREA algorithm outperforms existing algorithms in execution of any of these distributed programs.

VI. CONCLUSION

The 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

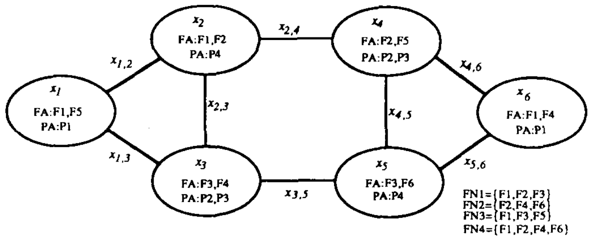

Fig. 17. Example of the same program executed at more than one site.

TABLE X
EXECUTION TIME (IN SECONDS) BY DIFFERENT ALGORITHMS FOR DISTRIBUTED PROGRAMS 1 TO 10 IN ARPA NETWORK UNDER IBM RISC/6000 WORKSTATION

Program Algorithm	P_1	P_2	P_3	P_4	P_5
MFST [6]	58.22	275.33	1462.69	>1800	15.59
FARE [13]	4.08	2.93	7.29	7.75	0.78
FREA	1.44	0.27	0.28	0.68	0.24
DPR	0.9708450	0.9739356	0.9766832	0.9345704	0.9847566
Program Algorithm	P_6	P_7	P_8	P_9	P_10
MFST [6]	93.31	474.28	104.00	246.17	>1800
FARE [13]	2.27	5.11	2.39	4.56	13.4
FREA	0.28	0.07	0.43	0.2	0.77
DPR	0.9334858	0.9143801	0.9821738	0.9703900	0.9695497

indexes such as source-to-terminal [7], survivability [8], multiterminal reliability [10], and K-terminal reliability [11] are not directly applicable for the analysis of the distributed reliability property in DCS without appropriate modification. Thus, new approaches and algorithms for the reliability analysis of the DCS must be developed.

In this correspondence, we propose an algorithm, called the Fast Reliability Evaluation Algorithm (FREA), based on the generalized factoring theorem by employing several reliability preserving reductions to speed up the reliability evaluation process. The use of the generalized factoring theorem implies that all subgraphs generated will be completely disjoint and, therefore, no replicated trees will be generated. The use of various reliability preserving reduction techniques implies that the size of the graph will be reduced and, therefore, less subgraphs will be generated. Compared with existing algorithms on various network topologies, file distributions, and program distributions, the FREA algorithm is much more economical in both time and space. This claim can also be supported by the actual execution time analysis reported in Section V-F. The feasibility of the proposed algorithm for distributed program reliability and distributed system reliability analyses can easily be confirmed by analysis on the ARPA computer network. The current FREA algorithm assumes that all nodes are perfect in its current analysis. For an imperfect node case, a slightly modified FREA algorithm can be used to generate all minimum file spanning trees, and then SYREL or a similar reliability package is called for the reliability evaluation. The more detailed treatment is reported in [21]. Also, the effect from task migration on the distributed program reliability is an important research issue, which we will study in the future.

REFERENCES

[1] D. P. Agrawal, *Advanced Computer Architecture.* Tutorial Text, IEEE Computer Society.
[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, Sept. 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, Sept. 1982.
[6] V. K. Prasnna Kumar, S. Hariri, and C. S. Raghavendra, "Distributed program reliability analysis," *IEEE Trans. Software Eng.,* vol. SE-12, no. 1, pp. 42–50, Jan. 1986.
[7] A. Satyanarayna and J. N. Hagstrom, "New Algorithm for Reliability Analysis of Multiterminal Networks," *IEEE Trans. Reliability,* vol. R-30, pp. 325–333, Oct. 1981.
[8] R. E. Merwin and M. Mirherkerk, "Derivation and use of survivability criterion for DDP systems," in *Proc. 1980 Nat. Comput. Conf.,* May 1980, pp. 139–146.
[9] K. K. Aggrawal and S. Rai, "Reliability evaluation in computer-communication networks," *IEEE Trans. Reliability,* vol. R-30, pp. 32–35, Apr. 1981.
[10] A. Grnarov and M. Gerla, "Multiterminal reliability analysis of distributed processing system," in *Proc. 1981 Int. Conf. Parallel Processing,* Aug. 1986, pp. 79–86.
[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,* 1988, pp. 851–860.
[14] A. Kumar, S. Rai, and D. P. Agrawal, "On computer communication network reliability under program execution constraints," *IEEE J. Selected Areas Commun.,* vol. 6, no. 8, pp. 1393–1399, Oct. 1988.
[15] F. Moskowitz, "The analysis of redundancy networks," *AIEE Trans. (Commun. Electron.),* vol. 29, pp. 627–632, 1958.
[16] M. O. Ball, "Computing network reliability," *Opt. Res.,* vol. 27, pp. 132–143.
[17] M. K. Chang, "A graph theoretic appraisal of the complexity of network reliability algorithms," Ph.D. dissertation, Dept. of IEOR, Univ. of California, Berkeley, 1981.
[18] A. Satyanarayana and M. K. Chang, "Network reliability and the factoring theorem," *Networks,* vol. 13, pp. 107–120, 1983.
[19] R. K. Wood, "A factoring algorithm using polygon-to-chain reductions for computing K-terminal network reliability," *Networks,* vol. 15, pp. 173–190, 1985.
[20] C. S. Raghavendra, V. K. Prasnna Kumar, and S. Hariri, "Reliability analysis in distributed system," *IEEE Trans. Comput.,* vol. 37, pp. 352–358, Mar. 1988.
[21] D. J. Chen, "On the reliability analysis of the distributed computing system," Comput. Sci. Inform. Engineering, National Chiao Tung Univ., China, Tech. Rep. CSI-1991-005, July, 1991.