Reliability Analysis of Distributed Systems Based on a Fast Reliability Algorithm

Deng-Jyi Chen, *Member, IEEE*, and Tien-Hsiang Huang

Abstract—**The reliability of distributed processing system (DPS) can be expressed by the analysis of distributed program reliability (DPR) and distributed system reliability (DSR). One of the good approaches to formulate these reliability performance indexes is to generate all disjoint File Spanning Trees (FST's) in the DPS graph such that the DPR and DSR can be expressed by the probability that at least one of these FST's is working. In this paper, we present a unified algorithm to efficiently generate disjoint FST's by cutting different links and compute the DPR and DSR based on a simple and consistent union operation on the probability space of the FST's. The DPS reliability related problems are also discussed in this paper. These include 1) the reliability of more than one copy of programs running on a given DPS starting from different sites, 2) the reliability of a specified program running on a given DPS starting from different sites, 3) the reliability of more than one different programs running on a given DPS, and so on. For speeding up the reliability evaluation, nodes merged, series, and parallel reduction concepts are incorporated in the algorithm. Based on the comparison of number of subgraphs (or FST's) generated by the proposed algorithm and by existing evaluation algorithms, we conclude that the proposed algorithm is much more economic in terms of time and space than the existing algorithms.**

Index Terms— **Distributed program reliability (DPR), distributed system reliability (DSR), graph theory, file spanning tree (FST), spanning tree, reliability.**

I. Introduction

DISTRIBUTED Processing Systems (DPS) provide cost-effective ways for improving computer system's resource sharing, performance, throughput, fault tolerance, and reliability [3]–[5], [8], [11], [15], [18]. Both the reliability performance and the fault tolerance are significantly affected by its redundant resources (i.e., the distribution of data files) and cooperation among processing elements. In general, a distributed program usually requires one or more of the resources (e.g., processing elements, data files, and so on) for successful execution. Thus, the operability of the communication link and the availability of the required data files all play a role in affecting the reliability of a given program running in the DPS. Also, different data files distribution could result in different reliability performance for a given distributed program as well as the overall system. To develop an effective approach for the reliability analysis of distributed system has become an important topic.

Traditional reliability evaluation approaches for computer networks such as source-to-terminal [1], [2], [9], [10], [16], [17], Boolean algebra method [6], [7] are not directly applicable for distributed systems for that the effects of redundant data files and programs are not captured in these methods. To overcome these problems, new approaches for the reliability analysis of distributed programs must be developed.

In reliability analysis of the DPS, Kumar proposed a very good notion called Minimal File Spanning Tree (MFST) and develop algorithms to find MFST's within a DPS graph [12]–[14]. The DPR and DSR can then be computed through the analysis of the probability that at least one of these MFST's is working. The MFST algorithm presented in [12] uses breadth-first search method to travel the DPS graph to generate all MFST's. According to the algorithm, firstly, all MFST's of size 0 are determined; next, all MFST's of size 1 are determined; and so on. This procedure is repeated for all possible sizes of MFST's up to $n-1$, where n is the number of nodes in the underlying DPS. Both the DPR and DSR can then be determined by computing the probability that at least one of the MFST is working.

In this paper, we present a unified reliability algorithm to efficiently compute the DPR and DSR. The concept of the algorithm is based on a special graph cutting approach to generate the probability subgraphs that represent the FST's in the DPS graph. The DPR and DSR computation can then be done by unioning all these disjoint probability subgraphs representing FST's. This approach guarantees no replicated FST's to be generated during the FST generating process. To speed the reliability evaluation process, node merged, series, and parallel reduction concepts are incorporated into the proposed algorithm. Although the performance of the reliability evaluation algorithm is affected by the data file distribution, the proposed reliability evaluation algorithm is much faster compared with algorithms presented in [12]–[14].

II. Notations, Definitions, and Problem Statements

Notations and definitions used in the rest of the paper are summarized here for easy reference.

A. Notations

x_i: a node representing a processing element i.
$x_{i,j}$: a link between processing elements i and j.
$p_{i,j}$ ($q_{i,j}$): probability that the link $x_{i,j}$ is up (down).

Manuscript received October 27, 1989; revised May 21, 1991. This work was supported in part by the National Science Council under Contract NSC 80-0408-E-009-16 and in part by the Chung-Shan Institute of Science and Technology under Contract CS79-0210-D009-03, Taiwan, R.O.C.

The authors are with the Computer Science and Information Engineering Department, National Chiao Tung University, Hsin Chu, Taiwan, R.O.C. 30050

IEEE Log Number 9105521.

t: a subgraph, that can be a tree or forests, of DPS's graph. The trees and forests are represented by sets of nodes and links.

FA_i: the set of data files available at processing element x_i.

FA_t: the set of data files available for subgraph t.

FN_i: the set of data files needed to execute program i.

FN: the set of data files needed for current programs to be executed in the DPS.

PV: the set of current programs which need to be executed in the DPS.

PRG_i: the set of programs available at processing element x_i.

PA_t: the set of programs available for subgraph t.

LS_t: a set of link states that represents the links' conditions in the current subgraph t.

The state conditions of each these links can be one of the followings:

$$\begin{cases} 0 & x_{i,j} \text{ is down} \\ 1 & x_{i,j} \text{ is up} \\ * & \text{don't care} \end{cases} \tag{1}$$

ST_t: a set of link states that can be used to construct the Spanning Tree of subgraph t.

NC_t: a set of link states that indicate which link cannot be cut in subgraph t.

LSP_t: a set of link states that can be used to compute the probability of subgraph t. The state condition could be either 1, 0, or *.

B. Definitions

Definition 1: An *FST* is a File 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.

Definition 2: An *MFST* is a minimal FST such that there exists no other FST which is subset of it.

Definition 3: Operator $\wedge$ represents the logic AND operation, and its applicable operations are listed below.

$$\begin{array}{lll} 0 \wedge 0 \rightarrow 0 & 1 \wedge 0 \rightarrow 0 & * \wedge 0 \rightarrow 0 \\ 0 \wedge 1 \rightarrow 0 & 1 \wedge 1 \rightarrow 1 & * \wedge 1 \rightarrow 1 \\ 0 \wedge * \rightarrow 0 & 1 \wedge * \rightarrow 1 & * \wedge * \rightarrow *. \end{array}$$

Definition 4: Operator $\vee$ represents the UNION operation and it is applied on LSP_t only for the summation of the probability space of subgraph t. The operation is $0 \vee 1 \rightarrow *$ under the condition of two LSP_t's only differ in one bit, e.g.,

$$LSP_{t1} = **011$$
$$LSP_{t2} = **001$$
$$LSP_{t1} \vee LSP_{t2} = **011 \vee **001 = **0*1.$$

Definition 5: A *probability graph* is a graph that has a probability space associated with it. For the original graph, the probability space is assumed to be 1. Also, the probability space of a subgraph will be equal to the sum of the probability space of all subgraphs generated by this subgraph.

Definition 6: A DPS graph is a graph representing the distributed processing system. In the rest of the paper, we also use the FST to represent a probability graph. Thus, they are used interchangeably.

C. The Problem Statements

Consider the distributed processing 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 ($F1$ and $F2$) and can run program 1 directly from here to communicate with other nodes for accessing data files required to complete the execution of program 1. The detailed information of each node is summarized in FA_j, FN_j, and $PRG_j (j = 1, 2, \cdots, 4)$.

We are interested in solving the following reliability performance problems:

1) What will be the reliability of a single distributed program 1, 2, or 3 running under the given DPS?
2) What will be the reliability of two or more distributed programs successfully running under a given DPS?
3) For the same program i, what will be the reliability of one program i, two copies of program i, three copies of program i, and so on running on a given DPS?
4) What will be the overall system reliability while executing all the programs in a given DPS?
5) What will be the distributed program reliability if users choose a particular site to run a particular distributed program or to execute all the programs in the system?

III. The Distributed Reliability Algorithm

A. The Basic Concept of The Algorithm

The basic idea for the algorithm is to find all disjoint FST's in each size starting from the original graph representing the DPS. Consider the DPS graph representation in Fig. 1, if we use an efficient method to cut one link each time from the graph at a different place to generate possible subgraphs recursively, then we are able to predict if each of these resulting subgraphs is an FST by examining the set of data files contained in this subgraph against the set of required data files for executing the distributed programs. This process can be repeated starting from graph size $n, n-1, \cdots,$ to 0 (where n is the number of links in the graph). Obviously, without an efficient method to remove appropriate links, the time and space required for the algorithm could be very poor. The concept of the algorithm is described informally below.

1) If the current graph has contained a FST be working, then store the current graph into list FOUND. Otherwise, choose an efficient method to cut one link from the current graph such that each of these resulting subgraphs is generated by cutting different link from the current graph and store the current graph with these cutting links be working into list FOUND.
2) Check each subgraph generated by step 1 with the set of required data files for executing the distributed programs. If the subgraph meets the requirement then store it into list TRY.

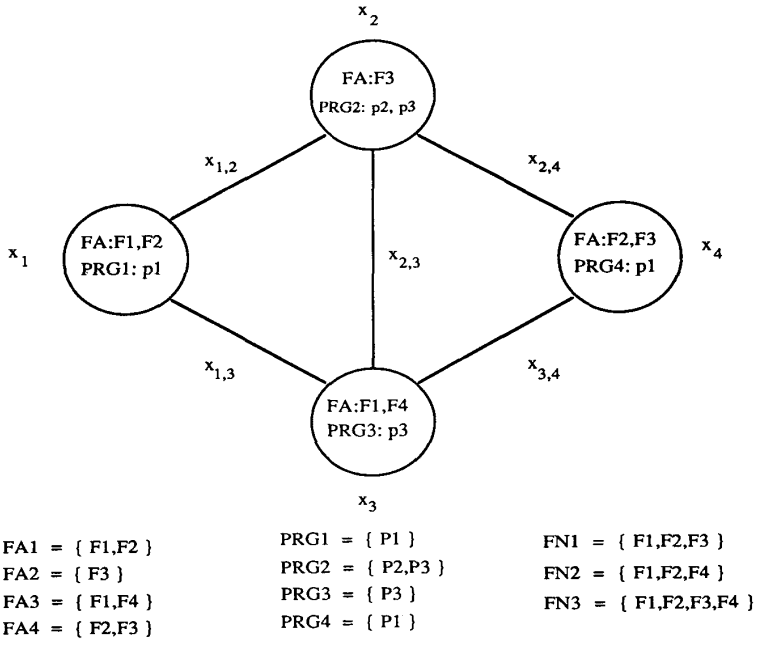


Fig. 1. A simple DPS with four processing elements

3) For each subgraph in TRY, apply steps 1 and 2 repeatedly until TRY is empty.
4) All the FST's in each size are now stored in list FOUND.

B. A Method for Graph Cutting and Reliability Computation

The method for cutting the graph plays an important role in finding the FST's and in computing the reliability of the DPS. Let us use an example to illustrate this graph cutting method. Consider the DPS graph in Fig. 1 again, we could make sure that all nodes in the DPS should be able to communicate with each other if links $x_{1,2}, x_{2,3}$, and $x_{3,4}$ are all working. This communicate path $(x_{1,2}, x_{2,3}, x_{3,4})$ is exactly a spanning tree of the original DPS graph. Let the probability space of the DPS graph be 1, then it should be equal to the sum of the following four disjoint probability subgraphs:

1) A probability subgraph that links $x_{1,2}, x_{2,3}, x_{3,4}$ are all working,
2) A probability subgraph that link $x_{1,2}$ fails,
3) A probability subgraph that link $x_{2,3}$ fails but link $x_{1,2}$ must work,
4) A probability subgraph that link $x_{3,4}$ fails but both links $x_{1,2}$ and $x_{2,3}$ must work.

This relationship can be depicted in Fig. 2. The whole circle represents the total probability space which is equal to 1. The portion A (shadowed) represents the probability space that the graph has been confirmed to be working and itself is an FST of size 5. While portions B, C, and D are needed to be determined further for finding the FST of size 4. It should be noted that the probability of portion A, B, C, and D is not graphically proportional but they should be summed up to 1.

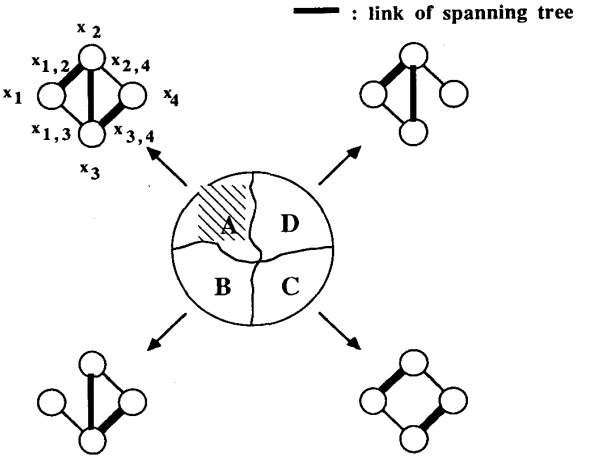


Fig. 2. The probability space of the original graph G.

What we need to do next is to determine the probability space of portion B, C, and D. Follow the same idea, we then find a spanning tree from the subgraph representing portion B. The spanning tree tells us that links $x_{1,3}, x_{2,3}$, and $x_{2,4}$ must be up in order to make sure the connection among all four nodes. Thus, the probability of portion B will be the sum of the following four disjoint probability subgraphs:

1) A probability subgraph that links $x_{1,3}, x_{2,3}, x_{2,4}$ are all working,
2) A probability subgraph that link $x_{1,3}$ fails,
3) A probability subgraph that link $x_{2,3}$ fails but link $x_{1,3}$ must work,

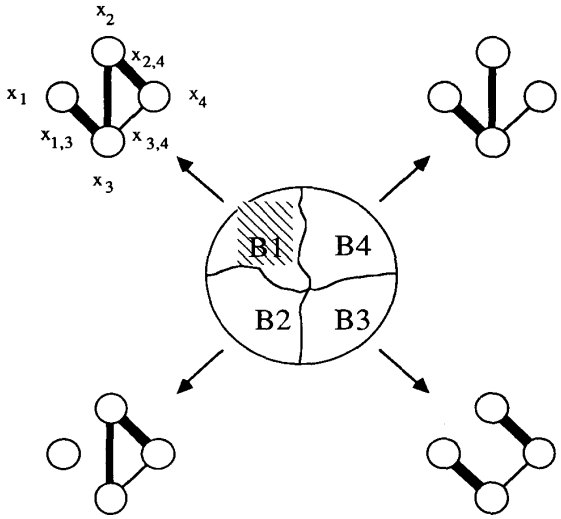


Fig. 3. The probability space of the subgraph representing portion B.

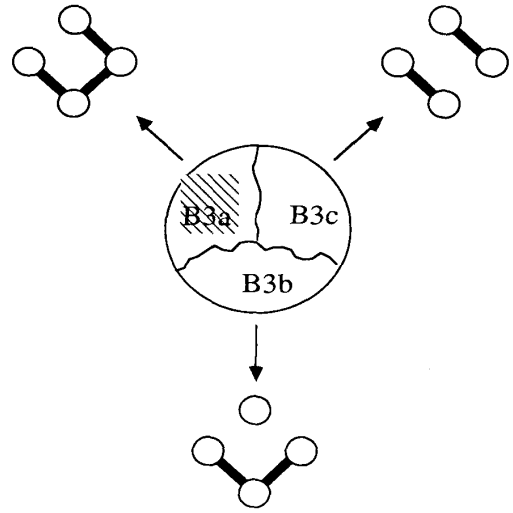


Fig. 4. The probability space of the subgraph representing portion B3.

Fig. 5. The probability space of the subgraph representing portion B3.

4) A probability subgraph that link $x_{2,4}$ fails but both links $x_{1,3}$ and $x_{2,3}$ must work.

This relationship is depicted in Fig. 3. The whole circle represents the total probability space of portion B. The portion B1 (shadowed) represents the probability space that the subgraph has been confirmed to be working and is an FST of size 4. While portions B2, B3, and B4 are needed to be determined further for finding the FST of size 3. It should be noted that the probability space of portion B1, B2, B3, and B4 is not graphically proportional but they should be summed up to the probability space of portion B (in Fig. 2).

Based on the same concept, we compute the probability subgraphs representing the portion B2, B3, and B4. For example, the probability subgraphs representing portion B3 is depicted in Fig. 4. Again, the probability subgraphs in Fig. 4 should be summed up to the probability space of portion B3. We repeat this process until either the FST of size 0 is obtained, no available links can be cut, or the subgraph does not contain the required data files for executing the distributed program. For example, Fig. 5 depicts one of the termination conditions of this process.

Once the concept of cutting the graph is understood, we need an appropriate representation scheme to illustrate. Let $s_{i,j}$ be the state of link $x_{i,j}$ of the DPS, where

$$S_{i,j} = \begin{cases} 0 & x_{i,j} \text{ is down} \\ 1 & x_{i,j} \text{ is up} \\ * & \text{don't care} \end{cases}$$

with applicable operations based on operator (defined in Section II-C), then notations LS_t, NC_t, ST_t, and LSP_t (defined in Section II-A) can be used to represent the graph cutting process. For example, for the graph of portion A in Fig. 2, suppose the bit state sequence of links are $x_{1,2}, x_{1,3}, x_{2,3}, x_{2,4}, x_{3,4}$, we represent it as

LS_A = ***** (the initial graph)

NC_A = 00000 (subgraphs can be generated by cutting any links)

ST_A = 1*1*1 (since links $x_{1,2}, x_{2,3}, x_{3,4}$ have to be working)

LSP_A = 1*1*1 (it is obtained by $LS \wedge ST$). Likewise, for the subgraphs B, C, and D can be represented as

LS_B = 0****	LS_C = 1*0**	LS_D = 1*1*0
NC_B = 00000	NC_C = 10000	NC_D = 10100
ST_B = *111*	ST_C = *****	ST_D = *****
LSP_B = 0111*	LSP_C = 1*0**	LSP_D = 1*1*0.

Since the subgraphs C and D both contain an FST be working (link $x_{1,2}$ must be working), thus we do not need to find a spanning tree in subgraphs C and D, so ST_C = ***** and ST_D = *****. NC_C = 10000 guarantee that the subgraph C is disjoint with the subgraph B, and NC_D = 10100 guarantee that the subgraph D is disjoint with the subgraph B and C. It can be inferred that $\Pr(LS) = \Pr(LSP_A) + \Pr(LS_B) + \Pr(LS_C) + \Pr(LS_D) = 1$, where $\Pr(LS_C) = \Pr(LSP_C), \Pr(LS_D) = \Pr(LSP_D)$, and $\Pr(LS)$ can be computed recursively using the graph cutting method introduced above. The informal algorithm for cutting the graph and computing the reliability can be described below.

1) Find a spanning tree from the current graph if necessary and compute ST_t,
2) Compute the vector LSP_t by $ST_t \wedge LS_t$, and convert vector LSP_t to the probability expression.
3) Cut the current graph according to the vectors ST_t and NC_t to obtain its subgraphs (or FST's),

4) Repeat 1 to 3 to compute each subgraph's vector LSP_t,
5) The reliability of the DPS graph is obtained by unioning all LSP_t-vectors that are associated with all the FST's.

C. The Complete Reliability Algorithm

Once the concept of finding all FST's and computing the reliability of the DPS is understood, we now present the complete algorithm for finding the FST's and computing the reliability of the FST.

FST Reliability Algorithm
begin
step 1: initialization
t = original graph ;
TRY = t ; (store the original graph into list TRY)
FOUND = ϕ
LS_t = **...* ;
NC_t = 00...0 ;
$FN = \cup_i FNi$ (where program $i \in PV$) ;
$R = \phi$;
step 2: generate spanning tree
repeat
2.1 get a subgraph t from TRY ;
2.2 checking step (check and find spanning tree)
remove t from TRY ;
if a FST has been working in t
then add t to FOUND ;
ST_t = **...* ;
else find a spanning tree of connected component i in t such that $FA_i \supseteq FN$ and $PA_i \supseteq PV$, and represent it by ST_t;
$LSP_t = LS_t \wedge ST_t$;
2.3: cutting step (generate subgraphs)
add subgraph(t) to TRY ;
until (TRY = ϕ)
step 3: compute reliability
for all t in FOUND do
$R = R \vee LSP_t$;
od
Reliability = Pr(R) ;

end

procedure subgraph(t)
begin
child = ϕ ;
temp = ϕ ;
find each link $x_{k,l}$ with its value in ST_t = 1 and in NC_t = 0 ;
/* ST_t represents all the links of the spanning tree, NC_t represents all the links cannot be cut */
cut = $\cup_{k,l} \{x_{k,l}\}$; /* cut store all the links can be cut */
for all $x_{i,j} \in$ cut do
$LS_{newt} = LS_t$;
set the state of $x_{i,j}$ in LS_{newt} to 0;
/* cut link $x_{i,j}$ now */
find all links $\in$ temp and set those links' states in LS_{newt} to 1 ;
temp = temp $\cup\{x_{i,j}\}$;
find a connected component i in t such that $FA_i \supseteq FN$ and $PA_i \supseteq PV$;
if there are any connected component i found
then add newt to child ;
od
return (child) ;
end (* subgraph *)

D. The FST Reliability Algorithm with Series and Parallel Reduction

How to speed the reliability evaluation process up is the major concern of the proposed algorithm. The basic principle of speeding the reliability evaluation is to generate correct FST's with less cutting steps. There are four methods that can be used interchangeably to speed the reliability evaluation. These methods are 1) nodes merged, 2) parallel reduction, 3) series reduction, and 4) degree-2 reduction.

1) *Nodes merged* occurs when a probability subgraph has bit value 1 in its LS vector. For example, probability subgraph B3 in Fig. 3 (Section III-B) with LS_{B3} = 010** indicates that node x_1 and x_3 can be merged together since all the subgraphs generated by subgraph $B3$ must contain link $x_{1,3}$. This characteristic is also indicated by the NC vector, NC_{B3} = 01 000 in Fig. 3, which tells that link $x_{1,3}$ cannot be cut to obtain its subgraphs. That is the reason why one can get disjoint FST's using this graph cutting technique.
2) *Parallel reduction* occurs when a probability subgraph contains two or more links between two nodes. With connectivity property, these redundant links can be reduced to one link.
3) *Series reduction* occurs when a probability subgraph has a node, with node degree = 2, that contains no data file required for executing the distributed program. Since such a node, after deletion, still does not affect the correct FST generation, we can remove this node and reduce two links that connect to its neighboring nodes into one link.
4) *degree-2 reduction* occurs when a probability subgraph has a node, with node degree = 2, that is not a leaf node of any MFST's of the current graph. Since this node is not a leaf node of any MFST's, then the two adjacent links of this node must be working or fail simultaneously, thus we can remove this node and reduce two links that connect to its neighboring nodes into one link, and copy the data files and programs in this node to its two neighboring nodes.

If node x_n satisfies the following checking procedures, then it is defined as a degree-2 reducible node and it will not be a leaf node of any MFST's.

The procedures of checking if node x_n is a degree-2 reducible node:

1) delete link $x_{n,i}$, where node x_i is a neighbor node of node x_n.

2) get a data file Fk from node x_n, delete all nodes in the DPS that contains data file Fk
3) try to find a FST starting from node x_n, if there is a FST be found, then node x_n is not a degree-2 reducible node and stop these checking procedures.
4) repeat step 2 to step 3 until each of the data files in node x_n is considered.
5) get a program Pk from node x_n, delete all nodes in the DPS that contain program Pk.
6) try to find an FST starting from node x_n, if there is an FST be found, then node x_n is not a degree-2 reducible node and stop these checking procedures.
7) repeat step 5 to step 6 until each of the programs in node x_n is checked.
8) recover link $x_{n,i}$ and delete the other link $x_{n,j}$, where node x_j is another neighbor node of node x_n.
9) repeat step 2 to step 7
10) If the checking procedures is not stopped in the middle then x_n is a degree-2 reducible node.

In the following, we use an example to illustrate these reduction methods. Suppose there is a subgraph generated as shown in Fig. 6(a), we need to compute the reliability of program 1 which requires data files $F1$, $F2$, $F3$, and $F4$ for completing its execution. Suppose the bit state sequence of links are $x_{1,2}, x_{1,3}, x_{2,3}, x_{2,4}, x_{3,5}, x_{4,5}$, and the following reduction steps are applicable for speedup the correct FST generation.

Step 1: Since link $x_{1,2}$ can no longer be cut and must be up for the rest of its subgraph generation (LS = 1**0**), we apply nodes merged reduction on nodes x_1 and x_2. The resulting subgraph(b) is shown in Fig. 6(b).

Step 2: A parallel reduction can be applied on the resulting subgraph (from step 1) since links $x_{1,3}$ and $x_{2,3}$ are parallel. The new resulting subgraph(c) is shown in Fig. 6(b).

Step 3: A series reduction occurs since node x_5 contains no data files for the execution of program 1. The new resulting subgraph(d) is shown in Fig. 6(b).

Step 4: A degree-2 reduction occurs since node x_3 is not a leaf node of any MFST's. The final subgraph(e) after these reductions is also shown in Fig. 6(b).

Once these reduction techniques are understood, we need new representations for computing ST_t and LS_{newt}. These new representing schemes are discussed below.

1) ST_t and LS_{newt} representation in *parallel reduction*: Let two links $x_{i,j}$ and $x_{k,l}$ in graph t be parallelly reduced into $x'_{i,j}, x'_{i,j}$ itself be one of the links in the spanning tree we found, and the bit state sequence of both ST_t and LS_{newt} be $(\cdots x_{i,j} x_{k,l} \cdots)$, then
   a) the state of ST_t after reduction will be $(\ldots 11 \cdots) \vee (\ldots 10 \ldots) \vee (\ldots 01 \ldots)$, and
   b) the state of LS_{newt} (newt is a child of t) after reduction will be $(\ldots 00 \ldots)$.

   This can be seen from the fact that $x'_{i,j} = x_{i,j} x_{k,l}$. It implies that if link $x'_{i,j}$ is working then either link $x_{i,j}$ or $x_{k,l}$ must be working, and if link $x'_{i,j}$ fails then both links $x_{i,j}$ and $x_{k,l}$ must have failed.
2) ST_t and LS_{newt} representation in series reduction or in degree-2 reduction: Let two links $x_{i,j}$ and $x_{k,l}$ in graph t be series reduced or be degree-2 reduced into $x'_{i,j}, x'_{i,j}$ itself be one of the links in the spanning tree we found, and the bit state sequence of both ST_t and LS_{newt} be $(\ldots x_{i,j} x_{k,l} \ldots)$, then
   a) the state of ST_t after reduction will be $(\ldots 11 \ldots)$.
   b) the state of LS_{newt} (new_t is a child of t) after reduction will be $(\ldots 00 \ldots) \vee (\ldots 10 \ldots) \vee (\ldots 01 \ldots)$.

   This can be seen from the fact that $x'_{i,j} = x_{i,j} \cap x_{k,l}$. It also implies that when link $x'_{i,j}$ is working then both links $x_{i,j}$ and $x_{k,l}$ must be working, and if link $x'_{i,j}$ fails then either link $x_{i,j}$ or $x_{k,l}$ must be failed.
3) ST_t and LS_{newt} representation in the combination of both series and parallel reduction or degree-2 and parallel reduction: There are several combination reduction cases that may occur during the FST generation. For example, links $x_{i,j}$ and $x_{k,j}$, firstly, are parallelly reduced into $x'_{i,j}$, and then $x'_{i,j}$ and $x_{m,n}$ are series or degree-2 reduced into $x''_{i,j}$. Let the bit state sequence of both ST_t and LS_{newt} be $(\ldots x_{i,j} x_{k,l} x_{m,n} \ldots)$, then
   a) the state of ST_t after reduction will be $(\ldots 111 \ldots) \vee (\ldots 101 \ldots) \vee (\ldots 011 \ldots)$.
   b) the state of LS_{newt} (newt is a child of t) after reduction will be $(\ldots 001 \ldots) \vee (\ldots 110 \ldots) \vee (\ldots 100 \ldots) \vee \vee (\ldots 010 \ldots)$.

   This can be seen from the fact that $x'_{i,j} = (x_{i,j} \cup x_{k,l}) \cap x_{m,n}$. Other combination cases can be represented in the same way.

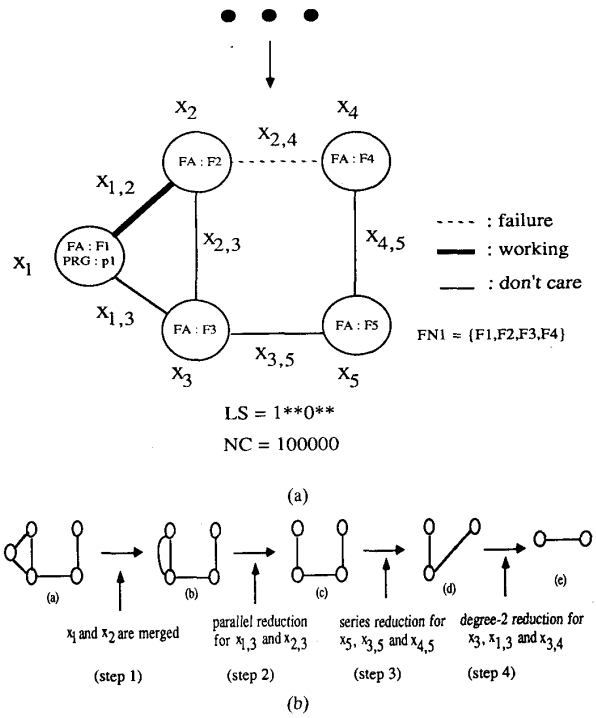


Fig. 6. (a) A subgraph during reliability evaluation process. (b) Reduction for subgraph of (a).

The FST reliability algorithm can be incorporated with these techniques to speed up the reliability evaluation. The complete FST reliability algorithm with series, parallel, and degree-2 reduction is listed below.

FST_SPR algorithm
begin
step 1: initialization
t = original graph;
TRY = $\{t\}$ (store the original graph into list TRY)
FOUND = ϕ
LS_t = ** . . . *;
NC_t = 00 . . . 0;
FN = $\cup_i FNi$ (where program $i \in PV$) ;
R = ϕ ;
step 2: generate spanning tree
repeat
2.1 get a subgraph t from TRY ;
2.2 reduction step
repeat
series_reduction(t) ;
degree-2_reduction(t) ;
parallel_reduction(t) ;
until no reduction occurs
2.3 checking step
remove t from TRY;
if an FST has been working in t
then add t to FOUND;
ST_t = ** . . . *
else find a spanning tree of connected component i in t such that
$FA_i \supseteq FN$ and $PA_i \supseteq PV$, and represent it by ST_t
with the new representation discussed in Section III-D ;
$LSP_t = LS_t \wedge ST_t$;
2.4 cutting step
add subgraph(t) to TRY ;
until (TRY = ϕ)
step 3: compute reliability
for all t in FOUND do
$R = R \vee LSP_t$;
od
Reliability = Pr(R);
end

procedure subgraph(t)
begin
child = ϕ ;
temp = ϕ ;
find each $x_{k,l}$ where $x_{k,l} \in$ stree and the state of $x_{k,l}$ in NC_t = 0
cut = $\cup_{k,l}\{x_{k,l}\}$; /* cut store all the links can be cut */
for all $x_{i,j}$ cut do
$LS_{newt} = LS_t$;
set the state of $x_{i,j}$ in LS_{newt} to 0; /* cut link$x_{i,j}$ now */
find all links which temp and modify LS_{newt} with these links by new representation discussed in Section III-D;
merge(newt,temp) ;
temp = temp $\cup\{x_{i,j}\}$;
find a connected component i in t such that $FA_i \supseteq FN$ and $PA_i \supseteq PV$;
if there are any connected component i found

```
                then add newt to child;
      od
     return (child);
end (* subgraph *)

procedure series_reduction(t)
begin
      for all node x_i in t do
            if degree of x_i = 2
               then if (FA_i ∩ FN = φ ) AND (PRG_i ∩ PV = φ )
                           then delete node x_i and link x_{i,j} (x_j is one neighbor of x_i) from t;
                                if x_{k,j} exists
                                  then rename x_{k,i} to x'_{k,j};
                                   else rename x_{k,i} to x_{k,j};
                                 record x_{k,j} (or x'_{k,j}) be series reduced by x_{k,i} and x_{i,j};
      od
end (* series_reduction *)

procedure parallel_reduction(t)
begin
   for all node x_i in t do
       for all x_i's neighbors x_j do
           if there are two links x_{i,j} and x'_{i,j} between x_i and x_j ;
              then record parallel_reduction on link x_{i,j} and x'_{i,j} ;
                   delete x'_{i,j} from t;
       od
   od
end (* parallel_reduction *)

procedure degree-2_reduction(t)
begin
      for all node x_i in t do
          if deg(x_i) = 2
             then for all the adjacent link x_{i,j} of node x_i do
                        delete link x_{i,j};
                        repeat.
                           get a data file Fk from FA_i or a program Pk from PRG_i, delete
                           all nodes that contain data file Fk or Pk in subgraph t;
                           try to find a FST starting from node x_i,
                           if there is a FST be found,
                               then node x_i is not a degree-2 reducible node and break;
                         until all Fk or Pk reside in node x_i has been considered.
                          recover link x_{i,j};
                od
             if node x_i has not been determined is a degree-2 reducible node or not
                 then delete node x_i and link x_{i,j} (x_j is one neighbor of x_i ) from t;
                      if x_{k,j} exists
                          then rename x_{k,i} to x'_{k,j};
                           else rename x_{k,i} to x_{k,j} ;
                      p_{k,j} (or p'_{k,j} ) = p_{k,i} * p_{i,j} ;
                      FA_k = FA_k ∪ FA_i; PRG_k = PRG_k ∪ PRG_i ;
                      FA_j = FA_j ∪ FA_i ; PRG_j = PRG_j ∪ PRG_i ;
      od
end

procedure nodes_merged(t,temp)
begin
   for all link x_{i,j} whose state in temp is 1 do
```

$FA_i = FA_i \cup FA_j$;
$\text{PRG}_i = \text{PRG}_i \wedge \text{PRG}_j$;
delete node x_j and link $x_{i,j}$;
parallel_reduction(t) ;
od
end (* nodes_merged *)

IV. Reliability Analysis of DPS Using the FST and FST_SPR Reliability Algorithms

A. Examples

In this section we use the FST and FST_SPR reliability algorithms to evaluate some distributed processing systems.

Example 1: Consider the simple DPS example in Fig. 1 again, we use the FST reliability algorithm to analyze the DPR1. We use the Distributed Program Reliability i (DPR_i) to describe the reliability of the distributed program i running under the DPS and Distributed System Reliability (DSR) to describe the overall reliability of the distributed system.

The splitting snapshot of subgraphs generated by the FST reliability algorithm is illustrated in Fig. 7.

To compute the reliability, we simply sum all the disjoint terms represented by vectors LSP_t. Let $\Pr(i)$ be the probability subgraph i, then

$\Pr(A) = \Pr(1*1*1)$
$\Pr(B) = \Pr(0111*) + \Pr(1) + \Pr(1c)$
$= \Pr(0111* \vee 0110* \vee \underline{01001 \vee 01011} \vee 00\ 101 \vee 0011* \vee 00\ 011 \vee 00\ 001)$
$= \Pr(0111* \vee 0110* \vee 010*1 \vee 00\ 101 \vee 0011* \vee \underline{00011 \vee 00001})$
$= Pr(\underline{0111*} \vee \underline{0110*} \vee 010*1 \vee 00\ 101 \vee 0011* \vee 000*1)$
$= \Pr(011** \vee \underline{010*1} \vee 00\ 101 \vee 0011* \vee 000*1)$
$= \Pr(011** \vee 0*0*1 \vee 00\ 101 \vee 0011*)$
$\Pr(C) = \Pr(1*0**)$
$\Pr(D) = \Pr(1*1*0)$
$\text{DPR1} = \Pr(A) + \Pr(B) + \Pr(C) + \Pr(D)$
$= \Pr(\underline{(1*1*1)} \vee (011** \vee 0*0*1 \vee 00\ 101 \vee 0011*) \vee (1*0**) \vee \underline{(1*1*0)})$
$= \Pr(\underline{1*1**} \vee 011** \vee 0*0*1 \vee 00\ 101 \vee 0011* \vee 1*0**)$
$= \Pr(1**** 011** 0*0*1\ 00\ 101\ 0011*)$
$= p_{1,2} + q_{1,2}p_{1,3}p_{2,3} + q_{1,2}q_{2,3}p_{3,4} + q_{1,2}q_{1,3}p_{2,3}p_{2,4}$
$+ q_{1,2}q_{1,3}p_{2,3}q_{2,4}p_{3,4} + q_{1,2}q_{1,3}p_{2,3}p_{2,4}$

If we assume all the links have the same reliability 0.9, then DPR_1 is equal to 0.99891.

For evaluating the DSR, the FN = {F1, F2, F3, F4}. Applying the same algorithm, we obtain $DSR = p_{1,2}p_{2,3} + p_{1,2}p_{1,3}q_{2,3} + p_{1,2}q_{1,3}q_{2,3}p_{2,4}p_{3,4} + q_{1,2}p_{1,3}p_{2,3} + q_{1,2}q_{2,3}p_{2,4}p_{3,4} + q_{1,2}q_{1,3}p_{2,3}q_{2,4}p_{3,4} + q_{1,2}q_{1,3}p_{2,3}p_{2,4}$. Again, if we assume all the links have the same reliability 0.9, then the DSR is equal to 0.9963. These results are correctly matched with the results in [12] and [13].

Example 2: Considering the DPS graph in Fig. 8(a), we will use the FST-SPR reliability algorithm to compute the DPR3. The splitting subgraphs generated by the FST-SPR reliability algorithm is illustrated in Fig. 8(b).

$\text{DPR}_3 = \Pr(\ 11**1 \vee 011*1 \vee *01*1 \vee 0*011 \vee 10011 \vee 11*10 \vee *0110 \vee 01110)$
$= p_{2,3}p_{2,4} + p_{2,3}q_{2,4}p_{3,4} + q_{2,3}p_{2,4}p_{3,4} + p_{1,2}p_{1,3}q_{2,3}q_{2,4}p_{3,4} + p_{1,2}p_{1,3}q_{2,3}p_{2,4}q_{3,4} = 0.98658$ (with the assumption of reliability 0.9 for all links)

Example 3: Considering the DPS shown in Fig. 9, it contains six processing elements and the detailed information of each node is given in sets of FA_j, PRGj, and FN_j.

Again, we want to compute the DPR_1 and DSR of the DPS. Applying the same reliability program, we get DPR_1 = 0.9995076 and DSR = 0.9975515 under the assumption of all links with reliability 0.9

B. Other Evaluation Results

Reliability problems listed in Section II-C are addressed here. Fig. 10 presents the single distributed program reliability (DPR_1, DPR_2, DPR_3, and DPR_4) running under the DPS in Fig 9. This analysis can help us to understand the reliability of each program under a certain data files distribution such that an optimal data files distribution for balancing each program's reliability can be validated.

Fig. 11 presents the reliability of two or more distributed programs running together under the DPS in Fig. 9. This analysis will help us to assign programs into different processing elements. An intended programs assignment to achieve higher reliability can be validated through this analysis.

Fig. 12 presents the reliability of one or more copies of the program 1 running together starting from different sites. The resulting reliability is under the DPS graph in Fig 9. This analysis will help us to make sure if the program under execution has reached the required reliability.

Fig. 13 presents the DPR1 under the choices of different user sites. This analysis will help users to choose a better site to execute a certain program.

C. The Correctness and Time Complexity of the Algorithm

Theorem 1: Given a graph $G = \{N, E\}$ where $N = \{x_1, x_2, \cdots, x_m\}$ is a set of nodes, and $E = \{L_1, L_2, \cdots, L_n\}$ is a set of links. Let $L = \{L_1, L_2, \cdots, L_{m-1}\} \in E$ be a set of links representing a spanning tree of G, then the probability space of the graph G can be expressed as the following disjoint terms.

$$\Pr(G) = p_1p_2 \cdots p_{m-1}\Pr(G_1) + q_1\Pr(G_2) + p_1q_2\Pr(G_3) + \cdots + p_1p_2 \cdots p_{m-2}q_{m-1}\Pr(G_m) \quad (1)$$

where p_i denotes the probability that L_i is working, q_i denotes the probability that L_i is failure, G_1 denotes G with $L_1, L_2, \cdots, L_{m-1}$ is working. G_2 denotes G with L_1 is failure. G_3 denotes G with L_1 is working and L_2 is failure, ... G_m denotes G with $L_1, L_2, \cdots, L_{m-2}$ is working and L_{m-1} is failure.

Proof: By factoring theorem (or conditional probability).

$$\Pr(G) = q_i\Pr(G|\bar{F}_i) + pi\Pr(G|\bar{F}_i) = qi\Pr(G_i) + p_i\Pr(G'_i)$$

where $\bar{F}_i$ denotes the event that L_i is failure, F_i denotes the event that L_i is working, G_i denotes G with L_i is failure, G'_i denotes G with L_i is working.

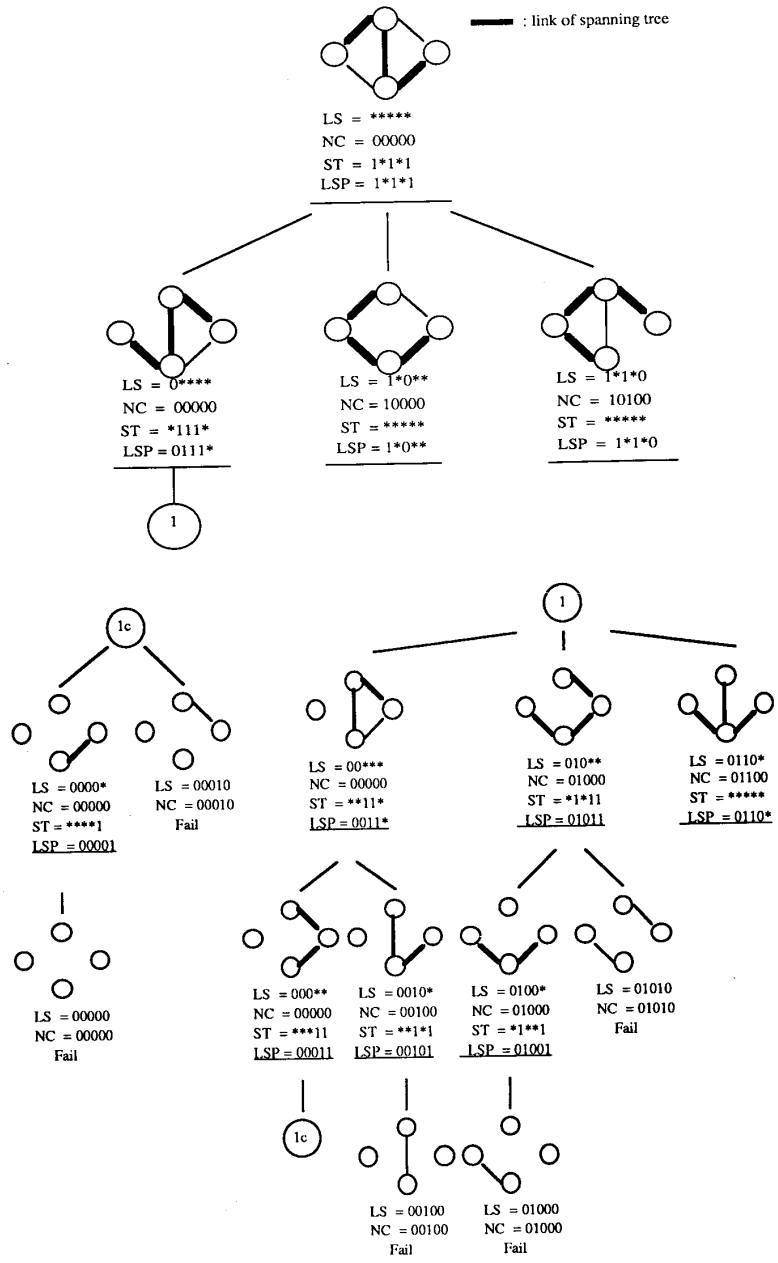


Fig. 7. The splitting snapshot of the original graph.

If we choose L_1 be the first factor, then

$$\Pr(G) = q_1 \Pr(G_2) + p_1 \Pr(G'_2).$$

Obviously, $\Pr(G'_2)$ can be expressed in the same way, thus we choose L_2 be the second factor, then

$$\Pr(G'_2) = q_2 \Pr(G_3) + p_2 \Pr(G'_3)$$

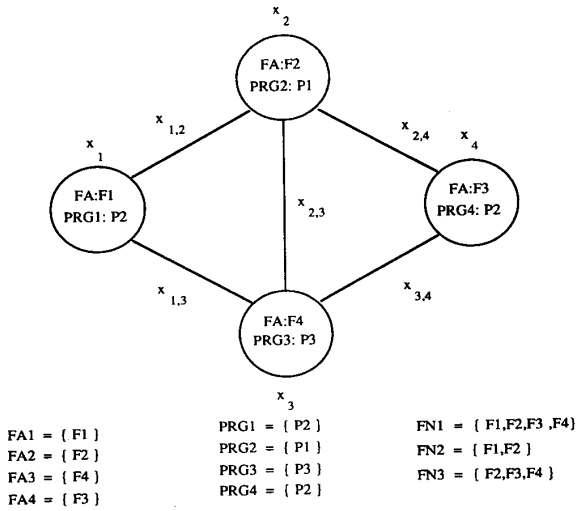
where G_3 denotes G'_2 with L_2 is failure, G'_3 denotes G'_2 with L_2 is working.

If we repeat the same action, then $\Pr(G'_{m-1})$ can be expressed as

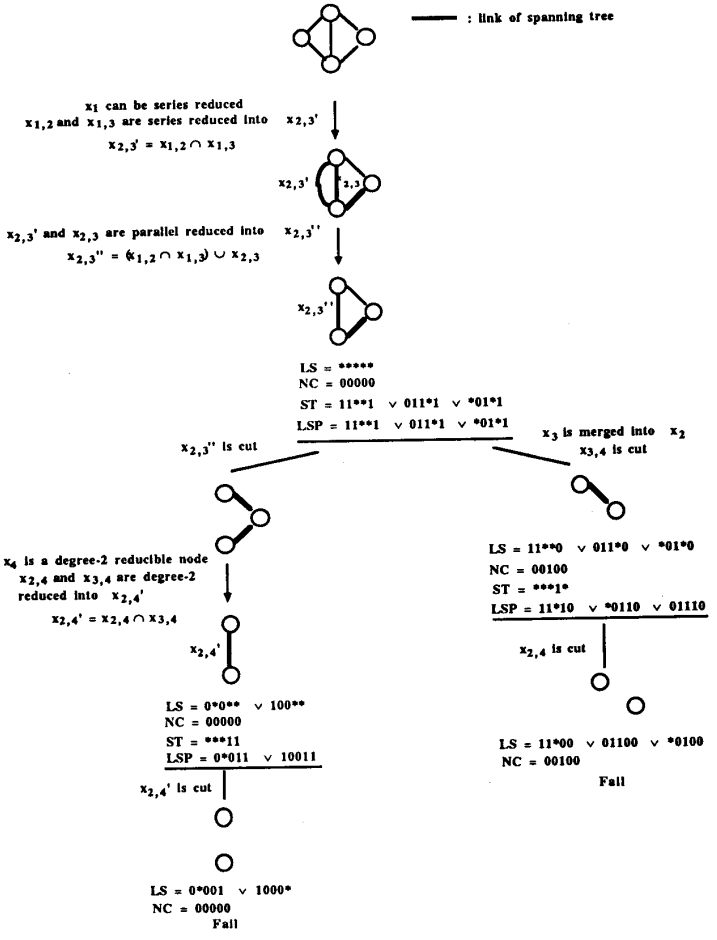
$$\Pr(G'_{m-1}) = q_{m-1} \Pr(G_m) + p_{m-1} \Pr(G'_m)$$

where G_m denotes G'_{m-1} with L_{m-1} is failure, G'_m denotes G'_{m-1} with L_{m-1} is working.

$$\begin{aligned}\Pr(G) &= q_1 \Pr(G_2) + p_1 \Pr(G'_2) \\ &= q_1 \Pr(G_2) + p_1 q_2 \Pr(G_3) + p_1 p_2 \Pr(G'_3) \\ &= q_1 \Pr(G_2) + p_1 q_2 \Pr(G_3) + p_1 p_2 q_3 \Pr(G_4)\end{aligned}$$



(a)



(b)

Fig. 8 (a) An example of DPS. (b) The splitting snapshot of example 2.

$$+ p_1p_2p_3q_4\Pr(G_5) + \cdots + p_1p_2\cdots p_{m-2}q_{m-1}\Pr(G_m) + p_1p_2\cdots p_{m-1}\Pr(G'_m) \quad (2)$$

where G'_m denotes G with $L_1, L_2, \cdots, L_{m-1}$ is working, G_2 denotes G with L_1 is failure, G_3 denotes G with L_1 is working and L_2 is failure, ... G_m denotes G with $L_1, L_2, \cdots, L_{m-2}$ is working, and L_{m-1} is failure. Let G_1 (in equation 1) = G'_m [in (2)], then (1) = (2). Thus, Theorem 1 is proved. Q.E.D.

Theorem 2: The FST reliability algorithm is correct.

Proof: To prove FST reliability algorithm correct, we transfer this problem into the one shown in Theorem 1. The novelty of the FST reliability algorithm lies in its graph cutting method. The cutting technique works exactly the same way as the enumeration of all disjoint terms in Theorem 1. Recalling the probability subgraphs in Fig. 2, the original graph is partitioned into four probability subgraphs A, B, C, and D by the cutting technique. By analogizing these probability subgraphs to the terms in Theorem 1, we found that the original probability graph represents $\Pr(G)$; subgraph A represents $p_{1,2}p_{2,3}p_{3,4}\Pr(G_A)$; subgraph B represents $q_{1,2}\Pr(G_B)$; subgraph C represents $p_{1,2}q_{2,3}\Pr(G_C)$; subgraph D represents $p_{1,2}p_{2,3}q_{3,4}\Pr(GD)$. More precisely,

$$\begin{aligned}\Pr(LS_G) &= \Pr(LSP_A) + \Pr(LS_B) + \Pr(LS_C) + \Pr(LS_D)\\ &= \Pr(1*1*1) + \Pr(0****) + \Pr(1*0**)\\ &\quad + \Pr(1*1*0)\\ &= p_{1,2}p_{2,3}p_{3,4}\Pr(G_a) + q_{1,2}\Pr(G_b)\\ &\quad + p_{1,2}q_{2,3}\Pr(G_c) + p_{1,2}p_{2,3}q_{3,4}\Pr(G_d)\\ &= \Pr(G).\end{aligned}$$

The probability of these subgraphs can be computed recursively based on the graph cutting technique again to obtain the probability of its subgraphs. The FST reliability algorithm uses such cutting technique to compute the reliability of any size and any data distribution DPS. Thus, generalized terms will be

$$\begin{aligned}\Pr(LS_G) &= \Pr(LSP_1) + \Pr(LS_2) + +\Pr(LS_3)\cdots +\\ &\qquad \Pr(LS_n)\\ &= p_{1,2}p_{1,3}p_{2,3}\cdots p_{m,n}\Pr(G_1) + q_{1,2}\Pr(G_2)\\ &\quad + q_{1,2}\Pr(G_2)p_{1,2}q_{1,3}\Pr(G_3)\\ &\quad + \cdots + p_{1,2}p_{1,3}p_{2,3}\cdots q_{m,n}\Pr(G_n)\\ &= \Pr(G).\end{aligned}$$

By such analogous, we have successfully transferred the FST reliability algorithm into the problem in Theorem 1. Since Theorem 1 is correct, therefore, Theorem 2 is also correct. Q.E.D.

Theorem 3: The FST_SPR reliability algorithm is correct.

Proof: We have shown the correctness of the FST reliability algorithm. What we need to show is the correctness of *nodes merged*, *series*, *parallel*, and *degree-2 reductions*. These reduction techniques are true intuitively. Since each nodes merged occurs when a particular link in the graph cannot be cut in the rest of its subgraphs generation, each parallel reduction occurs when a graph with two or more links are connected between two nodes, each series reduction occurs when a graph has a node with node degree = 2 contains no data files required for executing the distributed program, and each degree-2 reduction occurs when a graph has a node with node degree=2 and this node is not a leaf node of any MFST's. None of these cases has violated the disjoint property and FST definition. Thus, the FST_SPR reliability algorithm is also correct. Q.E.D.

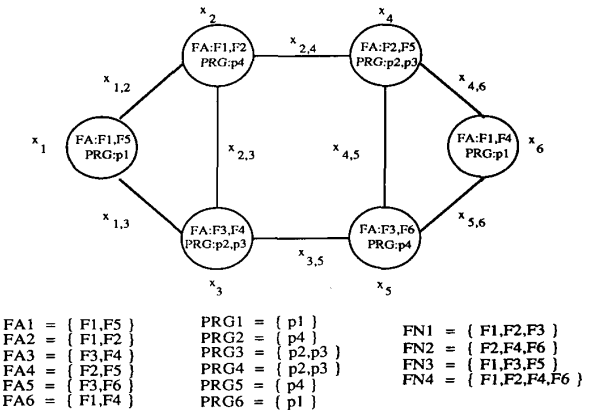


Fig. 9. A DPS with six processing elements.

Theorem 4: The FST and FST_SPR reliability algorithms guarantee no replicated FST's to be generated during the reliability evaluation.

Proof: Suppose there are two or more replicated trees generated by the FST or FST_SPR reliability algorithm, then both FST and FST_SPR algorithms will generate nondisjoint probability subgraphs. This is contradiction with Theorem 1 which generates the disjoint probability space of each term. Thus, the FST and FST_SPR algorithm guarantee no replicated FST's to be generated during the reliability evaluation.Q.E.D.

Unlike the time complexity analysis in the K-graph problem, which is statically dependent on the given k-terminal connection, the time complexity of distributed program reliability problem is dynamically bound to the data files required for each distributed program. The time complexity of the algorithms presented in [12]–[14], in worst case, can generate as many as $(n-1)^{(e-1)}$ intermediate trees, where n denotes number of nodes and e is the maximum in-degree of a node in the graph. However, in practical condition, such case never occurs since once an FST is found the tree expansion is stopped. The proposed FST algorithm uses the graph cutting technique with incorporated series and parallel reduction to speed the FST generation. The time complexity is quite difficult to quantify since the number of links and nodes may be reduced or merged during the evaluation process. However, by common reasoning, the complexity should be less than that of the algorithms presented in [12]–[14]. One of the good ways to compare the proposed FST algorithm with existing algorithms [12]–[14] will be based on the intermediate trees (or subgraphs) generated during the whole reliability evaluation process. In this way, one can tell how much memory space and time unit required for their algorithms to run the distributed program. We will present some such comparison results in Section IV-D.

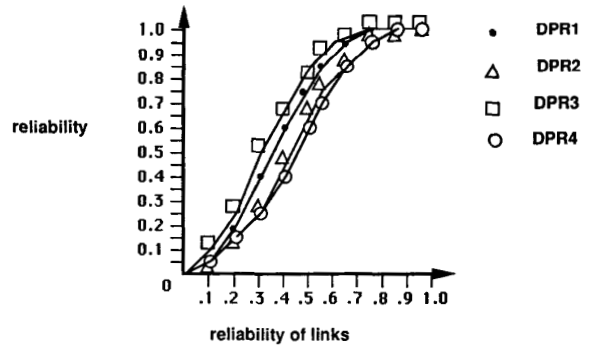


Fig. 10. Reliability of each DPR.

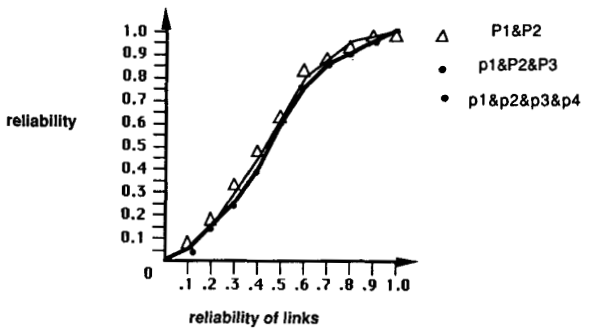


Fig. 11 Reliability of two or more programs running together.

D. Algorithms Comparison

Unlike Kumar's algorithm [12], [13] required two passes to formulate the reliability of the DPS, i.e., to generate all MFST's first and then use reliability analysis program such as SYREL [11] to compute the reliability DPR and DSR, our algorithm requires only one pass to generate the FST's and compute the reliability. Kumar's algorithm [12], [13] has potential to generate a lot of replicated MFST's during the MFST's generating process. Thus, it has to pay extra effort to remove the replicated trees. Our algorithm guarantees no replicated FST's to be generated during the subgraphs generating process. Fig. 14 presents the replicated FST's generated while computing the DPR_1 in the DPS in Fig. 8 by using Kumar 86's algorithm.

Although the algorithm presented in [14] also uses one pass to generate FST (by matrix representation) and compute the reliability, it only addresses a single distributed program issue. For problem statements such as 2, 3, and 4 listed in Section II-C, their algorithm cannot solve such problems. In general, the difference between the FST algorithm and existing algorithms [12]–[14] lies in that existing algorithms use the concept of tree growing while the FST algorithm uses the concept of graph cutting.

Since the time and space required of these algorithms are bound by the number of subgraphs (or trees) generated during the FST generating process, we present some sampling DPS graphs for comparison.

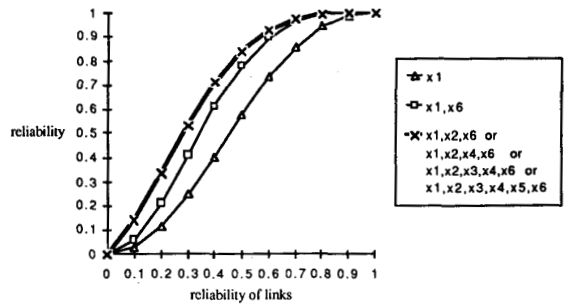


Fig. 12. DPR_1 with several copies of distributed program one.

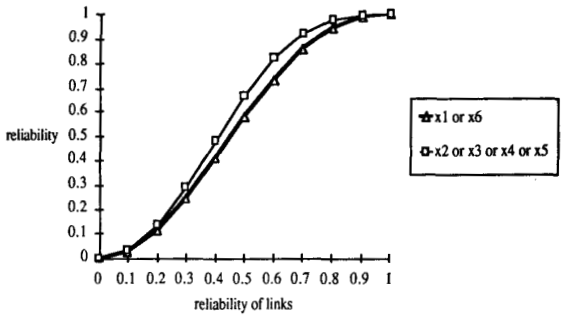


Fig. 13 The DPR_1 under different user sites

Table I presents the best case of our algorithm. It should be noted that the current algorithms in [Kumar 86, 88] cannot compute the reliability of a distributed program with no data files in the distributed networks. Results in Table I are based on the modification of their algorithms to take care such problem. Table II, III, and IV show the cases for the reliability analysis of programs 1, 2, and 3 respectively. The size of the graphs (or trees) is measured by the number of links.

For the actual execution time comparison, we present the DPR_i $(i = 1, 2, 3, 4)$ analysis based on the IBM RISC System/6000 under single user environment to collect execution time. All four algorithms are structured to have the same I/O activities to insure the fairness of the comparison. These four programs are listed in the Appendix in [19]. It is clear that the FST_SPR algorithm has the best performance while Kumar 86's algorithm is the worst one. This result justifies that the tedious and time consumming procedures to check replicated trees and to remove them from the TRY-LIST dominate the whole computation time. The computation time (in microseconds) of the DPR_i are listed in Table V.

Overall speaking, the FST algorithm has the following advantages compared that with existing algorithms.

1) The FST algorithm generates less subgraphs and thus saves the computation time and space. Unlike Kumar's algorithm [12], [13] which has potential to generate replicated FST's which requires a tedious checking process, our algorithm guarantees no replicated FST's to be generated during the subgraphs generation.
2) With the incorporation of *nodes merged*, *series*, *degree-2*, and parallel reduction methods into the FST relia-

TABLE I
SUBGRAPHS GENERATED FOR COMPUTING DPR_4

Size	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Total
FST	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
FST_SPR	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Kumar 86	0	0	0	0	0	0	1984	2206	1144	396	112	32	8	2	1	5885
Kumar 88	0	0	0	0	0	0	640	828	499	206	67	22	7	2	1	2272

TABLE II
SUBGRAPHS GENERATED FOR COMPUTING DPR_1

Size	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Total
FST	1	8	36	119	298	596	918	1001	633	202	25	0	0	0	0	3837
FST_SPR	1	0	0	0	0	1	1	2	3	5	8	9	11	11	67	119
Kumar 86	0	0	0	0	0	0	960	2416	2152	1092	400	116	24	4	1	7165
Kumar 88	0	0	0	0	0	0	464	901	764	420	177	64	18	4	1	2813

TABLE III
SUBGRAPHS GENERATED FOR COMPUTING DPR_2

Size	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Total
FST	1	8	33	96	232	451	651	646	308	52	0	0	0	0	0	2478
FST_SPR	1	0	0	0	1	1	2	2	5	9	10	15	15	18	109	188
Kumar 86	0	0	0	0	0	0	1260	2844	2092	884	284	80	20	4	1	7469
Kumar 88	0	0	0	0	0	0	321	863	748	373	139	46	14	4	1	2509

TABLE IV
SUBGRAPHS GENERATED FOR COMPUTING DPR_3

Size	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	Total
FST	1	8	33	102	251	504	800	939	684	280	59	5	0	0	0	3666
FST_SPR	1	0	0	0	0	0	1	0	2	1	4	5	6	7	26	53
Kumar 86	0	0	0	0	0	0	648	1794	1676	842	284	80	20	4	1	5349
Kumar 88	0	0	0	0	0	0	143	533	595	352	139	46	14	4	1	1827

bility algorithm, our reliability evaluation algorithm for distributed program is much faster and requires less memory space.

3) The FST reliability algorithm addresses some distributed program related problems which were not addressed by some other techniques.
4) The FST Reliability evaluation algorithm is simple and consistent through a special union operation on all vectors LSP representing the probability space of each FST. Our algorithm is a unified approach for both generating FST's and computing the reliability of the DPS.

Fig. 16 shows a different DPS configuration for more comparisons and the results are listed in Table VI.

V. CONCLUSION

Distributed Processing system provides cost-effective ways for improving computer system's performance such as throughput, fault-tolerance, reliability, and so on. The reliability analysis of the DPS becomes an important issue. Traditional approaches for the reliability analysis of computer networks may not be directly applicable for the DPS for that the effects of redundant data files and programs are not captured in these methods. To overcome these limitations, new method should be proposed. In this paper, we present a unified algorithm to generate FST's and to compute the reliability of the DPS. To speed up the reliability evaluation, *nodes merged*, *series*, *degree-2*, and *parallel reduction* techniques are incorporated into the algorithm.

The algorithm presented in this paper is based on the concept of graph cutting to generate FST's. The reliability computation is simple and consistent through a special union operation on all vectors LSP representing the probability space of each FST. The algorithm guarantees no replicated FST's to be generated. The proposed algorithm outperforms existing algorithms in terms of less time and space requirement. This can be evidenced from the various comparisons shown in Section IV-D. It should be mentioned that the best case performance of Kumar's algorithms is that all the data files required for the program to be executed are collided at one same node that also contains the executed program. In fact, this case is not like to happen in the distributed processing system which usually evenly distributed the available resources (programs,data files). Several DPS related problems which are not addressed by other algorithms are studied here

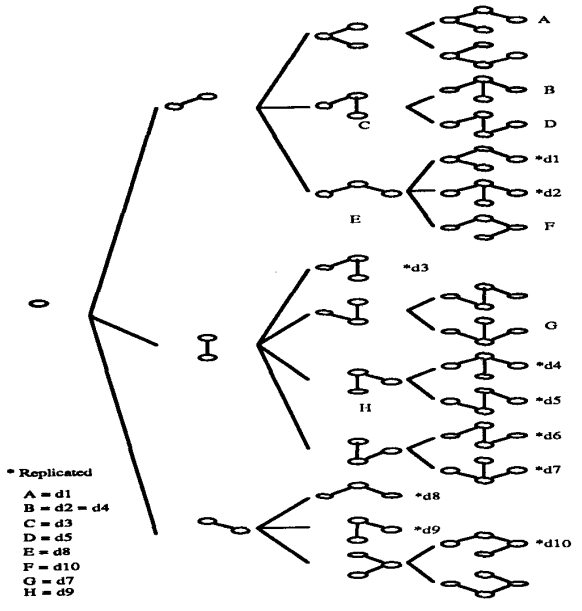


Fig. 14. The replicated FST's generated by Kumar's algorithm.

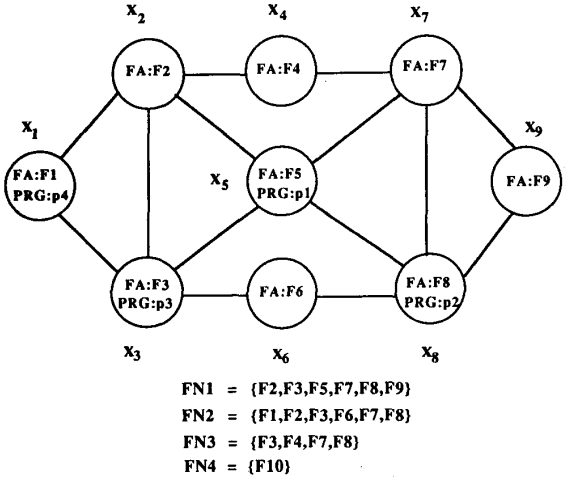


Fig. 15. A complex and large DPS example.

TABLE V
THE COMPUTATION TIME (IN MICROSECONDS) OF EACH DPR_i $(i = 1, 2, 3, 4)$

	DPR_1	DPR_2	DPR_3	DPR_4
FST	2200000	1480000	2090000	0.5
FST_SPR	14000	150000	50000	0.5
Kumar 86	4530000	4860000	2910000	3180000
Kumar 88	340000	310000	230000	250000

using the proposed techniques. These analyses allow us to validate if the reliability performance of an existing DPS meets the required reliability performance for executing a set of distributed programs.

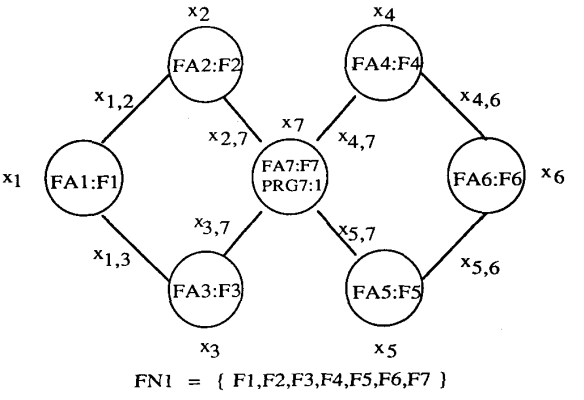


Fig. 16. A DPS with different configuration

TABLE VI
SUBGRAPHS GENERATED FOR COMPUTING DPR_1 IN FIG. 16

Size	8	7	6	5	4	3	2	1	0	Total
FST	1	6	21	18	0	0	0	0	0	46
FST_SPR	1	1	1	1	0	4	4	4	16	32
Kumar 86	0	0	48	68	64	40	16	4	1	241
Kumar 88	0	0	16	24	25	20	10	4	1	100

REFERENCES

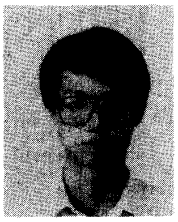
[1] K. K. Aggarwal and S. Rai, "Reliability evaluation in computer-communication networks," *IEEE Trans Reliability*, vol. R-30, pp. 32–35, Apr. 1981.
[2] A. Aggarwal and R. E. Barlow, "A survey of network reliability and domination theory," *Oper. Res.*, vol. 32, no. 3, May–June 1984.
[3] T. C. K. Chou and J. A. Abraham "Load redistribution under failure in distributed systems," *IEEE Trans. Comput.*, vol. C-32, pp. 799–808, Sept. 1983.
[4] 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.
[5] P. Enslow, "What is a distributed data processing system," *IEEE Comput. Mag.*, vol. 11, Jan. 1978.
[6] L. Fratta and U. G. Montanari, "Synthesis of available networks," *IEEE Trans. Reliability*, vol. R-25, no. 2, pp. 81–86, June 1976.
[7] ——, "A recursive method based on case analysis for computing network terminal reliability," *IEEE. Trans. Commun.*, vol. COM-26, pp. 1156–1177, Aug. 1978.
[8] J. Garcia-Molina, "Reliability issues for fully replicated distributed database," *IEEE Comput. Mag.*, vol. 16, pp. 34–42, Sept. 1982.
[9] A. P. Grnarov and M. Gerla, "Multiterminal reliability analysis of distributed processing system," in *Proc. 1981 Int. Conf. Parallel Processing*, Aug. 1981, pp. 79–86.
[10] E. Hansler, "A fast recursive algorithm to calculate the reliability of a communication network," *IEEE Trans. Commun.*, vol. COM-20, June 1972.
[11] S. Hariri, C. S. Raghavendra, and V. K. Prasanna Kumar, "Reliability measures for distributed processing systems," in *Proc. 6th Int. Conf. Distributed Comput. Syst.*, May 1986, pp. 564–571.
[12] K. P. Kumar, S. Hariri, and C. S. Raghavendra, "Distributed program reliability analysis, " *IEEE Trans. Software Eng.*, pp. 2–50 Jan. 1986
[13] C. S. Raghavendra, K. P. Kumar, and S. Hariri, "Reliability analysis in distributed systems" *IEEE Trans. Comput.* vol. 37, pp. 352–358, May 1988.
[14] A. Kumar, S. Rai, and D. P. Agrawal, "On computer communication network reliability under program execution constraints," *IEEE J. Select. Areas Commun.*, pp. 1393–1400, Oct., 1988.
[15] D. A. Rennels, "Distributed fault-tolerant computer systems," *IEEE Comput. Mag.*, vol. 13, pp. 55–65, Mar. 1980.
[16] A. Satyanarayana and A. Prabhakar, "New topological formula and rapid algorithm for reliability analysis of complex networks," IEEE Conf.,

1978.
[17] A. Satyanarayana, "A unified formula for analysis of some network reliability problems," *IEEE Trans. Reliability*, vol. R-31, pp. 23–32, Apr. 1982.
[18] J. A. Stankovic, "A perspective on distributed computer systems," *IEEE Trans. Comput.*, vol. C-33, pp. 1102–1115, Dec. 1984.
[19] D. J. Chen, "RFST: A distributed system reliability analysis program," TR-91-004, Comput. Sci. and Inform. Eng. Dep., National Chiao Tung Univ., 1991.

Deng-Jyi Chen (S'87–M'88) received the B.S. degree in computer science from the Missouri state University, Cape Girardeau, and the M.S. and Ph.D. degrees in computer science from the University of Texas, Arlington, in 1983, 1985, 1988, respectively.

He is an Associate Professor at the National Chiao Tung University, Hsinchu, Taiwan. Prior to joining the faculty of the National Chiao Tung University, he was with the National Cheng Kung University, Tainan, Taiwan, and the United Company, Fort Worth, TX. His interests include object-oriented computing (design methodology, programming language, and computer architecture), software reuse, reliability and performance evaluation of distributed systems, computer networks, and fault-tolerant systems.

Dr. Chen is a member of the IEEE Computer Society and Chinese Open System Association.



Tien-Hsiang Huang received the B.S. and M.S. degrees in computer science and information engineering from the National Chiao Tung University, Hsinchu, Taiwan, in 1989 and 1991, respectively.

His interests include reliability and performance evaluation of distributed systems, development of real-time systems, and object-oriented computing.

Mr. Huang is a member of Chinese Open System Association.