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 the DPR and DSR computation can then be determined by computing the probability that at least one of the FST's is working. The FST algorithm presented in [12] uses breadth-first search method to travel the DPS graph to generate all FST's. According to the algorithm, firstly, all FST's of size \( 0 \) are determined; and so on. For speeding up the reliability evaluation process, node 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, throughpout, 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 (e.g., 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 \( j \).

\( \mu_{i,j} (\eta_{i,j}) \): probability that the link \( x_{i,j} \) is up (down).
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.
- \( PRG_j \): the set of programs available at processing element \( x_j \).
- \( PA_t \): the set of programs available for subgraph \( t \).
- \( LS_t \): a set of link states that represents the links’ conditions in the current subgraph \( t \).
- \( ST_t \): 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 \( \land \) represents the logic AND operation, and its applicable operations are listed below.

\[
\begin{align*}
0 \land 0 & = 0 \\
1 \land 0 & = 0 \\
0 \land 1 & = 1 \\
1 \land 1 & = 1 \\
0 \land * & = 0 \\
1 \land * & = 1 \\
* \land 0 & = 0 \\
* \land 1 & = 1 \\
* \land * & = *.
\end{align*}
\]

**Definition 4:** Operator \( \lor \) 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 \( 0 \lor 1 = * \) under the condition of two \( LSP_t \)'s only differ in one bit, e.g.,

\[
\begin{align*}
LSP_{t1} & = **011 \\
LSP_{t2} & = **001 \\
LSP_{t1} \lor LSP_{t2} & = **011 \lor **001 = ++0*1.
\end{align*}
\]

**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 \( FA_{x_1}, FN_{x_1}, \) and \( PRG_{x_1}(j = 1, 2, \ldots, 4) \).

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 \( n, n - 1, \ldots, 0 \) (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.
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 $x_{2,4}$ 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.

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,
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 3. 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.

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

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

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

\[
LS_A = ***** \quad \text{(the initial graph)}
\]

\[
NC_A = 00000 \quad \text{(subgraphs can be generated by cutting any links)}
\]

\[
ST_A = 1*1*1 \quad \text{(since links } x_{1,2}, x_{2,3}, x_{3,4} \text{ have to be working)}
\]

\[
LSP_A = 1*1*1 \quad \text{(it is obtained by } LS \land ST) \]

Likewise, for the subgraphs B, C, and D can be represented as

\[
LS_B = 0**** \quad LS_C = 1*0** \quad LS_D = 1*1*0
\]

\[
NC_B = 00000 \quad NC_C = 10000 \quad NC_D = 10100
\]

\[
ST_B = *111* \quad ST_C = ***** \quad ST_D = *****
\]

\[
LSP_B = 0111* \quad LSP_C = 1*0** \quad LSP_D = 1*1*0
\]

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 \( Pr(LS) = Pr(LSP_A) + Pr(LSP_B) + Pr(LSP_C) + Pr(LSP_D) = 1 \), where \( Pr(LSP_C) = Pr(LSP_C) \), \( Pr(LSP_D) = Pr(LSP_D) \), 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 \( ST_t \lor LS_t \), 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 \( L_{ST_t} \).
5) The reliability of the DPS graph is obtained by unioning all \( L_{ST_t} \)-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
\( t = \text{original graph} \);
TRY \(= t; \) (store the original graph into list TRY)
FOUND = \( \phi \);
\( L_{Si} = *...* \);
\( NC_t = 00...0 \);
\( FN = \cup_i F_N i \) (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 \( ST_t \);
\( L_{SP_t} = L_{Si} \land ST_t \);
2.3: cutting step (generate subgraphs) add subgraph\( (t) \) to TRY ;
until (TRY \(= \phi \))
step 3: compute reliability
for all \( t \) in FOUND do
\( R = R \lor L_{SP_t} \);
end
Reliability = \( \Pr(R) \);
end

procedure subgraph\( (t) \)
begin
child = \( \phi \);
temp = \( \phi \);
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 \( x_{i,j} \in \text{cut} \) do
\( L_{SNewt} = L_{Si} \);
set the state of \( x_{i,j} \) in \( L_{SNewt} \) to 0; /* cut link \( x_{i,j} \) now */
find all links \( \in \text{temp} \) and set those links’ states in \( L_{SNewt} \) 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 ;
end (* subgraph *)
end

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 \( L_{SB3} = 010** \) indicates that node \( x_2 \) 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, \( NC_{B3} = 01000 \) 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 \( F_k \) from node \( x_n \), delete all nodes in the DPS that contains data file \( F_k \).
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 \( F_k \) from node \( x_n \), delete all nodes in the DPS that contain program \( F_k \).
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 \( x_{n,j} \) and delete the other link \( x_{n,j} \), where node \( x_i \) 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 \( F_1, F_2, F_3 \), and \( F_4 \) 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 \( LS_{\text{newt}} \). These new representing schemes are discussed below.

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

This can be seen from the fact that \( x'_{i,j} = x_{i,j} \otimes x_{k,l} \). 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.

2) \( ST_t \) and \( LS_{\text{newt}} \) representation in \( series \ reduction \): Let two links \( x_{i,j} \) and \( x_{k,l} \) in graph

   \( t \) be series reduced or be degree-2 reduced into \( x'_{i,j} \), \( x'_{k,l} \) itself be one of the links in the spanning tree we found, and the bit state sequence of both \( ST_t \) and \( LS_{\text{newt}} \) be \( \{\ldots x_{i,j} \land x_{k,l} \ldots\} \), then
   a) the state of \( ST_t \) after reduction will be \( \{\ldots 111\ldots\} \lor \{\ldots 101\ldots\} \lor \{\ldots 011\ldots\} \), and
   b) the state of \( LS_{\text{newt}} \) (newt is a child of \( t \)) after reduction will be \( \{\ldots 000\ldots\} \lor \{\ldots 010\ldots\} \lor \{\ldots 001\ldots\} \).

This can be seen from the fact that \( x'_{i,j} = (x_{i,j} \lor x_{k,l}) \). It also implies that if 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 both links \( x_{i,j} \) and \( x_{k,l} \) must have failed.

3) \( ST_t \) and \( LS_{\text{newt}} \) representation in \( degree-2 \ reduction \): Let two links \( x_{i,j} \) and \( x_{k,l} \) in graph

   \( t \) be degree-2 reduced into \( x'_{i,j} \) and \( x'_{k,l} \) itself be one of the links in the spanning tree we found, and the bit state sequence of both \( ST_t \) and \( LS_{\text{newt}} \) be \( \{\ldots x_{i,j} \lor x_{k,l} \ldots\} \), then
   a) the state of \( ST_t \) after reduction will be \( \{\ldots 111\ldots\} \lor \{\ldots 101\ldots\} \lor \{\ldots 011\ldots\} \), and
   b) the state of \( LS_{\text{newt}} \) (newt is a child of \( t \)) after reduction will be \( \{\ldots 000\ldots\} \lor \{\ldots 110\ldots\} \lor \{\ldots 010\ldots\} \lor \{\ldots 001\ldots\} \).

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

\[ t = \text{original graph}; \]

\[ \text{TRY} = \{t\} \quad (\text{store the original graph into list TRY}) \]

\[ \text{FOUND} = \phi \]

\[ L_{St} = ** \ldots *; \]

\[ NC_t = 00 \ldots 0; \]

\[ FN = \bigcup_i FN_i \quad (\text{where program } i \in PV); \]

\[ R = \phi; \]

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;

\[ ST_t = ** \ldots * \]

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;

\[ L_{Sp_t} = L_{St} \land ST_t; \]

2.4 cutting step

add subgraph(\( t \)) to TRY;

until (\( \text{TRY} = \phi \))

step 3: compute reliability

for all \( t \) in FOUND do

\[ R = R \lor L_{Sp_t}; \]

od

Reliability = \( \Pr(R); \)

end

procedure subgraph(\( t \))

**begin**

\[ \text{child} = \phi; \]

\[ \text{temp} = \phi; \]

find each \( x_{k,l} \) where \( x_{k,l} \in \text{stree} \) and the state of \( x_{k,l} \) in \( NC_t = 0 \)

\[ \text{cut} = \bigcup_{k,l} x_{k,l}; \quad /* \text{cut all the links can be cut } */ \]

for all \( x_{i,j} \) cut do

\[ L_{new} = L_{St}; \]

set the state of \( x_{i,j} \) in \( L_{new} \) to 0;

/* cut link \( x_{i,j} \) now */

find all links which \( \text{temp} \) and modify \( L_{new} \) with these links by new representation discussed in Section III-D;

merge(new, temp);

\[ \text{temp} = \text{temp} \cup \{x_{i,j}\}; \]

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;

end

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 \( (FA_i \cap FN = \phi) \) AND \( (PRG_i \cap PV = \phi) \)
        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
    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 \( \text{deg}(x_i) = 2 \)
      then for all the adjacent link \( x_{i,j} \) of node \( x_i \) do
        delete link \( x_{i,j} \);
        repeat.
          get a data file \( F_k \) from \( FA_i \) or a program \( P_k \) from \( PRG_i \), delete
          all nodes that contain data file \( F_k \) or program \( P_k \) 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 \( F_k \) or \( P_k \) 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 \( x_{k,j} \);
            \( p_k \) (or \( p_k' \) ) = \( p_k * p_{ij} \);
            \( FA_k = FA_k \cup FA_i \); \( PRG_k = PRG_k \cup PRG_i \);
            \( FA_j = FA_j \cup FA_i \); \( PRG_j = PRG_j \cup PRG_i \);
        od
  od

end

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


The splitting subgraphs generated by the FST-SPR reliability algorithm is illustrated in Fig. P1, with the results in [12] and [13]. DPR is equal to 0.9963. These results are correctly matched if we assume all the links have the same reliability 0.9, then DPR equals 0.99891.

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 DPR. We use the Distributed Program Reliability (DPR) to describe the reliability of the distributed program running under the DPS. The resulting reliability is under the assumption of all links with reliability 0.9.

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 LSPj. Let Pr(i) be the probability subgraph i, then

\[
Pr(A) = Pr(1^{*1})
Pr(B) = Pr(01111) + Pr(1) + Pr(1c) = Pr(01111 \lor 0110 \lor 01011 \lor 0101 \lor 0111 \lor 0111 \lor 0111 \lor 0001) = Pr(01111 \lor 0110 \lor 01011 \lor 0101 \lor 0111 \lor 0111 \lor 0111 \lor 0001) = Pr(01111 \lor 0110 \lor 01011 \lor 0101 \lor 0111 \lor 0111 \lor 0111 \lor 0001)
\]

**B. Other Evaluation Results**

Reliability problems listed in Section II-C are addressed here. Fig. 10 presents the single distributed program reliability (DPR1, DPR2, DPR3, and DPR4) running under the DPS in Fig 9. This analysis can help us to understand the reliability of each program and the detailed information of each node is given in sets of FALT, PRL, and FN.

Again, we want to compute the DPR1 and DSR of the DPS. Applying the same reliability program, we get DPR1 = 0.9995076 and DSR = 0.9975515 under the assumption of all links with reliability 0.9.

C. The Correctness and Time Complexity of the Algorithm

**Theorem 1:** Given a graph G = (V, E) where V = \{x1, x2, ..., x_m\} is a set of nodes, and E = \{L1, L2, ..., L_n\} is a set of links. Let L = \{L1, L2, ..., L_m\} \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.

\[
Pr(G) = p_1p_2...p_{m-1}Pr(G_1) + q_1Pr(G_2) + p_1q_2Pr(G_3) + \ldots + p_1p_2...p_{m-2}q_{m-1}Pr(G_m)
\]

where pi denotes the probability that Li is working, qi denotes the probability that Li is failure, G1 denotes G with L1, L2, ..., Lm-1 is working. G2 denotes G with L1 is failure. G3 denotes G with L1 is working and L2 is failure, ..., Gm denotes G with L1, L2, ..., Lm-2 is working and Lm-1 is failure.

**Proof:** By factoring theorem (or conditional probability).

\[
Pr(G) = q_1Pr(G|F_1) + p_1Pr(G|F_1) = q_1Pr(G_1) + p_1Pr(G'_1)
\]

where \(F_1\) denotes the event that L1 is failure, \(F_1\) denotes the event that L1 is working, \(G_1\) denotes G with L1 is failure, \(G'_1\) denotes G with L1 is working.
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 $G_2$ denotes $G'_2$ with $L_2$ is failure, $G'_3$ denotes $G'_3$ 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.

Prm(G) = q1Prm(G2) + p1Prm(G'2) = q1Prm(G2) + p1q2Prm(G3) + p1p2Prm(G'3)
Fig. 8 (a) An example of DPS. (b) The splitting snapshot of example 2.
when a graph has a node with node degree reduction occurs when a graph with two or more links are cut in the rest of its subgraphs generation, each parallel probability algorithm. What we need to show is the correctness of connected between two nodes, each series reduction occurs nodes merged, series, parallel, each nodes.

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 analyzing these probability subgraphs to the terms in Theorem 1, we found that the original probability graph represents Pr(G); subgraph A represents P1,2P1,3P2,3Pr(GA); subgraph B represents q1,2Pr(GB); subgraph C represents P1,2P2,3Pr(GC); subgraph D represents P1,2P2,3P3,4Pr(GD). More precisely,

$$\text{Pr}(LS_G) = \text{Pr}(LS_P) + \text{Pr}(LS_B) + \text{Pr}(LS_C) + \text{Pr}(LS_D)$$

$$= \text{Pr}(1 * 1 + 0 * * * * ) + \text{Pr}(1 * 0 * * ) + \text{Pr}(1 * 1 * 0)$$

$$= P1,2P2,3P3,4Pr(GA) + q1,2Pr(GB)$$

$$+ P1,2P2,3P3,4Pr(GC) + P1,2P2,3P3,4Pr(GD)$$

$$= \text{Pr}(G).$$

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

$$\text{Pr}(LS_G) = \text{Pr}(LS_P) + \text{Pr}(LS_B) + \text{Pr}(LS_C) + \text{Pr}(LS_D)$$

$$= P1,2P2,3P3,4 ... Pn,nPr(G1) + q1,2Pr(G2)$$

$$+ q1,2P2,3P3,4Pr(G3)$$

$$+ \cdots + P1,2P2,3P3,4 ... Pn,nPr(Gn)$$

$$= \text{Pr}(G).$$

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 node 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.

**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 non-disjoint 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.
D. Algorithms Comparison

Unlike Kumar’s algorithm \cite{12}, \cite{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 \cite{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 \cite{12}, \cite{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 DPR1 in the DPS in Fig. 8 by using Kumar 86’s algorithm.

Although the algorithm presented in \cite{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 \cite{12}–\cite{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.

Table I presents the best case of our algorithm. It should be noted that the current algorithms in \cite{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 \(i = 1, 2, 3, 4\) 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 \cite{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 consuming 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 with existing algorithms:

1) The FST algorithm generates less subgraphs and thus saves the computation time and space. Unlike Kumar’s algorithm \cite{12}, \cite{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 DPR1

<table>
<thead>
<tr>
<th>Size</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>FST SPR</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Kumar 86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5885</td>
</tr>
<tr>
<td>Kumar 88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2272</td>
</tr>
</tbody>
</table>

TABLE II
Subgraphs Generated for Computing DPR2

<table>
<thead>
<tr>
<th>Size</th>
<th>14</th>
<th>13</th>
<th>12</th>
<th>11</th>
<th>10</th>
<th>9</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST</td>
<td>1</td>
<td>8</td>
<td>36</td>
<td>119</td>
<td>298</td>
<td>596</td>
<td>918</td>
<td>1001</td>
<td>633</td>
<td>202</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3837</td>
</tr>
<tr>
<td>FST SPR</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>11</td>
<td>11</td>
<td>67</td>
<td>119</td>
</tr>
<tr>
<td>Kumar 86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>660</td>
<td>2416</td>
<td>2152</td>
<td>1092</td>
<td>400</td>
<td>116</td>
<td>24</td>
</tr>
<tr>
<td>Kumar 88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>464</td>
<td>901</td>
<td>764</td>
<td>420</td>
<td>177</td>
<td>64</td>
<td>18</td>
</tr>
</tbody>
</table>

TABLE III
Subgraphs Generated for Computing DPR3

| 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  | 0  | 1 | 1 | 2 | 2 | 5 | 9 | 10 | 15| 15 | 18 | 109 | 188  |
| Kumar 86 | 0 | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 | 1260| 2844| 2092| 884 | 284| 80 | 20 | 7469 |
| Kumar 88 | 0 | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 | 321 | 863 | 748 | 373 | 139| 46 | 14 | 2509 |

TABLE IV
Subgraphs Generated for Computing DPR4

| 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  | 0  | 3666 |
| FST SPR | 1 | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 | 1 | 2 | 4 | 5 | 6 | 7 | 26 | 53    |
| Kumar 86 | 0 | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 | 648| 1794| 1676| 842 | 284| 80 | 20 | 5349 |
| Kumar 88 | 0 | 0  | 0  | 0  | 0  | 0 | 0 | 0 | 0 | 143| 533 | 595 | 352 | 139| 46 | 14 | 1827 |

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, a new algorithm 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 node that also contains the executed program. In fact, this case is not likely to happen in the distributed processing system which usually evenly distributes the available resources (programs, data files). Several DPS related problems which are not addressed by other algorithms are studied here.
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.

TABLE V

<table>
<thead>
<tr>
<th>Size</th>
<th>8</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>FST</td>
<td>1</td>
<td>6</td>
<td>21</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>46</td>
</tr>
<tr>
<td>FST: SPR</td>
<td>1</td>
<td>1</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Kumar 86</td>
<td>0</td>
<td>0</td>
<td>48</td>
<td>68</td>
<td>64</td>
<td>40</td>
<td>16</td>
<td>4</td>
<td>1</td>
<td>241</td>
</tr>
<tr>
<td>Kumar 88</td>
<td>0</td>
<td>0</td>
<td>16</td>
<td>24</td>
<td>25</td>
<td>20</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>100</td>
</tr>
</tbody>
</table>

Fig. 16. A DPS with different configuration

TABLE VI

<table>
<thead>
<tr>
<th>SUBGRAPHS GENERATED FOR COMPUTING DPR1 IN FIG. 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN1 = (F1,F2,F3,F4,F5,F6,F7)</td>
</tr>
<tr>
<td>FN2 = (F1,F2,F3,F4,F5,F6,F7)</td>
</tr>
<tr>
<td>FN3 = (F3,F4,F5,F6)</td>
</tr>
<tr>
<td>FN4 = (F10)</td>
</tr>
</tbody>
</table>

REFERENCES

1978.


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.

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.