Reliability Evaluation of Dependable Distributed Computing Systems Based on Recursive Merge and BDD

Yung-Ruei Chang†, Hung-Yau Lin, and Sy-Yen Kuo
Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan
sykuo@cc.ee.ntu.edu.tw

Abstract

System reliability evaluation, sensitivity analysis, importance measures, failure frequency analysis and optimal design have become important issues for distributed dependable computing. Finding all the Minimal File Spanning Trees (MFST) and avoiding repeatedly computing the redundant MFSTs is the key technique for evaluating the reliability of a distributed computing system (DCS) in previous works. However, identifying all the disjoint MFSTs is difficult and very time consuming for large-scale networks. Although existing algorithms have been demonstrated that they work fine on medium-scale networks, they have two inherent drawbacks. First, they do not support efficient manipulation of Boolean algebra. The sum-of-disjoint-products method used by them is inefficient in dealing with large Boolean functions. Second, the tree-based partitioning algorithm does not merge isomorphic sub-problems and therefore, redundant computations cannot be avoided. In this paper, we propose a new efficient algorithm for the reliability evaluation of a DCS based on recursive merge and binary decision diagram (BDD). Using the BDD substitution technique, we can easily apply our algorithm to a network with imperfect nodes. The experimental results show a significant improvement on the execution time compared to previous works.

1. Introduction

The development of computer networking and embedded VLSI processing devices has led to an increasing interest in distributed computing systems (DCS) in which the computations are distributed among many processing elements (PEs). Distributed computing involves cooperation among several loosely coupled computers communicating over a network. Distributed systems provide cost-effective means for resource sharing and extensibility, and obtain potential increases in performance, reliability, and fault tolerance. A distributed program usually requires one or more of the resources for successful execution, such as PEs, data files, etc. For successful completion of a program, the local host (the PEs that contain the required files) and the interconnection links must all function correctly. Therefore, the distribution of data files can affect the overall reliability of the system. Thus, an important problem in distributed system design and analysis is to define and evaluate various reliability measures as well as estimate the effect of program and resource distributions on the reliability of a system efficiently. This analysis is crucially important for building a reliable distributed computing system.

There were many researchers studying the distributed program reliability (DPR) and distributed system reliability (DSR). Kumar *et al*. [1] seems to be the first to present the definition of the DPR and DSR. They constructed a distributed model including edges, nodes and resource files and proposed the Minimal File Spanning Trees (MFST)-based algorithm to evaluate the DPR and DSR. Later, based on MFST, Raghavendra [2] addressed two measures, distributed program-user reliability and distributed system-user reliability, and proposed an algorithm for their evaluations. Kumar [3] also developed a fast algorithm to evaluate the DPR and DSR. These methods are 2-step algorithms. First, they need to find all the MFSTs. Second, they convert these MFSTs to a symbolic reliability expression using an existing reliability evaluation algorithm like SYREL [4] to compute the disjoint probability. The major drawback with these methods is that finding all the MFSTs has high computational complexity; and prior knowledge about multi-terminal connections is required in order to compute the reliability expression, thereby making them inapplicable to large systems. To overcome these problems, Kumar [5] proposed a 1-step algorithm GEAR that can avoid computing the redundant MFSTs and reduce

† Y.R. Chang is also with Institute of Nuclear Energy Research, Atomic Energy Council, Taiwan.

Acknowledgment: This research was supported by the National Science Council, Taiwan, R.O.C. under Grant NSC 92-2213-E-002-011.

computational time. To further improve the efficiency of reliability assessment, Chen [6][7] proposed FST-SPR and HRFST algorithm based on the cut-set methods for reducing the reliability evaluation complexity. However, applying their methods to the network with imperfect nodes is not easy. Taking the existence of faulty nodes into account, Ke [8] proposed the ENR/KW algorithm to compute the reliability of a distributed computing network with imperfect nodes. ENR/KW algorithm needs to find the set of mandatory nodes and does not converge the isomorphic subproblems. Later, based on the model of [1], Zang [9] proposed a Binary Decision Diagram (BDD)-based algorithm to analyze the dependability of a DCS with imperfect fault-coverage. The researches in [10][11][12] continued with the study of the DPR and DSR based on the model of Kumar [1].

Finding all the MFSTs and avoiding the computation of generating the redundant MFSTs is the key technique to evaluate the reliability of a DCS in previous works. However, identifying all the disjoint MFSTs is difficult and is very time consuming. Although the algorithms in previous works have been demonstrated with reasonable efficiency on medium-scale networks, they have two inherent drawbacks. First, they do not support efficient manipulation of Boolean algebra. The sum-of-disjoint-products method used by them is inefficient in dealing with larger Boolean functions. Second, the tree-based partitioning algorithm does not consider the convergence of isomorphic subproblems and therefore, redundant computations cannot be exactly avoided.

Recent literature [13][14][15][16][17][18][19] show that BDD is a very efficient approach for reliability evaluation. In this paper, we propose a BDD-based algorithm, named CLK, to compute the reliability of a DCS with both perfect and imperfect nodes. The main idea, which makes the CLK algorithm more efficient than the previous works, is that the BDD representing the Boolean reliability expression of a DCS can be constructed by avoiding the redundant computation of the isomorphic sub-problems during the merging process. Therefore, the reliability can be quickly derived from the BDD. In addition, our method can be integrated with the methodologies that use the BDD to analyze the dependability of a system, such as system availability, system failure frequency, importance measures and sensitivity analysis [19].

Section 2 introduces the concepts of BDD and distributed computing systems. Section 3 illustrates an efficient algorithm based on recursive merge and BDD to evaluate the DPR and DSR of a distributed computing system. Based on the BDD substitution technique, our algorithm is applicable to not only a system with perfect nodes but also a system with imperfect nodes. The experimental results on various benchmark networks are shown in Section 4. Section 5 gives the conclusions and future works.

2. Preliminaries

2.1. Binary decision diagram (BDD)

BDD [13] is based on a disjoint decomposition of a Boolean function called the *Shannon expansion*. Given a Boolean function $f(x_1,\cdots,x_n)$, then for any $i \in \{1,\cdots,n\}$; $\overline{x}_i \equiv \neg x_i = 1 - x_i$:

$$f = x_i \cdot f_{x_i=1} + \overline{x}_i \cdot f_{x_i=0} \tag{1}$$

In order to express the Shannon decomposition concisely, the if-then-else (*ite*) format [20][21] is defined as:

$$f = ite(x_i, f_{x_i=1}, f_{x_i=0})$$

The way that BDDs are used to represent logical operations is simple. In practice, the BDD is generated by using logical operations on variables. Let Boolean expressions *f* and *g* be:

$$f = ite(x_i, f_{x_i=1}, f_{x_i=0}) = ite(x_i, F_1, F_0)$$
$$g = ite(x_j, g_{x_j=1}, g_{x_j=0}) = ite(x_j, G_1, G_0)$$

A logic operation between *f* and *g* can be represented by BDD manipulations as:

$$ite(x_i, F_1, F_0) \lozenge ite(x_j, G_1, G_0) =$$
$$\begin{cases} ite(x_i, F_1 \lozenge G_1, F_0 \lozenge G_0) & \text{ordering}(x_i) = \text{ordering}(x_j) \\ ite(x_i, F_1 \lozenge g, F_0 \lozenge g) & \text{ordering}(x_i) < \text{ordering}(x_j) \\ ite(x_j, f \lozenge G_1, f \lozenge G_0) & \text{ordering}(x_i) > \text{ordering}(x_j) \end{cases} \tag{2}$$

where $\lozenge$ represents a logic operation such as AND or OR. Fig. 1 illustrates the construction and manipulation steps of a Boolean function. For more details on using the op-

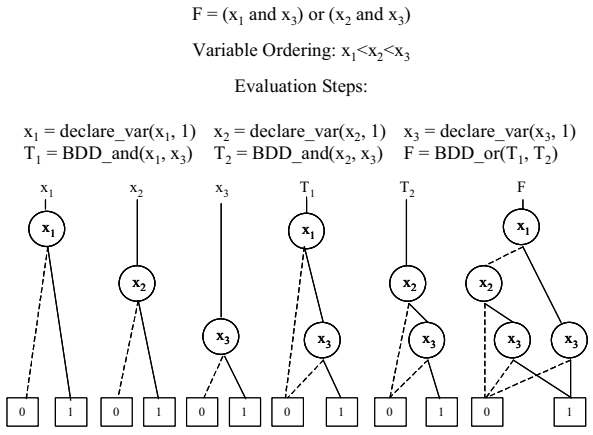

Figure 1. The BDD generated from a Boolean equation.

erations of BDD, please refer to [13].

A useful property of BDD is that all the paths from the root to the leaves are mutually disjoint. If f represents the Boolean expression of the system availability, based on the property of the disjoint decomposition of BDD, the reliability (or availability) of a system can be recursively evaluated by (1) as

$$\Pr\{f\} = \Pr\{x_i\} \cdot \Pr\{f_{x_i=1}\} + \Pr\{\overline{x}_i\} \cdot \Pr\{f_{x_i=0}\} \qquad (3)$$

where $\Pr\{\cdot\}$ means $\Pr\{\cdot=1\}$ for simplification. For example, if $\Pr\{x_i\}$ is the availability A_i of component i and U_i is the unavailability of component i, then the system availability A is:

$$A = \Pr\{f\} = A_i A_{x_i=1} + U_i A_{x_i=0} = A_i A_{x_i=1} + (1 - A_i) A_{x_i=0} \quad (4)$$

where $A_{x_i=1}$ and $A_{x_i=0}$ represent $\Pr\{f_{x_i=1}\}$ and $\Pr\{f_{x_i=0}\}$ respectively. Similarly, the unavailability of a system can be calculated as:

$$U = \Pr\{g\} = U_i U_{\overline{x}_i=1} + A_i U_{\overline{x}_i=0} \qquad (5)$$

where g is the system unavailability expression and the dual of f ; i.e. $f(x_1, x_2, \cdots, x_n) \equiv 1 - g(1 - x_1, 1 - x_2, \cdots, 1 - x_n) \equiv 1 - g(\overline{x}_1, \overline{x}_2, \cdots, \overline{x}_n)$, $U_{\overline{x}_i=1}$ and $U_{\overline{x}_i=0}$ represent $\Pr\{g_{\overline{x}_i=1}\}$ and $\Pr\{g_{\overline{x}_i=0}\}$ respectively.

2.2. Distributed computing system (DCS)

In [1], Kumar *et al*. modeled a DCS as an undirected graph $G[V, E]$ in which the nodes represent the hosts and the edges represent the communication links, where V is a set of nodes and E is a set of edges. Fig. 2 shows an example of a six-node DCS. FA_i represents the set of files that could be obtained at node i. PRG_i represents the set of programs that could be run at node i. FN_j represents the set of required files for the successful execution of program j. For example, program P_2 could be executed on either n_3 or n_4. According to $FN_2 = \{F_2, F_4\}$, program P_2 can run successfully on n_3 due to the successful access of the data files $\{F_2, F_4\}$. However, program P_2 could not be run successfully on n_4 without the successful access of the data $\{F_4\}$ since only the data $\{F_2\}$ is provided at node n_4. A file-spanning tree (FST) is defined as a spanning tree that connects the root node (the host node that runs the program under consideration) to some other nodes such that its nodes contain all the required files for the successful execution of the program. An FST is a minimal file-spanning tree (MFST) if there exists no other FST that is a subset of this FST. For instance, program P_2 in Fig. 2 will function if it can run at node n_3 or n_4, and can access files $\{F_2, F_4\}$. Therefore, $\{n_3\}$ and $\{n_4, x_4, n_2, x_3, n_3\}$ are two FSTs of P_2, but the later one is not a MFST since $\{n_3\}$ is a subset of $\{n_4, x_4, n_2, x_3, n_3\}$. The set of MFSTs of program P_2 in Fig. 2 are: $\{n_3\}$, $\{n_4, x_4, n_2, x_5, n_5\}$, $\{n_4, x_7, n_5\}$, $\{n_4, x_8, n_6, x_9, n_5\}$.

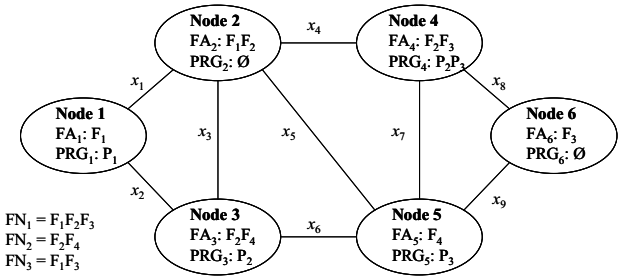

Figure 2. An example of a six-node DCS.

By the definition of MFST [1], the distributed program reliability (DPR) for program j and the distributed system reliability (DSR) are defined respectively as:

$$DPR_j = \Pr\{\bigcup_{i=1}^{n_{mfst}} MFST_i = 1\} \qquad (6)$$

$$DSR = \Pr\{\bigcup_{i=1}^{m_{mfst}} MFST_i = 1\} \qquad (7)$$

where n_{mfst} is the number of MFSTs belonging to program j and m_{mfst} is the number of MFSTs over all programs.

3. The CLK algorithm

Finding FSTs and using MFSTs to compute the DPR by the disjoint method in the previous works is difficult when the DCS network becomes large and complex. In this section, we will develop an efficient algorithm, named CLK, based on the convergence of the isomorphic sub-problems to compute the DPR and the DSR of a distributed computing system. With this method, redundant computations can be avoided. The experimental results presented later will show the effectiveness of our approach compared to the previous works [1][6][7][8][9][22]. Moreover, the CLK algorithm has the capability of dealing with large number of Boolean variables using BDD. Therefore, the CLK algorithm is applicable to a large-scale DCS. In this section, we will first discuss the algorithm for a DCS network with perfect nodes. Then, using the BDD substitution technique, the algorithm can be easily and efficiently applied to a DCS network with imperfect nodes.

3.1. Algorithm for perfect nodes

Based on the model in [1], a distributed program can be run successfully on a host node if all the required files in the DCS network can be correctly accessed. The basic idea of the CLK algorithm for computing the DPR is to begin

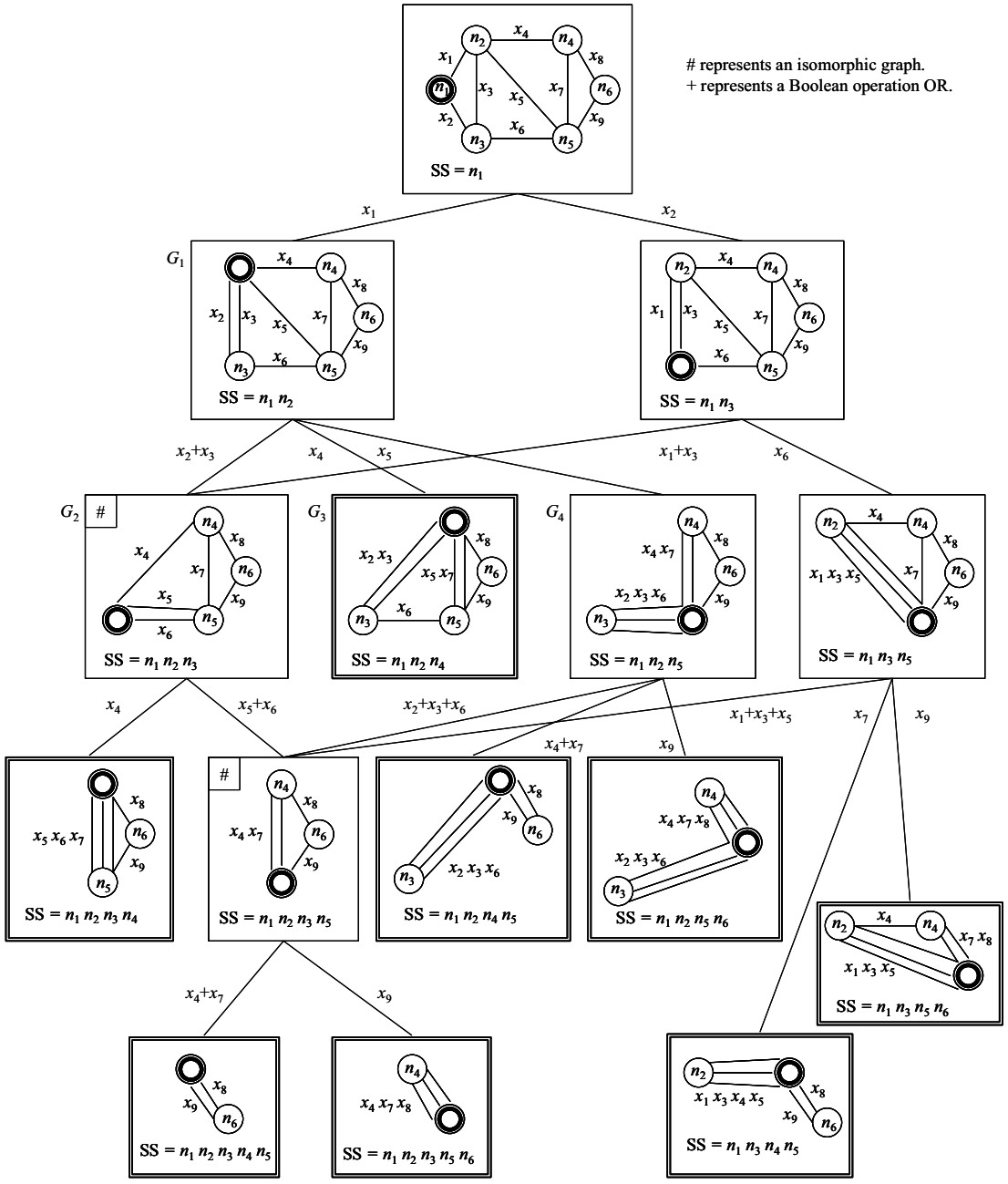

Figure 3. Recursive merge for evaluating DPR_1 in Fig. 2.

the merging procedure from a host node s where the program can run and produce subgraphs by recursively merging the adjacent nodes one by one into the source set (*SS*). Each subgraph has its corresponding *SS*. An *SS* is the set of nodes such that the host node $s \in SS$ and *SS* are connected. Therefore, if the *SS* of a subgraph satisfies the requirement (i.e. contains all the required files), then the merging with adjacent nodes stops. Fig. 3 illustrates the procedure of recursive merge in the CLK algorithm for evaluating the DPR_1 in Fig. 2. The traversed path, which includes the branches from the top-root graph to any one of the bottom-leaf subgraphs in Fig. 3, represents a FST. It can be transformed into the BDD representation by the CLK algorithm. Then taking the advantage of the disjoint property of BDD, the DPR can be efficiently calculated.

Based on the above idea, let each graph have its own

Boolean reliability expression $BF(\cdot)$, i.e. $BF(G)$ represents the Boolean reliability expression of G. Therefore, we get the recursive relationship of $BF(\cdot)$ between graph G and its subgraph $G*n$ as:

$$BF(G) = \bigoplus_{n \in N_{ss}} [BF(G * n) \otimes (\bigoplus_{i \in E_{ss:n}} x_i)] \quad (8)$$

where n is a node of G, N_{ss} is the set of nodes that are adjacently connected to SS, $E_{ss:n}$ is the set of edges that connect n and SS, $G*n$ is the subgraph of G obtained by merging adjacent node $n \in N_{ss}$ into SS and deleting any edge connecting n and SS, $x_i / \overline{x}_i$ is the Boolean edge variable representing edge i is functional/failed (i.e. $x_i = 1 / \overline{x}_i = 1$), $\oplus$ is the Boolean OR operation BDD_or(), and $\otimes$ is the Boolean AND operation BDD_and(). For example, in Fig. 3, $BF(G_1) = [BF(G_2) \otimes (x_2 \oplus x_3)] \oplus [BF(G_3) \otimes x_4] \oplus [BF(G_4) \otimes x_5]$. If $BF(\cdot)$ is implemented by a BDD, then each graph has its own BDD and the manipulation technique of BDD in (2) could be used recursively for performing Boolean operations on edge variables during the merging procedure. Finally, by the bottom-up procedure, we construct the BDD of the top-root graph representing the Boolean reliability expression of the DPR.

In addition, the CLK algorithm can avoid the redundant computations of isomorphic sub-problems. An *SS* of a subgraph implies a certain set of files can be obtained no mater how the nodes in *SS* are connected. During the merging procedure, the rest of edges outside *SS* in a subgraph will not be changed. Therefore, the Boolean reliability expression representing the traversing path outward *SS* of the subgraph will not be changed. This means two subgraphs derived from different merging procedures are isomorphic if their *SS* are the same since the follow-up traversing path for the rest of the required files will be the same. Therefore, the isomorphic subgraphs have the same Boolean reliability expression $BF(\cdot)$, i.e. the same BDD. We can use the property to avoid redundant computations.

According to the above idea, a hash table is built in the CLK algorithm to record these isomorphic subgraphs and their own BDD. When we traverse into an isomorphic subgraph, we can just retrieve the BDD belonging to the isomorphic subgraph from the hash table and use the BDD for Boolean operations in the merging procedure. Therefore, the redundant computations on isomorphic subgraphs can be avoided. This will save significant execution time.

The algorithm is depicted by the pseudo-code in Fig. 4 and is executed by initializing G to the original graph, *SS* to empty, and n to the node at which the program can be executed. The algorithm can recursively merge the nodes in the distributed network and build up the BDD that represents the Boolean reliability expression of a DPR. The hash table in the algorithm can avoid the redundant computations from isomorphic sub-problems. Fig. 3 illustrates the procedure for the evaluation of the DPR_1 in Fig. 2. The diagram contains 8 terminal subgraphs, 6 intermediate subgraphs, and 2 isomorphic subgraphs marked with #. The larger the scale of a distributed computing system becomes, the larger the number of isomorphic subgraphs is. Therefore, the scheme of using the hash table will bring a significant improvement on the efficiency of the CLK algorithm. By the manipulations of BBD_and() and BBD_or() during the merging procedure, the BDD representing the disjoint terms of the Boolean reliability expression of the DPR can be obtained from (8). Thus, the DPR can be recursively derived from the BDD by (4).

```
RecursiveMerage( G, SS, n )
{
   BDD result, tmp1, tmp2;
   SS = SS + {n};
   if ( SS gets all the required files) return BDD_one;
   if ( SS contains all the nodes) return BDD_zero;
   if ( SS gets a hit in the hash table )
   {
      result = getBDD_computed_table(SS);
   }
   else
   {
      result = BDD_zero;
      for each node n_i adjacent to SS
      {
         Sub_G = G * n_i;
         tmp1 = RecursiveMerage( Sub_G, SS, n_i );
         tmp2 = BDD_or( all the edge connected to n_i );
         tmp2 = BDD_and( tmp2, tmp1);
         result = BDD_or( result, tmp2);
      }
      insert_computed_table( SS, result);
   }
   return result;
}
```

Figure 4. The recursive merge algorithm for constructing the BDD with perfect nodes.

3.2. Ordering strategy

One of the key issues in getting full advantage out of BDD is to find a good variable-ordering. The size of BDD and therefore, the efficiency of the whole methodology, strongly depends on the chosen ordering. Unfortunately, the problem of determining the ordering is NP-Complete. However, approximate solutions are relatively easy to find by a local-search algorithm [18]. For fault trees, the ordering obtained by a depth-first left-most traversal of the tree is often good enough. For terminal-pair reliability networks, the ordering obtained by a breadth-first traversal of the network starting from its source is a good candidate [14][15][18].

In our experimental results, the following ordering heuristics is good for a distributed computing network. We order the variables by the ordering that we use for BDD

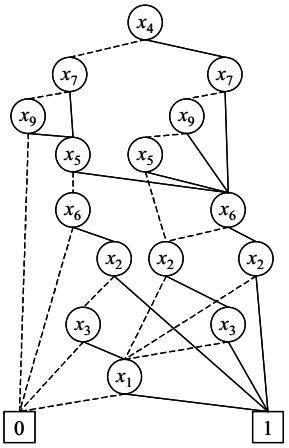

Figure 5. The BDD generated by the CLK algorithm for DPR_1 in Fig. 2 with perfect nodes.

Table 1. Rules of Boolean variable substitution for Fig. 2.

The rules of Boolean variable substitution		
$x_1 \leftarrow n_1x_1n_2$	$x_4 \leftarrow n_2x_4n_4$	$x_7 \leftarrow n_4x_7n_5$
$x_2 \leftarrow n_1x_2n_3$	$x_5 \leftarrow n_2x_5n_5$	$x_8 \leftarrow n_4x_8n_6$
$x_3 \leftarrow n_2x_3n_3$	$x_6 \leftarrow n_3x_6n_5$	$x_9 \leftarrow n_5x_9n_6$

manipulation in the algorithm shown in Fig. 4 during building up the BDD, i.e. we put the variables in the ordered list if we need a new variable to represent a component for BDD manipulation. Therefore, the prior ordering of the variables is used for the BDD manipulation in the algorithm and the prior ordering is in the BDD. For example, in Fig. 3, the ordering is as following:

$$x_4 < x_7 < x_9 < x_5 < x_6 < x_2 < x_3 < x_1 < x_8$$

The final BDD representing the Boolean reliability expression of DPR_1 is obtained as shown in Fig. 5. In the latter section of experimental results, we will make a comparison between the size of BDD generated with our ordering strategy and that with the breadth-first ordering strategy. The experiments show that the CLK algorithm has a great improvement on the size of BDD and is more efficient about 20% in execution time.

3.3. Algorithm for directed graph

The above algorithm is also applicable to directed graphs without modification. But care must be taken as which nodes can be merged to and which nodes can be absorbed into *SS*. In the directed graph, a node v can be merged into *SS* only if there is a node $u \in SS$ such that $<u, v>$ is an edge connecting from u to v. The other part of the algorithm for directed graph is the same as that for undirected graph.

3.4. Algorithm for imperfect nodes

A simple but efficient method dealing with node failures is proposed in [23] and can be embedded in any algorithm. That approach is based on a simple concept: "the failure of a node implies the failure of its incident edges". Therefore, after deriving the BDD of a DCS with perfect nodes in Section 3.1, we can apply the BDD substitution technique on it to solve the problems of a DCS with imperfect nodes. The BDD substitution technique is a useful property in BDD manipulation. The BDD derived in Section 3.1 consists of only edge variables. In order to reckon the effects of imperfect nodes, the node variables should be included. Because one edge connects two nodes, the Boolean edge variable in the BDD can be replaced by the Boolean edge variable and the other two Boolean node variables. Table 1 illustrates the rules of variable substitution in our example. The substitution approach in our algorithm can be implemented by BDD_and() and BDD_or() in the following:

$$IBF(G) = \left(n_j x_i n_k \otimes BF_{|x_i=1}(G)\right) \oplus \left(\overline{n_j x_i n_k} \otimes BF_{|x_i=0}(G)\right) \tag{9}$$

where $x_i / \overline{x}_i$ is the Boolean edge variable representing whether edge i is functional/failed (i.e. $x_i = 1/\overline{x}_i = 1$), $n_i / \overline{n}_i$ is the Boolean node variable representing node i functional/failed (i.e. $n_i = 1/\overline{n}_i = 1$), $IBF(G)$ is the BDD-based reliability function for a network G with imperfect nodes, two end nodes j and k represented by n_j and n_k respectively are connected by the edge i represented by x_i, and $BF_{|x_i=1}(G) / BF_{|x_i=0}(G)$ is the BDD-based reliability function for a network G with perfect nodes given that edge i is functional/failed. Therefore, we can evaluate the reliability of a DCS with imperfect nodes very efficiently. One thing we need to take care is the BDD ordering of the nodes and edges. Applying the ordering heuristic described in Section 3.2, the variable ordering for Fig. 3 becomes

$$n_4 < x_4 < x_7 < n_6 < x_9 < n_5 < x_5 < x_6 < n_3 < x_2 < x_3 < n_2 < x_1 < n_1 < x_8$$

It should be noted that the number of Boolean manipulations in the procedure of network merging and BDD constructing significantly affects the performance of the algorithm. Therefore, we first consider the problem case with perfect nodes to construct the BDD representing a DCS. We manipulate the Boolean operations with only edge variables but no node variables are included to reduce the number of BDD variables. The smaller the size of BDD variables is, the smaller the number of Boolean ma-

nipulations is and thus the faster the algorithm can run. After deriving the BDD representing the Boolean reliability expression of a DCS with perfect nodes, we apply the BDD substitution technique to take the effect of imperfect nodes into consideration and then obtain the final BDD. This scheme reduces the number of Boolean manipulations during the network merging procedure and can save about 20% of the execution time in average.

4. Reliability evaluation of a distributed program running at more than one node

If distributed program j can run at more than one host node in the system, we can separately construct each individual $BDD_{j:i}$ corresponding to the host node i where distributed program j can run. Then, by the Boolean OR operation BDD_or() with the $BDD_{j:i}$ for each node i, the BDD_j representing the Boolean reliability expression of distributed program j can be derived as:

$$BDD_j = \bigoplus_{i=1}^{h_j} BDD_{j:i} \tag{10}$$

where $\oplus$ represents the Boolean OR operation BDD_or(), h_j is the number of host nodes in the system where distributed program j can run, and $BDD_{j:i}$ is the BDD representing the Boolean reliability expression for successfully running distributed program j at node i. Therefore, the DPR_j of distributed program j running at more than one site can be easily evaluated from the BDD_j .

4.1. Distributed system reliability (DSR)

In the last section, we can get the BDD of a given distributed program. For reliability analysis of two or more programs executed simultaneously, we use the Boolean AND operation BDD_and() with the BDD_j for each program j included in the system.

$$BDD = \bigotimes_j BDD_j \tag{11}$$

where $\otimes$ represents the Boolean AND operation BDD_and(). Similarly, the DSR of a distributed computing system can be derived by (11) and (4).

4.2. *K*-terminal network reliability

The evaluation of k-terminal network reliability is a special case of the evaluation of DPR in a DCS. The only difference is that the k terminals are already identified antecedently for evaluating the k-terminal network reliability. Therefore, it is easy to use the algorithm for DCS to compute the k-terminal network reliability. However, it will be difficult to use the algorithm developed for evaluating the k-terminal network reliability to compute a DPR in a DCS since the priori knowledge that the k nodes in a DCS should be antecedently identified is not given. For computing the reliability of a k-terminal network with perfect or imperfect nodes using the CLK algorithm, all we need to do is to distribute the different k required files on the k nodes correspondingly, and empty the set of *FA* at the rest of nodes. Then, the DPR of such distribution of the resource files in the DCS will be the k-terminal network reliability.

Similarly, the evaluation of terminal-pair network reliability is a special case of the evaluation of k-terminal network reliability where $k = 2$. For computing the reliability of a terminal-pair network with either perfect nodes or imperfect nodes, we locate a required file at the source node as well as another required file at the destination node, and empty the set of *FA* in the rest of nodes. Therefore, the DPR of such distribution of the resource files will be the terminal-pair network reliability.

5. Experimental results

The efficiency of the CLK algorithm is compared with the algorithms in [1][6][7][8][9][22]. The implementation of the CLK algorithm is done with the C/C++ language on a primary-type SUN UltraSPARC workstation. Although the running times were measured on different platforms, the capabilities of data processing are roughly equivalent. The reliabilities of all the edges and nodes are assumed to be 0.9. First, we compare the CLK algorithm with the algorithms in [6][7] that were developed based on the cut-set method. Fig. 6 illustrates a complex DCS with eight processing elements in [6][7]. Table 2 shows a comparison between the CLK algorithm and the algorithms in [6][7]. Although they do not provide the execution time in [6][7], we can still compare the complexity by the number of subgraphs generated from different algorithms. The number of subgraphs produced by the CLK algorithm is about 29.09% less than previous results. That means the CLK algorithm has higher computational efficiency. In addition, the CLK algorithm can avoid the redundant computation of isomorphic sub-problems. The high average hit ratio, about 58.85%, of the hash table also makes the CLK algorithm very efficient. Further, by the use of the BDD substitution technique, we can efficiently compute the reliability of each individual distributed program even with imperfect nodes, while it will be difficult for the algorithms in [6][7].

Moreover, for considering the imperfect nodes, we use the benchmarks $G_{i:j}$ given in [8][9], $j \le i$, where i is the number of nodes in the network, and j represents that nodes n_1, n_2, …, n_j are fully connected in network G. For example, Fig. 7 shows the benchmark network $G_{8:6}$. The

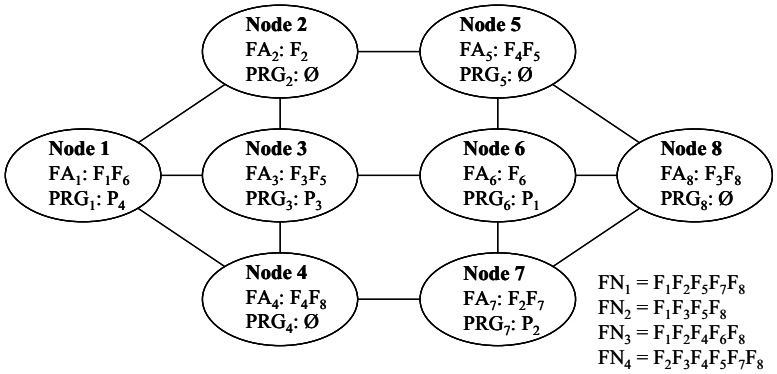

Figure 6. A complex DPS with eight processing elements.

Table 2. The comparison of the number of subgraphs generated from different algorithms.

Distributed program	DPR with perfect nodes	FST_SPR [6]	HRFST [7]	CLK				
		# of subgraphs	# of subgraphs	# of subgraphs	DPR with imperfect nodes	# of *SS* hash hits	Hit ratio (%)	Execution Time in seconds
P_1	0.9961182	445	113	95	0.700269	165	63.46	0.08
P_2	0.9963265	334	124	85	0.766953	107	55.73	0.05
P_3	0.9984532	309	118	95	0.777947	141	59.75	0.07
P_4	0.9963256	389	140	76	0.761795	89	53.94	0.06

program is located at n_1, and files F_1, F_3, F_5 are required to execute it. Table 3 gives the location of the files. Table 4 shows the comparison of the number of subgraphs and the execution time among the CLK algorithm, KHR [1], ENR/KW [8], and the algorithm in [9]. Due to the convergence of isomorphic sub-problems, the CLK algorithm generates far less subgraphs than other algorithms, especially for $G_{10:9}$. Although Zang *et al*. [9] provides a method to analyze the dependability of a DCS with imperfect fault-coverage using BDD, however, the BDD representing the structure function of the DPR is generated via the traditional MFST-searching method.

The CLK algorithm employs the merging method and a hash table to record the isomorphic subgraphs and the corresponding BDDs. Hence the redundant computations of isomorphic subgraphs can be avoided by looking up in the hash table. The average hit rate is about 28.03%. The larger the scale of the DCS becomes, the higher the hit rate is. This makes the CLK algorithm much more efficient. In addition, we apply the BDD substitution technique only after the BDD representation of the DCS with perfect nodes has been constructed. This technique can reduce the time spent in BDD manipulation and therefore, improve the performance of the CLK algorithm.

To compute the terminal-pair network reliability (TPR) for a network with perfect or imperfect nodes, we distribute a required file on the source node as well as another required file on the destination node, and empty the set of *FA* in the rest of nodes. Then, the DPR of such distribution of the resource files will be the TPR. Table 5 shows a comparison of the CLK algorithm with TPR/NF [22] and ENR/KW [8] for evaluating the TPR with imperfect nodes. The average hit rate is about 55.92%. The results show a great improvement on the execution time, especially for benchmark $G_{10:9}$.

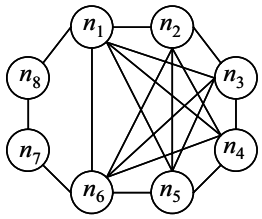

Figure 7. Benchmark network $G_{8:6}$.

Table 3. File distribution.

Node	Files	Node	Files
n_1	$F_1\ F_2\ F_3$	n_6	$F_6\ F_7\ F_8$
n_2	$F_2\ F_3\ F_4$	n_7	$F_1\ F_7\ F_8$
n_3	$F_3\ F_4\ F_5$	n_8	$F_1\ F_2\ F_8$
n_4	$F_4\ F_5\ F_6$	n_9	$F_3\ F_7\ F_8$
n_5	$F_5\ F_6\ F_7$	n_{10}	$F_1\ F_4\ F_7$

Table 4. The comparison of different algorithms for evaluating DPR.

Network	# of subgraphs			Time in seconds				# of SS hash hits	Hit ratio (%)	DPR
	KHR [1]	ENR/KW [8]	CLK	KHR [1]	ENR/KW [8]	ZST [9]	CLK			
$G_{8:4}$	37	16	25	0.030	0.005	0.05	0.02	3	10.71	0.891551
$G_{10:4}$	55	20	37	0.037	0.014	0.05	0.04	5	11.90	0.889355
$G_{8:6}$	306	72	55	0.412	0.015	0.06	0.05	12	17.91	0.898896
$G_{8:7}$	1159	289	63	4.28	0.055	0.07	0.06	17	21.25	0.899061
$G_{10:7}$	3443	462	175	9.85	0.148	0.08	0.11	65	27.08	0.899057
$G_{8:8}$	3225	1196	63	36.3	0.222	0.15	0.09	17	21.25	0.899090
$G_{10:8}$	20464	2556	223	230	0.817	0.25	0.21	101	31.17	0.899092
$G_{10:9}$	131899	17832	255	8000	5.64	1.83	0.42	129	33.59	0.899099

Table 5. The comparison of different algorithms for evaluating TPR.

Network	# of subgraphs			Time in seconds			# of *SS* hash hits	Hit ratio (%)	TPR
	TPR/NF [22]	ENR/KW [8]	CLK	TPR/NF [22]	ENR/KW [8]	CLK			
$G_{8:4}$	157	48	39	0.039	0.012	0.03	17	30.36	0.806942
$G_{10:4}$	365	226	75	0.157	0.065	0.05	65	46.43	0.701621
$G_{8:6}$	2823	1839	111	2.083	0.438	0.09	101	47.64	0.809823
$G_{8:7}$	33085	11883	127	26.0	2.90	0.23	129	50.39	0.809967
$G_{10:7}$	36513	27077	351	40.7	10.1	0.49	433	55.23	0.809964
$G_{8:8}$	383563	69918	127	230	18.8	0.84	129	50.39	0.809992
$G_{10:8}$	553431	234847	447	663	100	1.69	625	58.30	0.809993
$G_{10:9}$	10405589	2245128	511	13300	990	16.03	769	60.08	0.809999

Table 6. Different ordering strategies of CLK.

Network	# of Nodes in BDD		Time in seconds	
	CLK_BFS	CLK_ISO	CLK_BFS	CLK_ISO
$G_{8:4}$	56	28	0.03	0.02
$G_{10:4}$	60	36	0.04	0.04
$G_{8:6}$	416	74	0.07	0.05
$G_{8:7}$	1420	164	0.09	0.06
$G_{10:7}$	1424	192	0.17	0.11
$G_{8:8}$	3237	445	0.14	0.09
$G_{10:8}$	5327	535	0.39	0.21
$G_{10:9}$	22240	2177	1.10	0.42

Table 6 shows a comparison of the sizes of BDD and the execution times with different ordering strategies. CLK_BFS represents the CLK algorithm with breadth-first searching ordering strategy. CLK_ISO represents the CLK algorithm with the ordering strategy as described in Section 3.2. The result shows that a breadth-first variable-ordering obtained in [14][15][18] is not good enough for distributed computing networks. However, the heuristic ordering strategy depicted in Section 3.2 is a good candidate for DCS, especially in $G_{10:9}$.

6. Conclusions

In this paper, we have proposed an efficient algorithm for evaluating the reliability of a distributed computing system. The CLK algorithm can avoid redundant computations during the evaluation by converging isomorphic sub problems. The experimental results show that the CLK algorithm has a great improvement on the execution time compared to the previous works. Based on BDD, the CLK algorithm has the capability to support a large amount of Boolean manipulations and therefore, is applicable to large-scale distributed computing networks. In addition, using the BDD substitution technique, the CLK algorithm can be efficiently applied to a DCS network with imperfect nodes. Moreover, the BDD-based methodologies in [19] for the dependability analysis of systems can be integrated with the CLK algorithm. Based on this approach, researches on sensitivity analysis, importance measures, failure frequency analysis or optimal design issues of dis-

tributed processing systems will be the focus of our future works.

References

[1] V.K.P. Kumar, S. Hariri, and C.S. Raghavendra, "Distributed Program reliability analysis", *IEEE Trans. Software Engineering*, vol. 12, pp. 42-50, Jan. 1986.

[2] C.S. Raghavendra, V.K.P. Kumar, and S. Hariri, "Reliability analysis in distributed systems", *IEEE Trans. Computers*, vol. 37, no. 3, pp. 352-358, Mar. 1988.

[3] A. Kumar, S. Rai, and D.P. Agrawal, "On computer communication network reliability under program execution constraints", *IEEE Journal of Selected Areas in Communications*, vol. 6, pp. 1393-1400, Oct. 1988.

[4] S. Hariri and C.S. Ragavendra, "SYREL: A symbolic reliability algorithm based on path and cut set methods", *IEEE Trans. Computers*, vol. 36, pp. 1224-1232, Oct. 1987.

[5] A. Kumar and D.P. Agrawal, "A generalized algorithm for evaluating distributed-program reliability", *IEEE Trans. Reliability*, vol. 42, no. 3, pp. 416-426, Sept. 1993.

[6] D.J. Chen and T.H. Huang, "Reliability analysis of distributed systems based on a fast reliability algorithm", *IEEE Trans. Parallel and Distributed Systems*, vol. 3, no. 2, pp. 139-154, Mar. 1992.

[7] D.J. Chen, R.S. Chen, and T.H. Huang, "A heuristic approach to generating file spanning trees for reliability analysis of distributed computing systems", *Computers and Mathematics with Application*, vol. 34, pp. 115-131, Nov. 1997.

[8] W.J. Ke and S.D. Wang "Reliability evaluation for distributed computing networks with imperfect nodes", *IEEE Trans. Reliability*, vol. 46, no. 3, pp. 342-349, Sept. 1997.

[9] X. Zang, H. Sun, and K.S. Trivedi, "Dependability analysis of distributed computer systems with imperfect coverage", *Proc. 29th Ann. Int'l Conf. Fault-Tolerant Computing*, *(FTCS-29)*, 1999, pp. 330-337.

[10] M.S. Lin, M.S. Chang, and D.J. Chen, "Efficient algorithms for reliability analysis of distributed computing systems", *Information Sciences*, vol. 117, pp. 89-106, July 1999.

[11] C.D. Lai, M. Xie, K.L. Poh, Y.S. Dai, and P. Yang, "A model for availability analysis of distributed software/hardware systems", *Information and Software Technology*, vol. 44, no. 6, pp. 343-350, 2002.

[12] Y.S. Dai, M. Xie, and K.L. Poh, "Reliability analysis of grid computing systems", *Proc. 2002 Pacific Rim Int'l Symp. Dependable Computing (PRDC'02)*, 2002, pp. 97-104.

[13] R.E. Bryant, "Graph-based algorithms for Boolean function manipulation", *IEEE Trans. Computers*, vol. 35, pp. 677-691, Aug. 1986.

[14] S.Y. Kuo, S.K. Lu, and F.M. Yeh, "Determining terminal-pair reliability based on edge expansion diagrams using OBDD", *IEEE Trans. Reliability*, vol. 48, no. 3, pp. 234-246, Sept. 1999.

[15] F.M. Yeh, S.K. Lu, and S.Y. Kuo, "OBDD-based evaluation of *k*-terminal network reliability", *IEEE Trans. Reliability*, vol. 51, no. 4, pp. 443-451, Dec. 2002.

[16] S. Minato, "Streaming BDD manipulation", *IEEE Trans. Computers*, vol. 51, no. 5, pp. 474-485, May 2002.

[17] Hung-Yau Lin, Sy-Yen Kuo, and Fu-Min Yeh, "Minimal cutset enumeration and network reliability evaluation by recursive merge and BDD", *Proc. 8th IEEE Symp. Computers and Communications (ISCC'03)*, 2003, pp. 1341-1346.

[18] A. Rauzy, "A new methodology to handle Boolean models with loops", *IEEE Trans. Reliability*, vol. 52, no. 1, pp. 96-105, Mar. 2003.

[19] Y.R. Chang, S.V. Amari, and S.Y. Kuo, "Computing system failure frequencies and reliability importance measures using OBDD," *IEEE Trans. Computers*, Jan. 2004, (accepted).

[20] A. Rauzy, "New algorithms for fault tree analysis", *Reliability Engineering and System Safety*, vol. 40, pp. 203-211, 1993.

[21] R.M. Sinnamon, J.D. Andrews, "Improved efficiency in qualitative fault tree analysis", *Quality and Reliability Engineering Int'l*, vol. 13, pp. 293-298, 1997.

[22] V.A. Netes and B.P. Filin, "Consideration of node failures in network-reliability calculation", *IEEE Trans. Reliability*, vol. 45, no. 1, pp. 127-128, Mar. 1996.

[23] K.K Aggarwal, J.S. Gupta, and K.B. Misra, "A simple method for evaluation of a communication system," *IEEE Trans. Communications*, vol. 23, pp. 563-566, May 1975.