Applied Mathematics and Computation 217 (2010) 403–413

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Performance assessment and reliability analysis of dependable and distributed computing systems based on BDD and recursive merge

Yung-Ruei Chang [a], Chin-Yu Huang [b,*], Sy-Yen Kuo [c]

[a] Institute of Nuclear Energy Research, Atomic Energy Council, Taoyuan, Taiwan
[b] Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan
[c] Department of Electrical Engineering, National Taiwan University, Taipei, Taiwan

ARTICLE INFO

Keywords:
Reliability
OBDD
Fault-coverage
System availability
Distributed system

ABSTRACT

System reliability evaluation, sensitivity analysis, failure frequency analysis, importance measures, and optimal design are important issues that have become research topics for distributed dependable computing. Finding all of the Minimal File Spanning Trees (MFST) and avoiding repeatedly computing the redundant MFSTs have been key techniques for evaluating the reliability of a distributed computing system (DCS) in previous works. However, identifying all of the disjointed MFSTs is difficult and time consuming for large-scale networks. Although existing algorithms have been demonstrated to work well 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 these algorithms is inefficient when 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 the recursive merge and the binary decision diagram (BDD). Using the BDD substitution method, we can easily apply our algorithm to a network with imperfect nodes. The experimental results show a significant improvement in the execution time compared to previous works.

1. Introduction

The development of computer networking and embedded VLSI processing devices has led to an increased 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 a cost-effective means for resource sharing and extensibility, while providing potential increases in performance, reliability, and fault tolerance. A distributed program usually requires one or more resources for successful execution, such as PEs and data files. The local host (the PEs that contain the required files) and the interconnection links must all function correctly in order to successfully complete a program. 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, to efficiently estimate the effect of program and resource distributions on the reliability of a system. This analysis is crucially important for building a reliable distributed computing system.

Many researchers have studied distributed program reliability (DPR) and distributed system reliability (DSR). Kumar et al. [1] appears to be the first to present a definition for DPR and DSR. They constructed a distributed model including edges,

* Corresponding author.
E-mail address: cyhuang@cs.nthu.edu.tw (C.-Y. Huang).

0096-3003/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2010.05.075

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. A fast algorithm to evaluate the DPR and DSR was developed by Kumar [3]. These methods are 2-step algorithms. The first step is finding all of the MFSTs. The second step is converting these MFSTs into 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 of the MFSTs creates high computational complexity. Prior knowledge of multi-terminal connections is required in order to compute the reliability expression, thereby making these methods inapplicable for large systems. To overcome these problems, Kumar [5] proposed a 1-step algorithm, GEAR, which can avoid computing the redundant MFSTs and reduce computational time. To further improve the efficiency of reliability assessment, Chen [6,7] proposed the FST-SPR and HRFST algorithms based on the cut-set methods for reducing reliability evaluation complexity. However, applying their methods to a 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. The ENR/KW algorithm needs to find the set of mandatory nodes, and the isomorphic subproblems do not converge. Later, based on model [1], Zang [9] proposed a Binary Decision Diagram (BDD)-based algorithm to analyze the dependability of a DCS with imperfect fault-coverage. The research in [10] continued the study of DPR and DSR based on Kumar's model [1].

Previous works show that the key technique in evaluating the reliability of DCS is finding all the MFSTs and avoiding the computation of redundant MFSTs. However, identifying all of the disjointed MFSTs is difficult and 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 these methods is inefficient when dealing with larger Boolean functions. Second, the tree-based partitioning algorithm does not consider the convergence of isomorphic sub-problems, and therefore, redundant computations cannot be avoided.

Recent literature [11–15] shows that the BDD method is an 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 one, 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 [16].

Section 2 introduces the concepts of BDD and distributed computing systems. Section 3 illustrates an efficient algorithm based on recursive merge and BDDs to evaluate the DPR and DSR of a distributed computing system. Based on the BDD substitution technique, our algorithm is applicable not only to 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 provides the conclusions and future works.

2. Preliminaries

2.1. Binary decision diagram (BDD)

The BDD method [10] is based on a disjoint decomposition of a Boolean function called the *Shannon Expansion*. Given a Boolean function $f(x_1,\ldots,x_n)$, then for any $i \in \{1,\ldots,n\}; \bar{x}_i \equiv \neg x_i = 1 - x_i$:

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

In order to express the Shannon decomposition concisely, the if–then–else (*ite*) format [17,18] 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 the 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:

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

where $\diamondsuit$ 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 operations of BDDs, please refer to [10].