ELSEVIER

Information Processing Letters 64 (1997) 143-147

Information Processing Letters

The computational complexity of the reliability problem on distributed systems

Min-Sheng Lin [a,*], Deng-Jyi Chen [b,1]

[a] Department of Information Management, Tamsui Oxford University College, Tamsui, Taipei, 25103, Taiwan, ROC
[b] Institute of Computer Science and Information Engineering, National Chiao-Tung University, Hsin Chu, 30050, Taiwan, ROC

Received 31 January 1997; revised 16 July 1997
Communicated by T. Asano

Abstract

The reliability of a distributed program in a distributed computing system is the probability that a program which runs on multiple processing elements and needs to communicate with other processing elements for remote data files will be executed successfully. This reliability varies according to (1) the topology of the distributed computing system, (2) the reliability of the communication links, (3) the data files and program distribution among processing elements, and (4) the data files required to execute a program. This paper shows that solving this reliability problem is NP-hard even when the distributed computing system is restricted to a series-parallel, a 2-tree, a tree, or a star structure. © 1997 Elsevier Science B.V.

Keywords: Distributed systems; Distributed program reliability; Computational complexity; Graph theory

1. Introduction

A typical distributed computing system (DCS) consists of processing elements (nodes), communication links (edges), memory units, data files, and programs [5,6]. These resources are interconnected via a communication network that dictates how information flows between nodes. Programs residing on some nodes can run using data files at other nodes.

One important issue in the design of a DCS is reliability. A large amount of work [1,8,12,16] has been devoted to developing algorithms to compute measures of reliability for DCSs. One typical reliability measure for DCSs is the K-terminal reliability (KTR) [20]. KTR is the probability that a specified set of nodes K, which is subset of all the nodes in the DCS, remains connected in a DCS whose edges may fail independently of each other, with a known probabilities. However, the KTR measure is not applicable to practical DCSs since a reliability measure for DCSs should capture the effects of redundant distribution of programs and data files.

In [10], distributed program reliability (DPR) was introduced to accurately model the reliability of DCSs. Consider DCS in which the nodes are perfectly reliable but the edges can fail, statistically independently of each other, with known probabilities. For successful execution of a distributed program, it is essential that the node containing the program, other nodes that have required data files, and the edges between them be

* Corresponding author. Email: mlin@jupiter.touc.edu.tw.
[1] Email: djchen@csie.nctu.edu.tw.

PII S0020-0190(97)00150-6

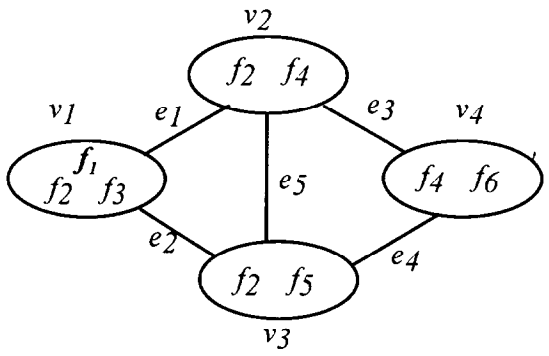

Fig. 1. A simple DCS. Program f_1 needs data files f_2, f_3, and f_4 to complete execution.

operational. DPR is thus defined as the probability that a program with distributed files can run successfully in spite of some faults occurring in the edges. To illustrate the definition of DPR, consider the DCS shown in Fig. 1. There are four nodes (v_1, v_2, v_3, v_4) and five edges (e_1, e_2, e_3, e_4, e_5). Program f_1 requires data files f_2, f_3, and f_4 to complete execution, and it is running at node v_1, which holds data files f_2 and f_3. Hence, it must access data file f_4, which is resident at both node v_2 and node v_4. Therefore, the reliability of distributed program f_1 can be formulated as follows:

DPR(program f_1)

= Prob((v_1 and v_2 are connected)

or (v_1 and v_4 are connected)).

Several algorithms have been proposed for evaluation DPR [3,4,9,11]. However, we have seen that none meets our desire for efficient algorithms. At this point, one must conclude that either the approaches examined are just not sufficiently clever, or that no efficient algorithms exist for our reliability problems. Nevertheless, tools in complexity theory do provide a vehicle for giving strong evidence that no polynomial time algorithm exists for certain problems. A generally accepted method for providing evidence of intractability is to prove NP-complete or NP-hard. We refer the reader to [7] for an excellent exposition of the theory of NP-complete, and for proofs of standard NP-complete results. NP-hard is not a proof of intractability, but is convincing evidence. An efficient solution for any NP-hard problem provides efficient methods for every problem in NP, which contains a formidable list of apparently difficult problems.

The purpose of this paper is to show that the DPR problem, in general, is NP-hard even on a DCS with a series-parallel, a 2-tree, a tree, or a star structure.

In this paper we will make use of the following notation:

- $D = (V, E, F)$: an undirected DCS graph with vertex (nodes) set V, edge set E, and file set F which is distributed in D,
- V: the set of nodes that are all perfectly reliable,
- E: the set of edges that can fail, statistically independently of each other, with known probability,
- F: the set of files (including data files and programs) distributed in D,
- $H \subseteq F$: the specified set of files that must communicate with each other through the edges,
- $FA_i \subseteq F$: the set of files available at node i,
- p_i: the reliability of edge i,
- $q_i \equiv 1 - p_i$,
- $R(D_H)$: the DPR of D with H specified $\equiv$ the probability that all files in H can communicate with each other through the edges in D.

Using the above notation, we can describe the example in Fig. 1 as follows:

$V = \{v_1, v_2, v_3, v_4\}$,

$E = \{e_1, e_2, e_3, e_4, e_5\}$,

$F = \{f_1, f_2, f_3, f_4, f_5, f_6\}$,

$H = \{f_1, f_2, f_3, f_4\}$,

//Program f_1 needs data files f_2, f_3, and f_4 to complete execution. //

$FAv_1 = \{f_1, f_2, f_3\}$, $FAv_2 = \{f_2, f_4\}$,

$FAv_3 = \{f_2, f_5\}$, and $FAv_4 = \{f_4, f_6\}$.

2. The computational complexity of the DPR problem

Complexity results are obtained by transforming known NP-hard problems into our reliability problems. For this reason, we first state some known NP-hard problems:

(1) *K-Terminal Reliability* (KTR) [15,18].

Input: an undirected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges that fail statistically independently of each other

with known probabilities. A set $K \subseteq V$ is distinguished with $|K| \geqslant 2$.
Output: $R(G_K)$, the probability that the set K of nodes of G is connected in G.

(2) *Number of Edge Covers* (#EC) [2].
Input: an undirected graph $G = (V, E)$.
Output: the number of edge covers for $G \equiv |\{L \subseteq E : \text{ each node of } G \text{ is an end of some edge in } L\}|$.

(3) *Number of Vertex Covers* (#VC) [14].
Input: an undirected graph $G = (V, E)$.
Output: the number of vertex covers for $G \equiv |\{K \subseteq V : \text{ every edge of } G \text{ has at least one end in } K\}|$.

Theorem 1. *Computing DPR for a general DCS is* NP-*hard.*

Proof. We reduce the well-known KTR problem to our DPR problem. For a given network $G = (V, E)$ and a specified set $K \subseteq V$, we can define an instance of the DPR problem. Construct a DCS graph $D = (V, E, F)$ in which the topology and the reliability of each edge are the same as G. Let $F = \bigcup_{\text{node } i \in K}\{f_i\}$ and $FA_i = \{f_i\}$ if node $i \in K$, else $FA_i = \emptyset$ for each node $i \in V$. If we set $H = F = \bigcup_{\text{node } i \in K}\{f_i\}$, then we have $R(D_H) = R(G_K)$. □

Corollary 2. *Computing DPR for a planar DCS is* NP-*hard.*

Proof. From the proof of Theorem 1, it is clear that the KTR problem is just a special case of the DPR problem. It has been shown that computing KTR over planar networks is NP-hard [13]. This also immediately implies that computing DPR over planar networks is NP-hard. □

The result of Theorem 1 implies that it is unlikely that polynomial-time algorithms exist for solving the DPR problem. One possible means of avoiding this complexity is to consider only a restricted class of structures. Classes of interest here include linear systems, which are widely used in bus local networks, ring systems, which are widely used in token ring local networks, stars, which are used in one-node circuit-switched networks, trees, which are used in hierarchical local access networks, and series-parallel system which arise in wide-area networks.

For the KTR problem polynomial-time (or linear-time) algorithms have been developed for other restricted networks, such as linear systems, ring systems, stars, trees, and series-parallel graphs [17]. Obviously, if there are no replicated files, i.e., if there is only one copy of each file in the DCS, then the DPR problem can be transformed into a KTR equivalent problem in which the K set is the set of nodes that contain the data files needed for the program under consideration. However, data files are usually replicated and distributed in DCS, so these two problems are different. In the remainder of this section, we will show that computing DPR over stars, trees, or series-parallel networks in general is still NP-hard.

Theorem 3. *Computing DPR for a DCS with a star topology is* NP-*hard even when each* $|FA_i| = 2$.

Proof. We reduce the #EC problem to our problem. For a given network $G = (V_1, E_1)$, where $E_1 = \{e_1, e_2, \ldots, e_n\}$, we construct a DCS $D = (V_2, E_2, F)$ with a star topology, where $V_2 = \{s, v_1, v_2, \ldots, v_n\}$, $E_2 = \{(s, v_i) \mid 1 \leqslant i \leqslant n\}$, and $F = \{f_i \mid \text{ for each node } i \in G\}$. Let $FAv_i = \{f_u, f_v \mid \text{ if } e_i = (u, v) \in G\}$ for $1 \leqslant i \leqslant n$, $FA_s = \emptyset$ and $H = F$. In the DCS we now define a file spanning tree (FST), which is a tree whose nodes hold all files $\in H$, i.e., $H \subseteq \bigcup_{\forall i \in FST}\{FA_i\}$. From the construction of D, it is easy to show that there is a one-to-one correspondence between one of the sets of edge covers and one FST. The DPR of D, $R(D_H)$, can be expressed as

$$R(D_H) = \sum_{\text{for all FST } t \in D} \left\{ \prod_{\text{for each edge } i \in t} p_i \times \prod_{\text{for each edge } i \notin t} (1 - p_i) \right\}.$$

If we set each $p_i = \frac{1}{2}$ for all $1 \leqslant i \leqslant n$, then we have

$$R(D_H) = \sum_{\text{for all FST } t \in D} (\tfrac{1}{2})^n$$

$$R(D_H)2^n = \sum_{\text{for all FST } t \in D} 1$$

$$= \#\text{ of FSTs in } D$$

$$= \#\text{ of edge covers in } G. \quad \square$$

Theorem 4. *Computing DPR for a DCS with a star topology is* NP*-hard even when there are only two copies of each file.*

Proof. We employ the reduce from #VC problem to our problem. For a given $G = (V_1, E_1)$, where $|E_1| = n$ and $V_1 = \{v_1, v_2, \ldots, v_m\}$, we construct a DCS $D = (V_2, E_2, F)$ with a star topology, where $V_2 = V_1 \cup \{s\}$, $E_2 = \{e_i = (s, v_i) \mid 1 \leqslant i \leqslant m\}$, and $F = \{f_i \mid$ for all edges $i \in G\}$. Let $FA_i = \{f_j \mid$ for all edges j that are incident on $v_i \in G\}$ and $H = F$. From the construction of D, it is easy to show that there are only two copies of each file in D and there is a one-to-one correspondence between one of the sets of vertex covers and one FST of D. The DPR of D, $R(D_H)$, can be expressed as

$$R(D_H) = \sum_{\text{for all FST } t \in D} \Bigg\{ \prod_{\text{for each edge } i \in t} p_i \times \prod_{\text{for each edge } i \notin t} (1 - p_i) \Bigg\}.$$

If we set each $p_i = \frac{1}{2}$ for all $1 \leqslant i \leqslant n$, then we have

$$R(D_H) = \sum_{\text{for all FST } t \in D} (\tfrac{1}{2})^n,$$

$$R(D_H)2^n = \sum_{\text{for all FST } t \in D} 1$$

$$= \#\text{ of FSTs in } D$$

$$= \#\text{ of vertex covers in } G. \quad \square$$

Corollary 5. *Computing DPR for a DCS with a tree topology is* NP*-hard.*

Proof. By Theorems 3 and 4, we see that the DPR problem for a DCS with a star topology in general is NP-hard. This implies that the DPR problem for a DCS with a tree topology in general is also NP-hard, since a DCS with a star topology is just a DCS with a tree topology which has one level branch. $\square$

For KTR, it is obviously true that polynomial-time algorithms exist over DCSs with a star or a tree topology. In addition, polynomial-time algorithms do exist for computing KTR over series-parallel graphs [14] and 2-trees [13]. A 2-*tree* is defined recursively as follows:

(1) The complete graph K2 (a single edge) is a 2-tree.
(2) Given any 2-tree G on $n \geqslant 2$ nodes, let (u, v) be an edge of G. Adding a new node w and two edges (w, u) and (w, v) produces a 2-tree on $n + 1$ nodes.

We now show that the DPR problem for a DCS with a 2-tree structure in general is NP-hard.

Theorem 6. *Computing DPR for a DCS with a* 2*-tree topology in general is* NP*-hard.*

Proof. We reduce an arbitrary instance of a star topology to a 2-tree topology. Assume we have a DCS graph $D = (V, E, F)$ where $V = \{s, v_1, v_2, \ldots, v_m\}$ and $E = \{(s, v_i) \mid 1 \leqslant i \leqslant n\}$ with a star topology. We construct from D a DCS graph $D' = (V, E', H)$, where $E' = E \cup \{(v_i, v_{i+1}) \mid 1 \leqslant i \leqslant n - 1\}$. It is easy to see that D' is a 2-tree on $n + 1$ nodes. If we stipulate that all added edges (v_i, v_{i+1}), $1 \leqslant i \leqslant n - 1$, of D' have a reliability of 0, then we have $R(D_H) = R(D'_H)$ for any given $H \subseteq F$. $\square$

Corollary 7. *Computing DPR over a series-parallel DCS is* NP*-hard.*

Proof. From [19], a 2-tree is a maximal series-parallel graph. A maximal series-parallel graph is a series-parallel graph with neither loops nor parallel edges. Since computing DPR over a DCS with a 2-tree topology is NP-hard, computing DPR over a series-parallel DCS is also NP-hard. It is easy to see that the DCS graph D' constructed in Theorem 6 is also a series-parallel DCS. The theorem follows. $\square$

In this section, we have shown that computing DPR over a DCS with a star, a tree, a 2-tree, a series-parallel, a planar, or a general topology in general is NP-hard.

3. Conclusions

The reliability of a distributed program in a distributed computing system is the probability that a program which runs on multiple processing elements and needs to communicate with other processing elements for remote data files will be executed successfully. This reliability varies according to (1) the topol-

ogy of the distributed computing system, (2) the reliability of the communication links, (3) the data files and program distribution among processing elements, and (4) the data files required to execute a program. This paper shows that solving this reliability problem is NP-hard even when the distributed computing system is restricted to a series-parallel, a 2-tree, a tree, or a star structure.

References

[1] K.K. Aggrawal, S. Rai, Reliability evaluation in computer-communication networks, IEEE Trans. Reliability 30 (1981) 32–35.

[2] M.O. Ball, J.S. Provan, D.R. Shier, Reliability covering problems, Networks 21 (1991) 345–357.

[3] D.J. Chen, T.H. Huang, Reliability analysis of distributed systems based on a fast reliability algorithm, IEEE Trans. Parallel Distributed Systems 3 (2) (1992) 139–153.

[4] D.J. Chen, M.S. Lin, On distributed computing systems reliability analysis under program execution constraints, IEEE Trans. Comput. 15 (12) (1993).

[5] P. Enslow, What is a distributed data processing system, Computer 11 (1978).

[6] J. Garcia-Molina, Reliability issues for fully replicated distributed database, IEEE Comput. 16 (1982) 34–42.

[7] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman, San Francisco, CA, 1979.

[8] A.P. Grnarov, M. Gerla, Multi-terminal reliability analysis of distributed processing system, in: Proc. 1981 Internat. Conf. Parallel Processing (1981) 79–86.

[9] A. Kumar, S. Rai, D.P. Agrawal, On computer communication network reliability under program execution constraints, IEEE JSAC 6 (1988) 1393–1399.

[10] V.K. Prasanna Kumar, S. Hariri, C.S. Raghavendra, Distributed program reliability analysis, IEEE Trans. Software Engineering 12 (1986) 42–50.

[11] M.S. Lin, D.J. Chen, General reduction methods for the reliability analysis of distributed computing systems, Comput. J. 36 (7) (1993) 631–644.

[12] R.E. Merwin, M. Mirhakak, Derivation and use of a survivability criterion for DDP systems, in: Proc. 1980 Nat. Comput. Conf. (1980) 139–146.

[13] J.S. Provan, The complexity of reliability computations in planar and acyclic graphs, SIAM J. Comput. 15 (1986) 694–702.

[14] J.S. Provan, M.O. Ball, The complexity of counting cuts and of computing the probability that a graph is connected, SIAM J. Comput. 12 (4) (1983) 777–788.

[15] A. Rosenthal, A computer scientist looks at reliability computations, in: Reliability and Fault tree Analysis SLAM (1975) 133–152.

[16] A. Satyanarayana, J.N. Hagstrom, A new algorithm for the reliability analysis of multi-terminal networks, IEEE Trans. Reliability 30 (1981) 325–334.

[17] A. Satyanarayana, R.K. Wood, A linear-time algorithm for computing K-terminal reliability in series-parallel networks, SIAM J. Comput. 14 (4) (1985) 818–832.

[18] L.G. Valiant, The complexity of enumeration and reliability problems, SIAM J. Comput. 8 (1979) 410–421.

[19] P. Winter, Steiner problem in networks: A survey, Networks 17 (1987) 129–167.

[20] R.K. Wood, Factoring algorithms for computing K-terminal network reliability, IEEE Trans. Reliability 35 (1986) 269–278.