PAPER

A Hierarchical Approach to Dependability Evaluation of Distributed Systems with Replicated Resources

Eun Hye CHOI†, *Nonmember*, **Tatsuhiro TSUCHIYA**†, *and* **Tohru KIKUNO**†, *Regular Members*

SUMMARY We propose a two-level hierarchical method for dependability evaluation of distributed systems with replicated programs and data files. Since Markov modeling is limited only to each component in this method, state explosion can be circumvented successfully. Simulation results show that the method can accomplish evaluation even for large systems for which Markov modeling is not feasible.

key words: *distributed systems, dependability evaluation, replicated resources, Markov models, fault trees*

1. Introduction

In distributed systems, programs and data files are often replicated and allocated to multiple nodes in a redundant manner in order to achieve fault tolerance and high dependability. Thus it is essential to capture the effects of such redundant distribution of resources for dependability analysis. For this purpose, Kumar et al. proposed an analytic model of distributed systems [11]. In the model, a program is assumed to be executable when at least one node that can execute the program remains operational and all files required for the execution of the program are accessible from the node. Based on this model, several algorithms have been proposed to evaluate the availability of systems [1], [8], [11], [12]. However, they do not explicitly consider the failure-repair behavior of system components and thus, they can evaluate the system dependability only in a static manner.

To model dynamic behavior of components, Lopez-Benitez proposed a Petri-net-based method [10]. In this method, the behavior of a distributed system is represented by a stochastic Petri-net. In [10], two typical repair models, called the *local repair model* and the *global repair model*, are proposed and discussed. Since failures and repairs of system components are explicitly considered in these two models, dependability measures other than availability can be evaluated. In addition, the *coverage* for each component is explicitly taken into consideration in these models. Coverage means the probability that handling of a component failure, which usually involves detection and system reconfiguration, is completed successfully. If the handling is not successful, then the component failure causes system failure even though other components remain operational. We call such a component failure a *noncovered* failure. It has been shown that the system dependability is highly sensitive to the coverage factor [3], [4].

Unfortunately, this Petri-net-based method is prone to suffer from state explosion, which often occurs when Markov models are used. Although the use of a stochastic Petri-net is known as a common way to circumvent state explosion in representing the failure-repair model, for evaluating dependability measures, it requires to generate all states of the underlying Markov chain. Consequently, the state explosion problem cannot be avoided. As shown later, the number of states generated from the Petri-net model in [10] becomes very large even when the number of nodes in the system is less than 15.

As an alternative approach, a hierarchical method is proposed in [6]. In this method, the availability of each component is computed by using a Markov model. Based on the availability of each component, the availability of the system is computed by the above mentioned algorithms [1], [8], [11], [12]. Since this method necessitates the assumption that the behavior of each component is independent of others, it can deal with neither the coverage factor nor global repair.

To cope with the defects of the previous methods, we propose a new evaluation method in this paper. Assuming the global repair model in [10], the proposed method models the system as a two-level hierarchical structure. At the lower level (component level), the behavior of each component is described by a Markov model. At the higher level (system level), a fault tree is used to model the behavior of the whole system. Unlike in [6], the coverage factor is explicitly taken into account at both of the two levels. Once a model has been constructed, a software tool called SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) [13], [14] can be used to analyze the model and calculate some dependability measures. Since Markov modeling is localized to each component, the state explosion problem is circumvented in the proposed method.

Manuscript received November 30, 1999.
Manuscript revised September 11, 2000.
†The authors are with the Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University, Toyonaka-shi, 560–8531 Japan.

2. Preliminaries

2.1 System Model

We consider a distributed system modeled by an undirected graph $G = (V, E)$, where each vertex $x_i \in V$ represents a computing node and each edge $x_{i,j} \in E$ represents a communication link between node x_i and node x_j. We assume that G is connected and has no self-loop and no parallel edges. We use the term *component* to indicate a node or a link. Additionally, we define S as the set of all the components (nodes $x_i \in V$ and links $x_{i,j} \in E$) of the distributed system G, i.e., $S = V \cup E$. In the following, we will often use $s_i \in S(i = 1, 2, \cdots, |V| + |E|)$ to indicate a component x_i or $x_{i,j}$ for simplicity.

We assume that each component $s_i (\in S)$ is either fully operational or completely failed, and that its times to failure are exponentially distributed with parameter λ_i. Usually, this parameter is referred to as the *failure rate*. For each component s_i, the coverage is given by $c_i (0 \leq c_i \leq 1)$. This means that if s_i fails, then the system also fails with probability $1 - c_i$ due to unsuccessful handling of the component failure.

Let $\mathcal{P} = \{P_1, P_2, \cdots, P_{|\mathcal{P}|}\}$ and $\mathcal{F} = \{F_1, F_2, \cdots, F_{|\mathcal{F}|}\}$ denote a set of programs and a set of data files in the system, respectively. We assume that the programs and the data files are distributed throughout the nodes of the system in a redundant manner. Let $PRG_i (\subseteq \mathcal{P})$ denote the set of programs that node x_i can execute, and let $FA_i (\subseteq \mathcal{F})$ denote the set of data files located on node x_i. If node x_i can execute program P_j, i.e., $P_j \in PRG_i$, then we call the node x_i a *host node* of P_j. Let $FN_i (\subseteq \mathcal{F})$ denote the set of data files needed for the execution of program P_i. Suppose that x_j is a host node of P_i and data file $F \in FN_i$ is located on x_k. If x_j and x_k are operational, and there is an operational path from x_j to x_k, then we say that F is accessible from x_j. We assume that the program P_i is executable if and only if at least one host node of P_i is operational, and every file in FN_i is accessible from the host node. We also assume that if any program in $\mathcal{P}$ becomes not executable, then the system fails. That is, the system fails if a program becomes not executable, or a noncovered component failure occurs.

Concerning repair, we assume the *global repair model* which is discussed in [10]. In the global repair model, a repair action is taken only when the system fails. In this model, any repair action is performed in a centralized manner, and the whole system is restored to its initial status after the repair. We assume that the mean time to repair, MTTR, is given. Unlike in [10], the assumption of the exponentially distributed repair times is not made in this paper.

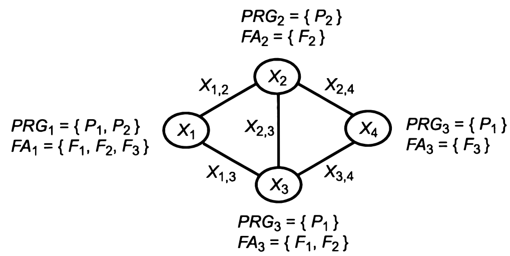


Fig. 1 Distributed system with four nodes.

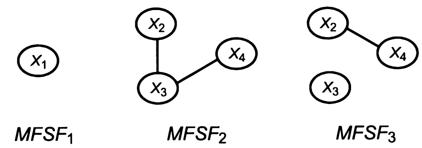


Fig. 2 All MFSFs in the system in Fig. 1.

2.2 Minimal File Spanning Forest (MFSF)

In general, the set of nodes and links involved in the execution of a program forms a tree in G. In [11], such a tree is called a *File Spanning Tree* (FST). For a program P_i, an FST is defined as a tree that contains a host node for P_i and holds all files in FN_i in some of its nodes.

Example 1: Consider the system shown in Fig. 1. Let $\mathcal{P} = \{P_1, P_2\}, FN_1 = \{F_1, F_2\}$, and $FN_2 = \{F_2, F_3\}$. In a tree $(\{x_2, x_4\}, \{x_{2,4}\})$, node x_2 can run program P_2, and every file in FN_2 is held by x_2 or x_4. Hence, this tree is an FST for P_2. It is clear that if all components of this tree are operational, then P_2 is executable. Similarly, $(\{x_3\}, \emptyset\})$ is an FST for P_1.

In contrast, the subgraphs that will provide required paths for the execution of all the programs do not always form trees because they are collections of FSTs associated with the programs. To efficiently calculate the probability that such subgraphs remain operational, the notion of a *Minimal File Spanning Forest* (MFSF) is introduced in [12]. An MFSF is defined as a subgraph of G such that FSTs for all programs are contained in the subgraph but none of its proper subgraphs has this property.

Example 2: In the system in Fig. 1, a subgraph $G' = (\{x_2, x_3, x_4\}, \{x_{2,4}, x_{3,4}\})$ of G contains FSTs for all the programs in $\mathcal{P}$. However, this subgraph is not an MFSF since its proper subgraph $G'' = (\{x_2, x_3, x_4\}, \{x_{2,4}\})$ is an MFSF. All MFSFs in this example are shown in Fig. 2. (G'' corresponds to $MFSF_3$ in Fig. 2.)

By definition, all the programs are executable if and only if an operational MFSF exists. Hence, the system is operational if and only if at least one MFSF is operational and no noncovered failure has occurred on any component of the system†.

2.3 Dependability Measures

As dependability measures to be evaluated, we consider reliability $R(t)$, Mean-Time-To-Failure(MTTF), and steady-state availability A.

The reliability of a system is its ability to function correctly over a specified period of time. Formally the reliability at a given time t, $R(t)$, can be expressed as

$$R(t) = \Pr(\text{the system is operational in } [0, t]).$$

Once $R(t)$ has been obtained, we can compute the MTTF as follows:

$$\text{MTTF} = \int_0^\infty R(t)dt.$$

Steady-state availability A, which is the probability that the system is operational when sufficiently long time elapses, is known as a suitable measure for evaluation of systems experiencing a number of failures and repairs. This measure A is formally defined as

$$A = \lim_{t\to\infty} \Pr(\text{the system is operational at } t).$$

In this paper, we assume that a repair action is taken when the system fails, and that the system is restored to the initial status after repair. Hence, if the MTTF and the mean time to repair (MTTR) of the system are given, then A is computed as

$$A = \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}.$$

3. Proposed Evaluation Method

The proposed method constructs a two-level hierarchical model of a distributed system so that software tool SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) [13], [14] can analyze the model. As the name suggests, SHARPE supports hierarchical model composition. By using this tool, we can estimate the dependability measures.

The proposed method has three steps listed below.

Step 1. Enumerate all MFSFs.
Step 2. Construct a hierarchical model, which consists of the component level (the lower level) and the system level (the upper level). The MFSFs enumerated in Step 1 are used in modeling at the system level.
Step 3. Evaluate the dependability measures by applying SHARPE to the constructed model in Step 2.

We use an algorithm proposed in [11] for MFSF enumeration in Step 1. This algorithm first enumerates FSTs for all the programs in $\mathcal{P}$, and then generates MFSFs by merging them. (Though another algorithm to generate MFSFs is proposed in [12], we found that this algorithm can produce incorrect results. See the appendix for details.) In the rest of this section, we describe Step 2 and Step 3.

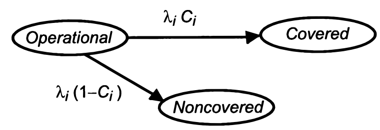


Fig. 3 Markov model for a component s_i.

3.1 Hierarchical Model Construction (Step 2)

3.1.1 Modeling at the Component Level

At the lower level, a Markov model is used for each component. The behavior of component s_i is represented by the continuous-time Markov chain illustrated in Fig. 3. In this figure, state 'Operational' means that s_i is operational. Two other states represent the fact that s_i has failed. State 'Covered' means that s_i failed and the failure was handled successfully, while state 'Noncovered' means that the failure was noncovered. Transitions between states are expressed by arrows, each of which is associated with its transition rate.

In the following, we use O_i and N_i to represent the event that s_i is in state 'Operational' and the event that s_i is in state 'Noncovered', respectively. These events are used as its inputs to the model at the upper level.

3.1.2 Modeling at the System Level

At the higher level, we model the whole system by using a fault tree. As mentioned before, the system is operational if and only if at least one MFSF remains operational and no noncovered component failure has occurred. In other words, the system is not operational if and only if none of the MFSFs is operational or a noncovered component failure has occurred on at least one component. This is represented by the fault tree shown in Fig. 4. Below, we explain modeling at the system level by using this figure.

As stated before, MFSFs enumerated in Step 1 are used to construct this fault tree. Let $M = \{MFSF_1, MFSF_2, \cdots, MFSF_{|M|}\}$ denote the set of all the MFSFs. In the fault tree, OR gate A_i corresponds to the ith MFSF, $MFSF_i$, and thus inputs to the OR gate correspond to the components of the MFSF. Each of these inputs, denoted as $\overline{O_j}$, represents

†Under the assumption described in Sect. 2.1, even for a component not in MFSFs, its noncovered failure causes system failure. Though this assumption is the same as [10], it may be impractical in some situations. Thus we mention this point in Sect. 6.

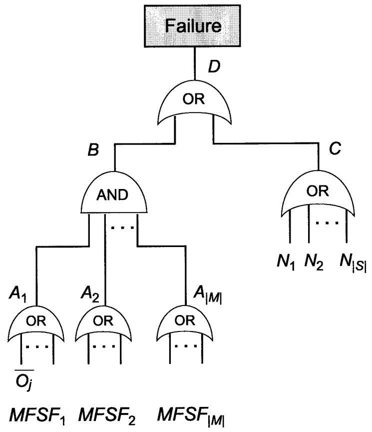


Fig. 4 Fault tree at the system level.

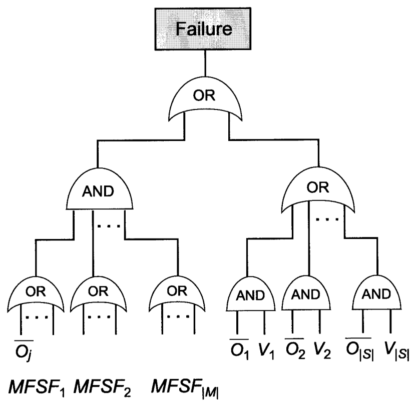


Fig. 5 Modified fault tree.

the event that component s_j of $MFSF_i$ is not operational. Then, the output of gate A_i represents the event that $MFSF_i$ is not operational. (If the MFSF is composed of one component, then the OR gate is not necessary.) Therefore, the output of AND gate B represents the event that none of the MFSFs in M is operational.

Each input to OR gate C, denoted as N_i, corresponds to component s_i of the system, and it represents the event that a noncovered component failure has occurred on s_i. Thus, the output of OR gate C represents the event that a noncovered failure has occurred on at least one component. Hence the system is not operational if and only if the output of AND gate B or that of OR gate C is true. As a consequence, the output of the top OR gate D represents the event that the whole system has failed.

3.1.3 Modification of the Fault Tree

In the proposed method, we analyze of the constructed model by using software tool SHARPE. To do so, some minor modifications of the fault tree are required. In SHARPE, input events of a fault tree must be independent of each other. In contrast, the fault tree in Fig. 4 contains dependent events such as $\overline{O_i}$ and N_i for each $i(= 1, 2, \cdots, |S|)$.

In order to satisfy this requirement, we modify the original fault tree by introducing virtual events $V_i(i = 1, 2, \cdots, |S|)$. Let V_i be a virtual event that always occurs with probability $1 - c_i$. Due to the nature of the Markov model at the component level in Fig. 3, it is known that event N_i occurs with probability $1 - c_i$ whenever event O_i does not occur. Therefore, the event N_i can be replaced with an AND gate with two inputs $\overline{O_i}$ and V_i. After the modifications, the fault tree shown in Fig. 5 is obtained from the fault tree shown in Fig. 4.

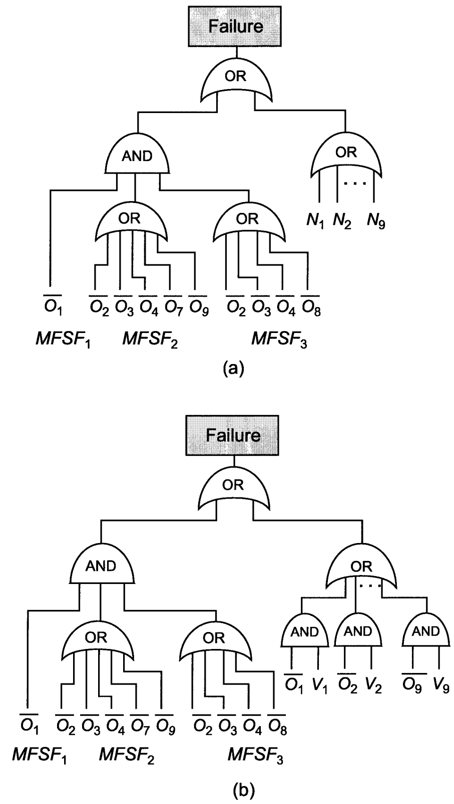


Fig. 6 (a) Fault tree for the system in Example 1, and (b) modified one.

Example 3: Figure 6 (a) shows the fault tree for the system shown in Fig. 1, where $s_i = x_i (i = 1, 2, 3, 4)$, $s_5 = x_{1,2}$, $s_6 = x_{1,3}$, $s_7 = x_{2,3}$, $s_8 = x_{2,4}$, and $s_9 = x_{3,4}$. With modification to this fault tree, the new fault tree shown in Fig. 6 (b) is obtained.

3.2 Dependability Evaluation by SHARPE (Step 3)

Once a model is constructed, dependability measures $R(t)$, MTTF, and A, which are described in Sect. 2.3, are evaluated by SHARPE. In SHARPE, a model to be analyzed must be described as an input file written in the SHARPE language. SHARPE analyzes the input file and outputs symbolic expressions for reliability $R(t)$ and the value of MTTF. Once MTTF is obtained,

```
markov law(lam,c)
       operational covered lam*c
       operational noncovered lam*(1-c)
end
       operational 1.0
end
bind  lam1    0.001000
bind  c1      0.950000
bind  lam2    0.001000
bind  c2      0.950000
bind  lam3    0.001000
bind  c3      0.950000
bind  lam4    0.001000
bind  c4      0.950000
bind  lam5    0.001000
bind  c5      0.950000
bind  lam6    0.001000
bind  c6      0.950000
bind  lam7    0.001000
bind  c7      0.950000
bind  lam8    0.001000
bind  c8      0.950000
bind  lam9    0.001000
bind  c9      0.950000
ftree failure
      repeat  nonop1  cdf(law;lam1,c1)
      repeat  nonop2  cdf(law;lam2,c2)
      repeat  nonop3  cdf(law;lam3,c3)
      repeat  nonop4  cdf(law;lam4,c4)
      basic   nonop5  cdf(law;lam5,c5)
      basic   nonop6  cdf(law;lam6,c6)
      repeat  nonop7  cdf(law;lam7,c7)
      repeat  nonop8  cdf(law;lam8,c8)
      repeat  nonop9  cdf(law;lam9,c9)
      basic   v1      gen    1-c1,0,0
      basic   v2      gen    1-c2,0,0
      basic   v3      gen    1-c3,0,0
      basic   v4      gen    1-c4,0,0
      basic   v5      gen    1-c5,0,0
      basic   v6      gen    1-c6,0,0
      basic   v7      gen    1-c7,0,0
      basic   v8      gen    1-c8,0,0
      basic   v9      gen    1-c9,0,0
      or      mfsf1   nonop1
      or      mfsf2   nonop2 nonop3 nonop4 nonop7 nonop9
      or      mfsf3   nonop2 nonop3 nonop4 nonop8
      and     d1 nonop1 v1
      and     d2 nonop2 v2
      and     d3 nonop3 v3
      and     d4 nonop4 v4
      and     d5 nonop5 v5
      and     d6 nonop6 v6
      and     d7 nonop7 v7
      and     d8 nonop8 v8
      and     d9 nonop9 v9
      or      dect    d1 d2 d3 d4 d5 d6 d7 d8 d9
      and     mfsf    mfsf1 mfsf2 mfsf3
      or TOP mfsf dect
end
cdf(failure)
end
```

Fig. 7 Input file to SHARPE.

steady-state availability A is easily calculated by equation $A = \mathrm{MTTF}/(\mathrm{MTTF}+\mathrm{MTTR})$, which has already been described in Sect. 2.3.

```
CDF for system failure:
    1.0000e+00 t(  0) exp( 0.0000e+00 t)
 + -6.6342e-01 t(  0) exp(-1.0000e-03 t)
 + -2.7933e-01 t(  0) exp(-2.0000e-03 t)
 + -5.1456e-02 t(  0) exp(-3.0000e-03 t)
 + -7.7920e-01 t(  0) exp(-4.0000e-03 t)
 + -2.0398e-01 t(  0) exp(-5.0000e-03 t)
 +  1.6933e+00 t(  0) exp(-6.0000e-03 t)
 + -6.3288e-01 t(  0) exp(-7.0000e-03 t)
 + -8.0780e-02 t(  0) exp(-8.0000e-03 t)
 + -2.2503e-03 t(  0) exp(-9.0000e-03 t)
mean: 8.7438e+02
variance: 7.6146e+05
```

Fig. 8 Output of SHARPE.

Example 4: Consider the system shown in Fig. 1 again and assume MTTR = 10. Figure 7 shows an input file representing the hierarchical model constructed by the proposed method, where $\lambda_i = 0.001$ and $c_i = 0.95$ for any component s_i. Analyzing this input file, SHARPE computes reliability and MTTF. Figure 8 shows the output for the input file. In the output, "CDF for system failure" means the cumulative distribution function for system failure, i.e., $1 - R(t)$. From the result, we can get the reliability $R(t)$, the MTTF, and the steady-state availability A of the system as follow:

$$
\begin{aligned}
R(t) &= 0.66342e^{-0.001t} + 0.27933e^{-0.002t} \\
&\quad + 0.051456e^{-0.003t} + \cdots \\
\mathrm{MTTF} &= 874.38 \\
A &= 874.38/(874.38 + 10) = 0.98869
\end{aligned}
$$

4. Running Time Analysis

Using the C language, we wrote a program for Step 2 that constructs a model based on the proposed method and outputs it as an input file to SHARPE. For Step 1 we implemented an algorithm to enumerate all MFSFs [11]. By applying these programs and SHARPE to various example systems, we conducted running time analysis in order to show the applicability of the proposed method.

We took a collection of networks from [2], as shown in Fig. 9, and used them as the topologies of benchmark systems. Distribution of resources (programs and files) were set as illustrated in this figure.

For each system in Fig. 9, we executed the programs on a SUN Ultra SS1 workstation and measured the running time needed for evaluation. Note that the total running time consists of (1) the time needed for MFSF enumeration (Step 1), (2) the time needed for model construction (Step 2), and (3) the time needed for analysis by SHARPE (Step 3).

As mentioned before, the previous method [10] based on Petri-nets cannot evaluate the dependability measures with admissible running time when the system is large. Thus for comparison, we counted the total number of states of the underlying Markov chain that

Table 1 Results of performance analysis.

	#of components (nodes, links)	#of MFSFs	total running time(s)	Step 1	Step 2	Step 3	#of states of previous model [10]
1	22 (9,13)	45	13.50	1.59	0.01	11.50	2415
2	23 (9,14)	25	13.76	0.11	0.01	12.64	6859
3	29 (11,18)	27	18.82	0.20	0.02	18.60	253049
4	31 (10,21)	35	347.98	0.17	0.01	347.80	2053569
5	32 (11,21)	69	2058.14	1.08	0.02	2057.04	2080116
6	35 (13,32)	59	1699.71	4.95	0.02	1694.74	$\gg$ 10000000
7	35 (14,21)	40	1599.98	0.30	0.01	1599.67	$\gg$ 10000000
8	40 (16,24)	50	1090.68	9.90	0.03	1080.75	$\gg$ 10000000
9	45 (18,27)	19	260.20	0.04	0.02	260.14	$\gg$ 10000000
10	46 (16,30)	12	260.09	0.03	0.02	260.04	$\gg$ 10000000
11	47 (21,26)	27	1157.96	0.09	0.02	1157.85	$\gg$ 10000000
12	50 (20,30)	21	2161.58	0.31	0.01	2161.26	$\gg$ 10000000

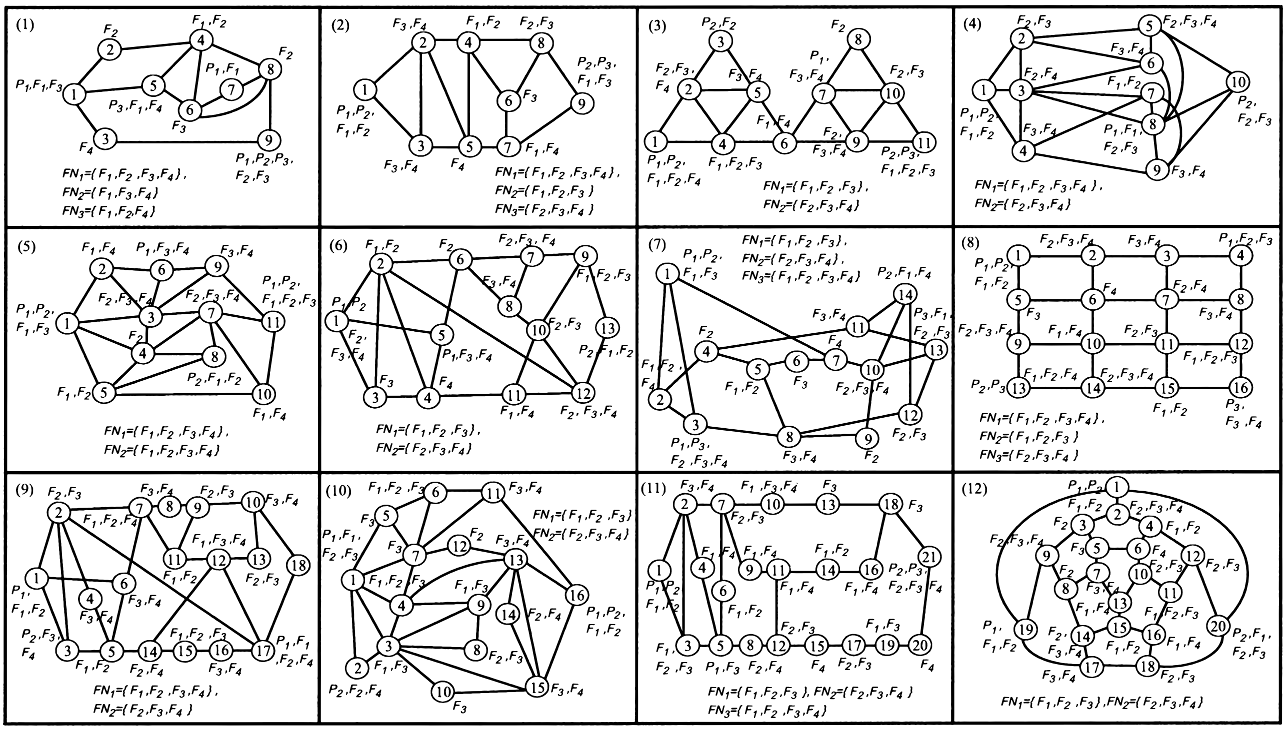


Fig. 9 Benchmarks.

the Petri-net-based method would generate.

Table 1 summarizes the total running times and the total numbers of states. From this table, it is seen that the running times of the proposed method were fairly acceptable in all cases. (They were less than one hour.) It is also observed that most of the running time was spent in analyzing a constructed model by SHARPE. On the contrary, one can see that the previous method in [10] would cause state explosion even when the number of nodes was less than 15. The result of this comparison study clearly shows that the applicability of the proposed method is much superior to that of the previous method.

5. Discussion on Failure Types

In this section, we remark on failure types that the proposed method can deal with. Various kinds of component failures may occur in a distributed system. In general, component failures are classified into the following four categories based on the failed component's behavior [5], [7]: crash failure, omission failure, timing failure, and Byzantine failure. Crash failure causes the component to halt. Omission failure and timing failure cause the component to not respond or to respond too late, respectively. These three types of failures can be detected by defining the response time of the component if the system is not completely asynchronous. Finally, Byzantine failure causes the component to behave in a totally arbitrary manner. This type of failures is exceedingly difficult to detect, and thus, their detection requires strong assumptions on underlying distributed systems. In general, therefore, a system may experience both detectable and undetectable component failures or detectable failures only, depending on the assumptions made on the system and the detection mechanism used.

In many dependability models of distributed systems that have been proposed so far, a component is assumed to be in one of the two states: operational or failed. This assumption is appropriate for detectable failures, but it cannot deal with undetectable failures. On the other hand, in our model, a component is in one of the three state: operational, covered, or noncovered. As mentioned before, state 'covered' represents the fact that the component failed and the failure was handled successfully, and state 'noncovered' signifies that the component failed and handling (including detection) of the failure was not completed. Therefore, our model can deal with both detectable failures and undetectable failures.

Another type of failures that can occur are those that degrade performance of a component (or a system). In order to deal with this type of failures, a new dependability measure and system model are necessary. Our future research includes their development.

6. Conclusions

We have proposed a hierarchical method for dependability evaluation of distributed systems where programs and data files are distributed in a redundant manner. The proposed method explicitly takes the failure-repair behavior of the system into account. Since the method employs Markov modeling in a localized manner, it can avoid explosive state-space growth. By running time analysis, we have shown that the proposed method outperforms the previous method [10] in terms of applicability.

Future research includes the following. (1) In the system model used in this work, it is assumed that any noncovered component failure leads to system failure. This assumption may be impractical if the component is isolated from MFSFs due to link and node failures. We are planning to extend the system model so as to consider connectivity of components. (2) In SHARPE, input events of a fault tree must be mutually independent, and this requirement limits features that can be analyzed. As mentioned before, we make use of the fact that when a component s_i is not operational, it is in the state 'Noncovered' with probability $1 - c_i$, in order to construct a model satisfying this requirement. However, since the property does not hold for components with local repairs, this technique is not applicable to them. We are planning to improve the proposed method to cope with this problem.

Acknowledgements

The authors would like to thank anonymous referees for their helpful suggestions.

References

[1] D.J. Chen and T.H. Huang, "Reliability analysis of distributed systems based on a fast reliability algorithm," IEEE Trans. Parallel & Distributed Systems, vol.3, no.2, pp.139–154, 1992.
[2] Y.G. Chen and M.C. Yuang, "A cut-based method for terminal-pair reliability," IEEE Trans. Reliability, vol.45, no.3, pp.413–416, 1996.
[3] H. de Meer, K.S. Trivedi, and M. Dal Cin, "Guarded repair of dependable systems," J. Theoretical Computer Science, vol.128, no.1-2, pp.179–210, 1994.
[4] J.B. Dugan and K.S. Trivedi, "Coverage modeling for dependability analysis of fault-tolerant systems," IEEE Trans. Comput., vol.38, no.6, pp.775–787, 1989.
[5] P.D. Ezhilchelvan and S.K. Shrivastava, "A characterization of faults in systems," Sixth Symposium on Reliability in Distributed Software and Database Systems, pp.215–222, 1986.
[6] S. Hariri and H. Mutlu, "Hierarchical modeling of availability in distributed systems," IEEE Trans. Software Eng., vol.21, no.1, pp.50–56, 1995.
[7] P. Jalote, Fault Tolerance in Distributed Systems, PTR Prentice Hall, 1994.
[8] A. Kumar and D.P. Agrawal, "A generalized algorithm for evaluating distributed-program reliability," IEEE Trans. Reliability, vol.42, no.3, pp.416–426, 1993.
[9] A. Kumar, S. Rai, and D.P. Agrawal, "On computer communication network reliability under program execution constraints," IEEE J. Select. Area Commun., vol.6, no.8, pp.1393–1399, 1988.
[10] N. Lopez-Benitez, "Dependability modeling and analysis of distributed programs," IEEE Trans. Software Eng., vol.20, no.5, pp.345–352, 1994.
[11] V.K. Prasanna Kumar, S. Hariri, and C.S. Raghavendra, "Distributed program reliability analysis," IEEE Trans. Software Eng., vol.12, no.1, pp.42–50, 1986.
[12] C.S. Raghavendra, V.K. Prasanna Kumar, and S. Hariri, "Reliability analysis in distributed systems," IEEE Trans. Comput., vol.37, no.3, pp.352–358, 1988.
[13] R.A. Sahner and K.S. Trivedi, "Reliability modeling using SHARPE," IEEE Trans. Reliability, vol.36, no.2, pp.186–193, 1987.
[14] R.A. Sahner, K.S. Trivedi, and A. Puliafito, Performance and Reliability Analysis of Computer Systems, Kluwer Academic Publishers, 1995.

Appendix

In [12], a breadth-first search-based algorithm, called *MFSF algorithm*, is proposed to enumerate MFSFs. However, this algorithm can produce incorrect results.

As an example, consider the system represented by the graph in Fig. A· 1. In this system, node x_1 holds programs P_1, P_2 and file F_1, while node x_2 holds program P_2 and file F_2. Suppose $FN_1 = \{F_1\}$ and $FN_2 = \{F_2\}$. It is then clear that subgraph $(\{x_1, x_2\}, \emptyset)$ is the

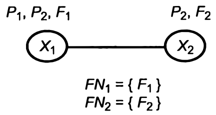


Fig. A· 1 Example for which the algorithm in [12] produces an incorrect result.

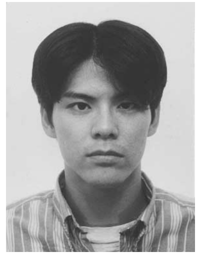
only MFSF.

For this system, the MFSF algorithm works as follows: In Step 1, the Cartesian product of PA_i's, i.e., $PA_1 \times PA_2$ is computed, where PA_i denotes the set of nodes that can host P_i. In this case, it is equal to $\{(x_1, x_1), (x_1, x_2)\}$, so TRY is set to $\{(\{x_1\}, \emptyset), (\{x_1, x_2\}, \emptyset)\}$. Next, $(\{x_1, x_2\}, \emptyset)$ is removed from TRY because $(\{x_1\}, \emptyset)$ is its subgraph. Then, $TRY = \{(\{x_1\}, \emptyset)\}$. $FOUND$ is initialized to be empty.

In Step 2, two substeps called checking step and expanding step are performed repeatedly. The checking step checks for each subgraph in TRY whether it is an MFSF or not. In this case, since x_1 does not hold file F_2, the algorithm finds that a subgraph $(\{x_1\}, \emptyset)$ is not an MFSF. Next, the expanding step is performed. This substep constructs larger forests by adding edges and nodes to the forests in TRY. In this case, since $x_{1,2}$ is incident on x_1, $(\{x_1, x_2\}, \{x_{1,2}\})$ is constructed and stored in TRY. Then, the checking step is performed again. Since $(\{x_1, x_2\}, \{x_{1,2}\})$ holds all the needed files, it is stored in $FOUND$ and removed from TRY. Then, TRY becomes empty, and the MFSF algorithm halts.

Consequently, $FOUND$ becomes $\{(\{x_1, x_2\}, \{x_{1,2}\})\}$. This means that MFSF algorithm concludes that $(\{x_1, x_2\}, \{x_{1,2}\})$ is the only MFSF. However, this result is incorrect. In general, such a case may occur when $PA_i \bigcap PA_j \neq \emptyset$ for $P_i, P_j (i \neq j)$.

Eun Hye Choi received ME degree in computer engineering from Osaka University in 1999. She is currently a doctoral student in the Department of Informatics and Mathematical Science at Osaka University. Her research interests include distributed and fault-tolerant computing systems.



Tatsuhiro Tsuchiya received ME and PhD degrees in computer engineering from Osaka University in 1995 and in 1998, respectively. He is currently an assistant professor in the Department of Informatics and Mathematical Science at Osaka University. His current research interests are in the areas of distributed computing and fault-tolerant computing.

Tohru Kikuno received MS and PhD degrees from Osaka University in 1972 and 1975, respectively. He was with Hiroshima University from 1975 to 1987. Since 1990, he has been a professor in the Department of Informatics and Mathematical Science at Osaka University. His research interests include the quantitative evaluation of software development processes, the analysis and design of fault-tolerant systems, and the design of procedures for testing communication protocols. He served as a program co-chair of the First International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'98) and the Fifth International Conference on Real-Time Computing Systems and Applications (RTCSA'98).