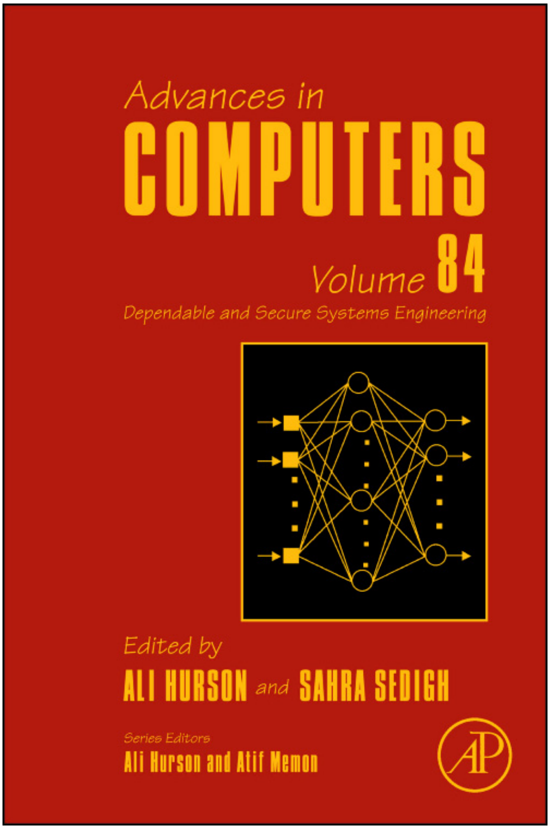

Advances in
COMPUTERS
Volume 84
Dependable and Secure Systems Engineering
Edited by
ALI HURSON and SAHRA SEDIGH
Series Editors
Ali Hurson and Atif Memon
AP

From: Kishor Trivedi, Ermeson Andrade, and Fumio Machida, Combining
Performance and Availability Analysis in Practice.
In Ali Hurson and Sahra Sedigh, editors: *Advances in Computers,* Vol. 84,
Burlington: Academic Press, 2012, pp. 1-38.
ISBN: 978-0-12-396525-7
© Copyright 2012 Elsevier Inc.
Academic Press

Combining Performance and Availability Analysis in Practice

KISHOR TRIVEDI

Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, USA

ERMESON ANDRADE

Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, USA
Informatics Center, Federal University of Pernambuco (UFPE), Recife, Pernambuco, Brazil

FUMIO MACHIDA

Department of Electrical and Computer Engineering, Duke University, Durham, North Carolina, USA
Service Platforms Research Laboratories, NEC Corporation, Kawasaki, Japan

Abstract

Composite performance and availability analysis of computer systems has gained considerable attention in recent years. Pure performance analysis of a system tends to be optimistic since it ignores the failure–repair behavior of the system. On the other hand, pure availability analysis tends to be too conservative since the behavior of the system is captured by only two states (functioning or failed). To analyze the degradation of a system's performance in consideration with availability metrics, combined measures of performance and availability are essential. This chapter introduces the basics of analytic models for the combined performance and availability analysis of computer systems together with some practical examples.

ADVANCES IN COMPUTERS, VOL. 84
ISSN: 0065-2458/DOI: 10.1016/B978-0-12-396525-7.00001-0

1. Introduction

The need for combining performance and availability analysis of computer systems is increasing, since most computer systems can continue their operations even in the presence of faults. However, software/hardware designers are still using performance and availability measures separately to evaluate the quality of the systems. Such separated analysis is not sufficient to properly understand and predict the behavior of these systems because the performance is affected by the failures and recoveries of the system components. Thus, the use of evaluation methods which combine performance and availability analysis is essential [1–7].

In recent decades, several approaches have been developed for considering the combined evaluation of performance, availability, and reliability [8–14]. Beaudry [15] is the first author to develop the measures which provide trade-offs between reliability and performance of degradable systems. Thereafter, the term performability, where the concept of performance and reliability is unified, was introduced by Meyer [16]. He developed a general modeling framework that covers performability measures.

Quantitative evaluation of systems' performance and reliability/availability can be broadly classified into measurement and model-based approaches. In the measurement approach, the collected data accurately show the phenomena observed in the system, but the evaluation tends to be expensive. Some experiments are not always feasible because they are either time-consuming or need expensive procedures (like fault injections). By contrast, in the model-based approach, the evaluation of systems can be carried out without the actual execution on the real system. The model provides an abstraction of the system which does not always predict the performance and availability accurately. However, if the models are properly validated, the model-based approach might present a better cost-effective approach over the measurements. Both the approaches can be used together depending on the criticality of the system and/or availability of resources. Often, measurements are made at the subsystem level, and these are rolled up to the system level by means of models [17,18]. In this chapter, we discuss the model-based approach.

Different modeling techniques can be used for combining performance and availability analysis. Among of them, the exact composite approach [3] has been widely used because of its accuracy. However, this approach faces largeness and stiffness problems. Largeness occurs because of a cross-product of states of performance model and availability model. To deal with the largeness problem, two basic techniques can be applied: largeness tolerance and largeness avoidance [4]. Stiffness arises when the rates related to performance models are much faster than the rates of availability models. Aggregation techniques [19] and stiffness-tolerance [20] are effective methods in dealing with the stiffness problem. Hierarchical modeling approach [12] is another potential largeness and stiffness avoidance technique. This approach divides the system model into several small submodels. The submodels can be of different types, such as non-state-space models and state-space models. The solution of the hierarchical model is computed by passing outputs of lower-level submodels as inputs to the higher level submodels. In case of cyclic dependence among submodels, fixed-point iterative can be applied [8,21].

This chapter aims to present an overview of main techniques used in model construction and solution of composite performance and availability analysis, such as exact composite approach and hierarchical modeling approaches. We also describe techniques used for pure availability analysis and pure performance analysis. Practical examples where such techniques were successfully applied are detailed.

The chapter is organized as follows: Section 2 introduces basics of analytic models for evaluating performance and availability of systems and also describes modeling techniques for combining performance and availability analysis. Section 3 describes a set of practical examples for combining availability and performance analysis. Section 4 concludes the chapter.

2. Approaches to Modeling

In pure performance modeling, probabilistic nature of user demands (workload) as well as internal state behavior needs to be represented under the assumption that the system/components do not fail [4]. Several stochastic models can be used for performance analysis, such as series–parallel directed acyclic graphs [4], product form queuing networks [22], Markov chains [23], semi-Markov process (SMP) [1], Markov regenerative process [24], generalized stochastic Petri nets (GSPNs) [25], stochastic reward nets (SRNs) [4], hierarchical [12] and fixed-point iterative [26], and the combination of these. Metrics such as throughput, blocking probability, mean response time, response time distribution, and utilization can be computed based on these models.

According to ITU-T Recommendation E.800 [27], "availability is the ability of an item to be in a state to perform a required function at a given instant of time or at any instant of time within a given time interval, assuming that the external resources, if required, are provided". On February 1991, the Patriot missile defense system failed to intercept an incoming missile. This incident resulted in the death of 28 US Army reservists [28]. Thus, high availability of mission-critical systems is extremely important, since failures can be catastrophic. For business critical systems and critical infrastructures, high availability is also important to minimize the cost of downtime.

Analytic models have been widely used to predict the system availability. These models can provide important insights about the availability considering different scenarios before the system is released for use. The availability aspects of the system are usually described by non-state-space models (reliability block diagram (RBD), fault tree (FT), and reliability graph), state-space models such as Markov chains, SMP, Markov regenerative process, stochastic Petri nets (SPNs) of various ilk, and hierarchical and fixed-point iterative models. Downtime, steady-state availability, instantaneous availability, and interval availability are frequently used as measures. Assuming exponential failure and repair time distributions with respective rates λ and μ, the availability at time t and the interval availability can be computed by the following expressions [23]:

$$A(t) = \frac{\mu}{\lambda+\mu} + \frac{\mu}{\lambda+\mu}\mathrm{e}^{-(\lambda+\mu)t}$$

$$A_I(t) = \frac{\int_0^t A(x)\mathrm{d}x}{t} = \frac{\mu}{\lambda+\mu} + \frac{\lambda}{(\lambda+\mu)^2 t}\left(1 - \mathrm{e}^{-(\lambda+\mu)t}\right)$$

Taking a limit to infinity of the instantaneous availability, the steady-state availability A_{ss} can be computed as below

$$A_{\mathrm{ss}} = \lim_{t\to\infty} A(t) = \frac{\mu}{\lambda+\mu}$$

The steady-state unavailability U_{ss} and downtime (in minutes per year) are obtained from A_{ss} by the following expressions

$$U_{\mathrm{ss}} = (1 - A_{\mathrm{ss}})$$

$$\text{Downtime} = (1 - A_{\mathrm{ss}}) \times 8760 \times 60$$

Composite performance and availability analysis is required especially in the evaluation of degradable systems. In degradable systems, when some system components fail, the system can undergo a graceful degradation of performance and still be able to continue operation at a reduced level of performance. In other words, the system can have more than two working states (i.e., functioning, partially functioning, and down). One of the most used analytic model types for combining performance and availability analysis is Markov reward model in which each state of Markov chain is assigned a reward rate according to the performance delivered in the state. In the following subsections, we introduce the basics of analytic modeling for performance and availability evaluation based on two types of models: non-state-space models and state-space models.

2.1 Non-State-Space Models

Availability models can be constructed using non-state-space models such as reliability block diagram (RBD), reliability graph (Relgraph), and FT with and without repeated events. Non-state-space models are easy to use and have a relatively quick solution because they can be solved without generating the underlying state space [29]. For a rapid solution, these models assume that system components are independent of each other. System availability, system unavailability, system reliability, and system mean time to failure can be computed using these models. The three commonly used solution techniques for non-state-space model are factoring [23], sum of disjoint products [23], and binary decision diagram [30]. Large non-state-space model can be solved by deriving upper and lower bounds as described in Ref. [31].

RBD is a non-state-space model type that enables analysis of reliability and availability of complex systems using block diagrams. In a block diagram model, components are combined into blocks in series, parallel, or *k-out-of-n*. A series structure represents a direct dependency between the components where the entire system fails if one of its components fails. A parallel structure is used to show redundancy and means that the whole system can work properly as long as at least one component is working properly. A *k-out-of-n* structure represents that the whole subsystem can work properly as long as *k* or more components are working properly out of *n* components. Series and parallel structures are special cases of *k-out-of-n* structures [4]. A series structure is an *n-out-of-n* and a parallel structure is a *1-out-of-n* structure. Figure 1 shows an RBD representing a storage system availability model with one server, one hub, and *n* storages devices. The system is working properly if at least one of each device (server, hub, and storage) is working properly.

FT can be used for quantitative analysis of system reliability/availability as well as qualitative analysis. FT depicts a combination of events and conditions that can lead to an undesired event such as system failure. Basic FT consists of events and logical event connectors such as OR gates, AND gates, and *k-out-of-n* gates. The events can be combined in several ways using logical gates according to the system configuration. FT can have repeated events in situations in which the same failure event propagates along different paths. Figure 2 presents an FT model for the storage system with one server (S), one hub (H), and *n* storage devices (SD). In contrast to the RBD model, the FT takes a "negative" view in that it describes the condition under which the system fails. Since FTs allow repeated events, they are more powerful than series–parallel RBDs. For a comparison of modeling power of these model types, see Ref. [32].

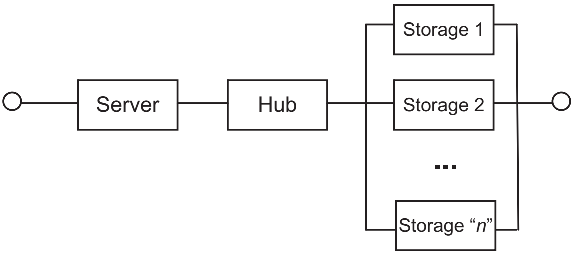


FIG. 1. RBD for a storage system.

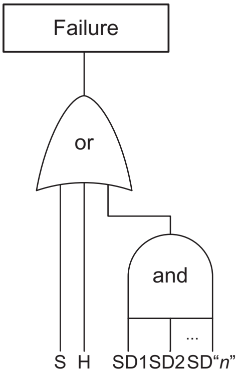


FIG. 2. FT for a storage system.

2.2 State-Space Models

Non-state-space models such as RBD and FT cannot easily handle detailed failure/repair behavior such as imperfect coverage, correlated failures, repair dependencies, etc. On the other hand, state-space models are capable of capturing such detailed behavior. As a well-known state-space model type, continuous-time Markov chains (CTMCs) are widely used in performance and availability studies. Homogenous CTMCs are represented by states and transitions between the states whose sojourn time follows exponential distribution. If we relax the assumption of exponential distribution, then it might become an SMP [1], a Markov regenerative process [24], or a non-homogeneous Markov chain [33]. Labels on the transitions for homogenous CTMC are time-independent rates. Figure 3 presents a simple example of CTMC model for a two-component parallel redundant system with the same repair rate μ. The failure rate of both components is μ. When both components have failed, the system is considered as failed. There is a single shared repair person.

Solving for the steady-state probabilities, we have:

$$\pi_2 = \frac{\mu}{2\lambda}\pi_1$$

$$\pi_1 = \frac{\mu}{\lambda}\pi_0$$

Since

$$\pi_0 + \pi_1 + \pi_2 = 1$$

Thus

$$\pi_0 + \frac{\mu}{\lambda}\pi_0 + \left(\frac{\mu}{\lambda}\right)\left(\frac{\mu}{2\lambda}\right)\pi_0 = 1$$

Then, the steady-state unavailability of the parallel redundant system with a shared repair is expressed as below:

$$\pi_0 = \frac{1}{1 + \frac{\mu}{\lambda} + \frac{\mu^2}{2\lambda^2}}$$

As an approach for combining performance and availability analysis, this chapter focuses on Markov reward models. MRMs are one of the most commonly used techniques for combining performance and availability analysis of degradable systems. Formally, an MRM consists of a CTMC $\{Z(t), t \geq 0\}$ with state space Ω. Let $P_i(t)$ be the unconditional probability of the CTMC being in state i at time t, then the row vector $\boldsymbol{P}(t)$ represents the *transient state probability vector*. Given

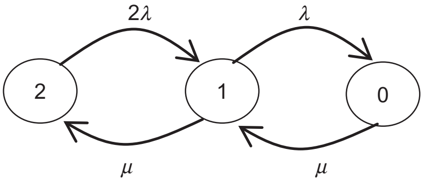
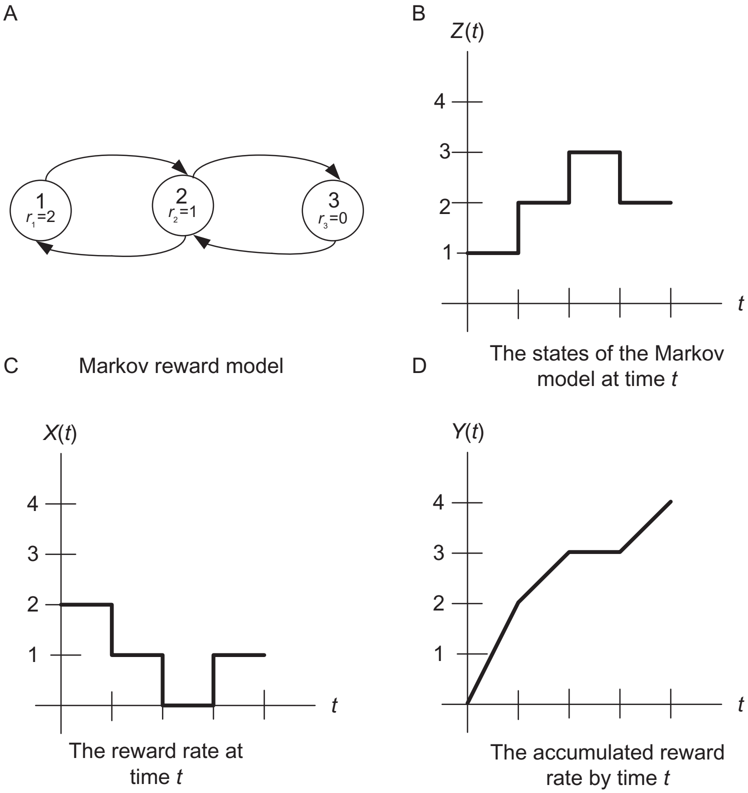


FIG. 3. An example of CTMC.



Markov reward model

The states of the Markov model at time *t*

The reward rate at time *t*

The accumulated reward rate by time *t*

FIG. 4. Sample of paths.

$L(t) = \int_0^t P(u)\mathrm{d}u$, $L_i(t)$ is the expected total time spent by the CTMC in state i during the interval $[0,t)$. The MRMs are obtained by assigning a reward rate r_i to each state i of the CTMC (See Fig. 4A). Let $X(t){=}r_{Z(t)}$ denote the reward rate at time t and $Y(t)$

represent the accumulated reward in the interval [0,t). Then, the Markov reward model and the possible sample paths of $X(t)$, $Y(t)$, and $Z(t)$ are detailed in Fig. 4.

The expected instantaneous reward rate $E[X(t)]$, the expected accumulated reward $E[Y(t)]$, and the steady-state expected reward rate $E[X(\infty)]$ can be computed as follows [23]:

$$E[X(t)] = \sum_{i\in\Omega} r_i P_i(t),$$

$$E[Y(t)] = \sum_{i\in\Omega} r_i \int_0^t P_i(\tau) = \sum_{i\in\Omega} r_i L_i(t),$$

and

$$E[X] = E[X(\infty)] = \sum_{i\in\Omega} r_i \pi_i.$$

where π_i is the limiting state probability, that is, $\pi_i = \lim_{t\to\infty} P_i(t)$.

The expected accumulated reward until absorption and the distribution $X(t)$ can be computed by the following expressions:

$$E[Y(\infty)] = \sum_{i\in\Omega} r_i \int_0^\infty P_i(\tau)\mathrm{d}\tau = \sum_{i\in B} r_i z_i,$$

and

$$P[X(t) \le x] = \sum_{r_i\le x,\ i\in\Omega} P_i(t).$$

One major drawback of MRM (or CTMC) is the largeness of their state space, since Markov chain for complex systems can easily reach hundreds, thousands, or millions of states. SPNs can be used for the specification and automatic generation of the underlying Markov chain in order to tolerate the state explosion problem through a more concise and smaller model as a starting point. CTMC underlying the SPN can then be generated, stored, and solved using efficient and numerically stable algorithms [22,23,34]. SPNs are composed of places, transitions (timed and immediate), arcs, and tokens. Figure 5 presents a simple SPN that shows the failure/recovery behavior of a system. The server starts in Up state, indicated by a token in place $\mathrm{P_{up}}$. The transition $\mathrm{T_{fail}}$ fires when the server goes down, and then the token in $\mathrm{P_{up}}$ is removed and a token is deposited in $\mathrm{P_{down}}$. $\mathrm{T_{recv}}$ fires when the server has recovered, then the token in $\mathrm{P_{down}}$ is removed and a token is deposited in $\mathrm{P_{up}}$.

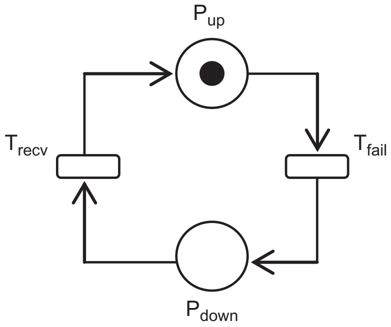


FIG. 5. SPN example.

Many extensions of SPNs have been proposed for allowing more concise and powerful description of Petri net models, such as GSPN. GSPN is a generalization of SPN by allowing both immediate transitions and timed transitions. To facilitate the automatic generation of MRMs, reward rates are also defined in term of GSPN. Thus, GSPNs are extended by introducing reward and guard functions to obtain the SRNs. For a comparison of GSPN and SRN, the reader should refer to Refs. [35,36]. Some of the most prominent functionalities of SRNs are

- *Priorities*. This extension is used when more than one transition is enabled at the same time, where the transition with higher priority is the only one allowed to fire. Although inhibitor arcs can be used to achieve the same purpose, priorities make the model simpler.
- *Guards*. This feature extends the concept of priorities and inhibitor arcs, providing a powerful means to simplify the graphical representation and to make SRNs easier to understand, since it allows the designers to use the entire state-space of the model by adding an enabling expression to a transition.
- *Marking-dependent arc multiplicity*. This feature provides a way to model situations where the number of tokens removed (or deposited) from (to) a place can depend upon the system state.
- *Marking-dependent firing rates*. This functionality allows the firing rate of a transition to depend on the current marking of the model.
- *Rewards*. This feature allows the assignment of reward rates to the marking of the SRN model. It can be used to obtain not only system performance/availability measures, but also combined measures of performance and availability.

Petri net-based models have been successfully applied to several types of systems [37–42] and allow the modeling of parallel, concurrent, asynchronous, and non-deterministic behaviors. Many tools for modeling and analysis of Petri nets are available like TimeNet [43], GreatSPN [44], SHARPE [45], and SPNP [46]. For review of other Petri net extensions, see Ref. [47].

Although CTMCs provide a useful approach to construct models of systems that include structural variations and performance levels, the models can be very large and complex, and the model construction becomes error-prone. A two-level hierarchical model utilizing MRMs can alleviate these problems in the combined performance and availability analysis. In this hierarchical approach, a sequence of lower performance models is solved, one for each state of the availability model. The obtained performance measures become reward rates to be assigned to states of the upper level MRM availability model. Upper level MRM is then solved to compute the overall performance measure. Although approximation is involved in this process, the errors caused by the approximation are generally acceptable in practice. CTMC/MRM for real complex systems tend to be composed of huge number of states. Hence, formalisms for a concise description of the models as well as the automatic conversion into a CTMC/MRM are necessary. GSPN and SRN have been widely used for this purpose, since they are isomorphic to CTMCs and MRMs, respectively. In the Petri net-based models, SRN models are more powerful and more manageable than GSPN models [35,36].

3. Practical Examples

We next review several studies on pure availability and pure performance analysis as well as composite performance and availability analysis. For some examples, we detail the SHARPE input files. SHARPE (Symbolic Hierarchical Automated Reliability/Performance Evaluator) [45] is a software package which allows the specification and analysis of stochastic models. It supports the following model types: Markov chains (acyclic, irreducible, and phase type), semi-Markov chains (acyclic and irreducible), Markov regenerative processes, RBDs, FTs, reliability graphs, single-chain product form queuing networks, multiple-chain product form queuing networks, GSPN, SRN, and series–parallel acyclic graphs.

3.1 Pure Reliability/Availability and Pure Performance Analysis

In this subsection, we describe techniques used in the construction and solution of pure availability models and pure performance models.

3.1.1 Two-Board System

Many techniques have been proposed to capture the multistate system availability. In Ref. [48], we used three analytic model types (CTMC, SRN, and FT) and compared the results among them. To show the comparative study, we adopted an example of two-board system as shown in Fig. 6. The system consists of two boards (B_1 and B_2), each of which has a processor (P_1 or P_2) and a memory (M_1 or M_2). The state of each board is (1) both P and M are down, (2) P is working properly but M is down, (3) M is working properly but P is down, or (4) both P and M are functional. We assumed that the time to failure of the processor and the memory is exponentially distributed with rates λ_p and λ_m, respectively. Common cause failure in which both the processor and the memory on the same board fail is also taken into account by assuming exponential distribution with rate λ_{mp}.

Figure 8 presents the CTMC reliability model of the two-board system. The states of the CTMC are represented by a binary vector showing the states of P_1, M_1, P_2, and M_2. Note that 1 represents up state of the device and 0 represents its down state. Figures 7 and 9 depict the SHARPE input file for the CTMC. First of all, any character after "*"(asterisk) is considered to be a comment and is ignored for SHARPE execution. On line 1, *format 8* means the number of digits after the decimal point to be printed in results. On lines 4 through 8, the variables used as parameters are given values. The failure rates of the processor and the memory (λ_p and λ_m) are set to 1/1000 and 1/2000 failures per hour, respectively. The mean time to common cause failure, $1/\lambda_{mp}$, is 1/3000h. When a group of parameters is given values, then the block must start with a keyword *bind* and finishes with the keyword *end*.

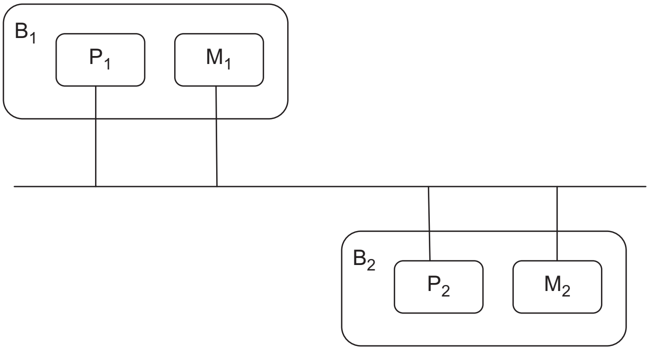


FIG. 6. Two-board system example.

```
 1  format 8
 2
 3
 4  bind
 5  lambda_mp   1/3000
 6  lambda_p   1/1000
 7  lambda_m   1/2000
 8  end
 9
10  markov PM
11  1111 1110  lambda_m
12  1111 1101    lambda_p
13  1111 1011    lambda_m
14  1111 0111    lambda_p
15  1111 0011 lambda_mp
16  1111 1100 lambda_mp
17  1110 1100  lambda_p
18  1110 1010    lambda_m
19  1110 0110    lambda_p
20  1110 0010 lambda_mp
21  1101 1100    lambda_m
22  1101 1001    lambda_m
23  1101 0101    lambda_p
24  1101 0001 lambda_mp
25  1011 1010    lambda_m
26  1011 1001    lambda_p
27  1011 0011    lambda_p
28  1011 1000 lambda_mp
29  0111 0110    lambda_m
30  0111 0101    lambda_p
31  0111 0011    lambda_m
32  0111 0100 lambda_mp
33  1100 1000    lambda_m
34  1100 0100    lambda_p
35  1100 0000 lambda_mp
36  1010 1000     lambda_p
37  1010 0010    lambda_p
38  0110 0100    lambda_p
39  0110 0010    lambda_m
40  1001 1000    lambda_m
41  1001 0001     lambda_p
42  0101 0100  lambda_m
43  0101 0001    lambda_m
44  0011 0010    lambda_m
45  0011 0001    lambda_p
46  0011 0000  lambda_mp
47  1000 0000    lambda_p
48  0100 0000    lambda_m
49  0010 0000    lambda_p
50  0001 0000    lambda_m
```

FIG. 7. SHARPE input for the CTMC example.

The model specification begins with a model type and a name (see line 10). In this case, the model type is *markov* which denotes Markov chain (CTMC) and the name is *PM*. SHARPE allows three kinds of Markov chains: irreducible, acyclic, and PH type. Lines 10 through 50 define the states and state transitions of the Markov chain. From lines 50 through 68, we define the reward configuration, where for each state of the CTMC a reward rate is assigned to it. Note that the keyword *reward* (see line 51 in Fig. 9) denotes that in the next group of lines, SHARPE will assign reward

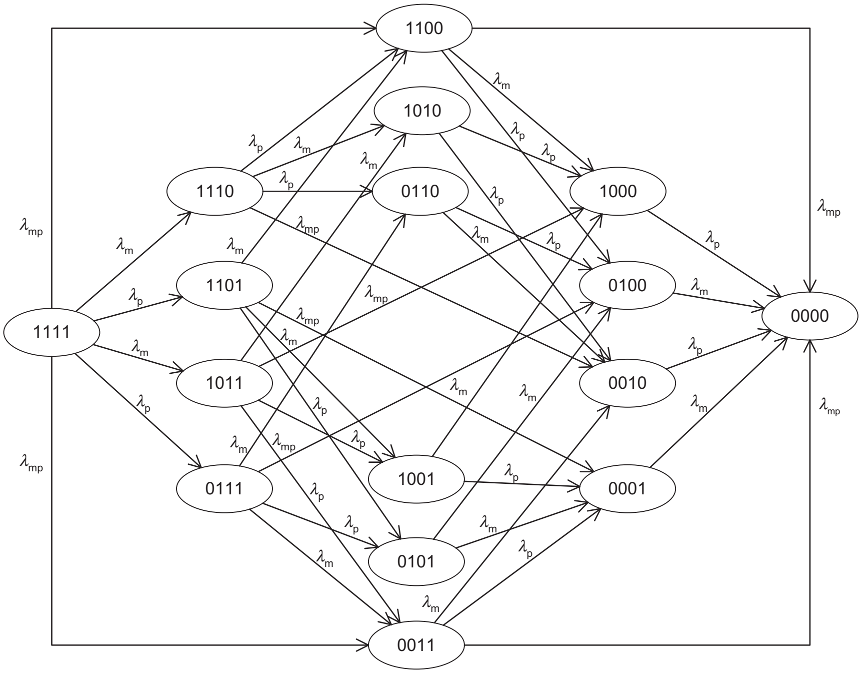


FIG. 8. CTMC for a two-board system.

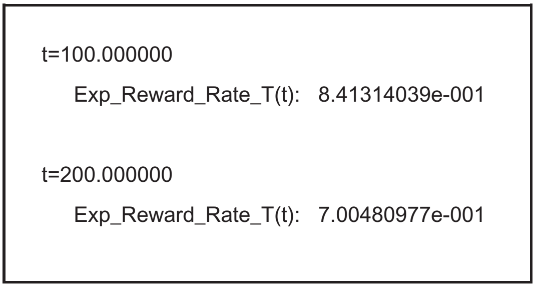
rates to the model states. For the two-board system, we adopted the CTMC model to compute the expected reward rate at time *t* for the case that at least one processor and both of the memories are operational. For that, we assigned the reward rate 1 to the UP states ((1,1,1,1) (1,1,0,1), and (0,1,1,1)) and 0 to the other states (down states). From lines 70 through 86, the initial state probabilities are specified. Initial state probabilities denote the likelihood of a sequence starting in a certain state. On lines 88 through 91, we define a function (*func*) to compute the expected reward rate at *t*=100 and *t*=200. It is important to highlight that the keyword *exrt* is a built-in function which gives the expected reward rate at time *t*. It takes as arguments a variable *t* and a Markov model name. The keyword *expr* says to evaluate an expression. Finally, line 93 contains the keyword *end* which means the end of the input file. The outputs for this example are shown in Fig. 10.

The SRN model for the same two-board system is shown in Fig. 11. Figure 11A describes the failure behavior of the processor P_1 and the memory M_1, while Fig. 11B depicts the failure behavior of the processor P_2 and the memory M_2. Tokens in the places M1U and M2U represent that the memories are operational.

```
* Reward configuration:
Reward
1111 1
1110 0
1101 1
1011 0
0111 1
1100 0
1010 0
0110 0
1001 0
0101 0
0011 0
1000 0
0100 0
0010 0
0001 0
0000 0
end
* Initial Probabilities:
1111 1
1110 0
1101 0
1011 0
0111 0
1100 0
1010 0
0110 0
1001 0
0101 0
0011 0
1000 0
0100 0
0010 0
0001 0
0000 0
end
*Output
func Exp_Reward_Rate_T(t) exrt(t; PM)
loop t,100,200,100
expr Exp_Reward_Rate_T(t)
end

end
```

FIG. 9. SHARPE input for the CTMC example (continuation).

Otherwise they are down. Likewise, tokens in the places P1U and P2U represent that the processors are operational. Otherwise, they are down. The SHARPE input file for the SRN model is shown in Fig. 13. From lines 4 through 8, the variables are given values. Note that the parameter values are the same as the ones used for the CTMC model. On lines 10 through 16, we define a reward function to compute the probability that at least one processor and both of the memories are operational. That is, the places P1U or P2U must have at least one token and the places M1U and M2U must have exactly two tokens.



```
t=100.000000
    Exp_Reward_Rate_T(t):   8.41314039e-001

t=200.000000
    Exp_Reward_Rate_T(t):   7.00480977e-001
```

FIG. 10. SHARPE output for CTMC example.

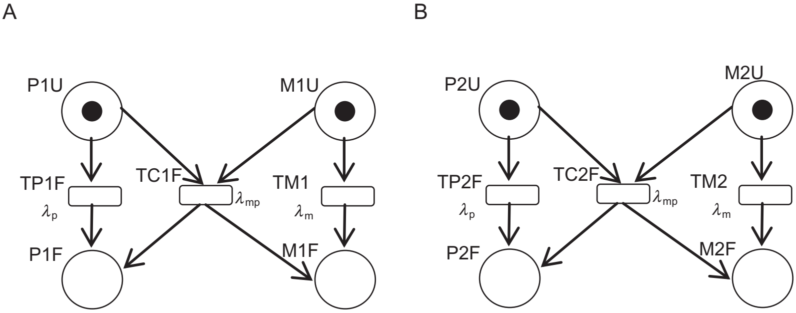
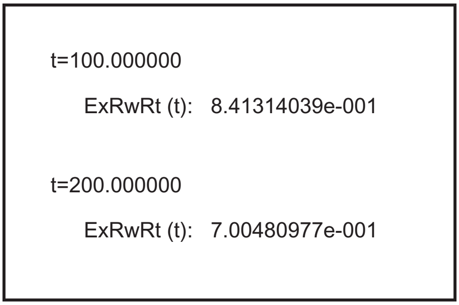


FIG. 11. SRN for a two-board system.



```
t=100.000000
    ExRwRt (t):   8.41314039e-001

t=200.000000
    ExRwRt (t):   7.00480977e-001
```

FIG. 12. SHARPE output for SRN example.

```
format 8


bind
lambda_mp   1/3000
lambda_p   1/1000
lambda_m   1/2000
end

func aval()
if((#(P1U)+#(P2U)>=1)and(#(M1U)+#(M2U)==2))
1
else
0
end
end

srn BS
  * == PLACE ==
  P6 0
  P1U 1
  M1U 1
  P1F 0
  M1F 0
  P2F 0
  P2U 1
  M2U 1
  end
  * == Timed Transitions ==
  TP1F ind lambda_p
  TC1F ind lambda_mp
  TM1F ind lambda_m
  TM2F ind lambda_m
  TC2F ind lambda_mp
  TP2F ind lambda_p
  end
  * == Immediate Transitions ==
  end
  * == ARC ==
  * Input Arcs
  P1U TP1F 1
  P1U TC1F 1
  M1U TC1F 1
  M1U TM1F 1
  P2U TP2F 1
  M2U TM2F 1
  M2U TC2F 1
  P2U TC2F 1
  end
  * Output Arcs
  TP1F P1F 1
  TM1F M1F 1
  TC1F M1F 1
  TC1F P1F 1
  TP2F P2F 1
  TM2F P6 1
  TC2F P2F 1
  TC2F P6 1
  end
  * Inhibtor Arcs
end

func ExRwRt(t) srn_exrt(t,BS; aval)
loop t,100,200,100
expr ExRwRt(t)
end

end
```

FIG. 13. SHARPE input for the SRN example.

On line 18, we begin the specification of model with a keyword *srn* which means SRNs and a name *BS*. The SRN specification is divided into the following basic blocks: places, timed transitions, immediate transitions, inputs arcs, output arcs, and inhibitor arcs. Lines 20 through 28 specify the places. Each line contains the name of a place and the number of token in the place. Lines 30 through 36 comprise the timed transitions. Each line contains the name of a timed transition followed by the keyword *ind* and the value/variable assigned to it. Lines 41 through 49 define the input arcs. Each line consists of a place name followed by a transition name and the multiplicity of the arc. Lines 51 through 59 specify the output arcs. Each line consists of a transition name followed by a place name and the multiplicity of the arc. Note that the SRN models do not have immediate transitions and inhibitor arcs. On lines 63 through 66, we define a function to compute the reward rate at time t=100 and t=200. The keyword *srn_exrt* is a built-in function which computes the expected reward rate at time t. It takes as arguments a variable t, an SRN model name, and a reward function, as multiple reward functions can be defined for the SRN. The outputs for the specified model are presented in Fig. 12. One should note that the results from the SRN models are identical to those from the CTMC model.

Finally, the multistate FT model considering that at least one processor and both of the memories are operational is depicted in Fig. 14. The SHARPE input file for the FT model is shown in Fig. 15. On line 4, we begin the specification of the FT

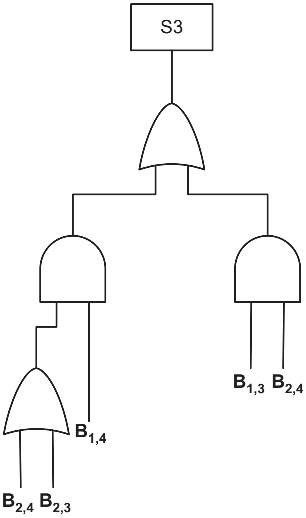
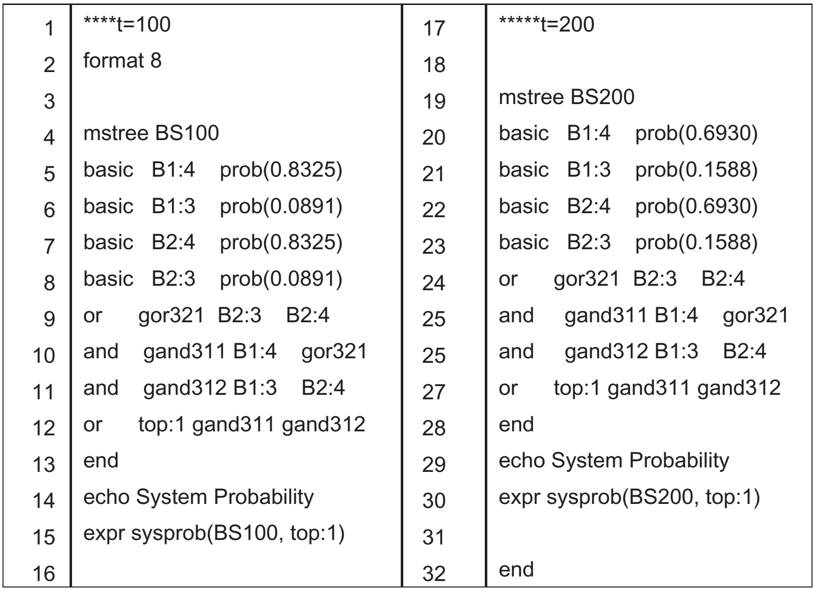


FIG. 14. FT for a two-board system.



```
1   ****t=100
2   format 8
3
4   mstree BS100
5   basic  B1:4  prob(0.8325)
6   basic  B1:3  prob(0.0891)
7   basic  B2:4  prob(0.8325)
8   basic  B2:3  prob(0.0891)
9   or     gor321  B2:3   B2:4
10  and    gand311 B1:4   gor321
11  and    gand312 B1:3   B2:4
12  or     top:1 gand311 gand312
13  end
14  echo System Probability
15  expr sysprob(BS100, top:1)
16
17  *****t=200
18
19  mstree BS200
20  basic  B1:4  prob(0.6930)
21  basic  B1:3  prob(0.1588)
22  basic  B2:4  prob(0.6930)
23  basic  B2:3  prob(0.1588)
24  or     gor321  B2:3   B2:4
25  and    gand311 B1:4   gor321
25  and    gand312 B1:3   B2:4
27  or     top:1 gand311 gand312
28  end
29  echo System Probability
30  expr sysprob(BS200, top:1)
31
32  end
```

FIG. 15. SHARPE input for the multistate FT example.

with a keyword *mstree* which means multistate FT and a name *BS100*. On lines 5 through 8, we define the events. An event begins with the keyword *basic*. For example, line 5 defines the event *B1:4* and assigns to it a transient probability of being in the component state $\pi_{\mathrm{B}_{1,4}}(t)$, where $\mathrm{B}_{1,4}$ denotes the board B_1 is in state 4. Each board is considered as a component with four states as stated above. The probability is obtained by solving the Markov chain in Fig. 16. The states of the CTMC are represented by a binary vector showing the states of P (processor) and M (memory), where 1 denotes up and 0 denotes down for each device. One should note that the probability is computed at t=100 but can be assigned with a variable value of t as a parameter. From lines 9 to 12, the structure of the multistate FT is defined. For instance, on line 9, the gate *or* is defined followed by the name *gor321* and its inputs *B2:3* and *B2:4*. On line 15, the system (failure) probability is computed. Note that for the second part of the specification (from lines 17 through 30), we consider t=200h. The results are depicted in Fig. 17. The solutions of the three models (CTMC, SRN, and FT) yield same results within numerical accuracy [48]. Note that the 4-state CTMC of a single board can also be in the input file and its state probabilities at time t can be directly passed onto the multistate FT, making a much better use of the capability of SHARPE.

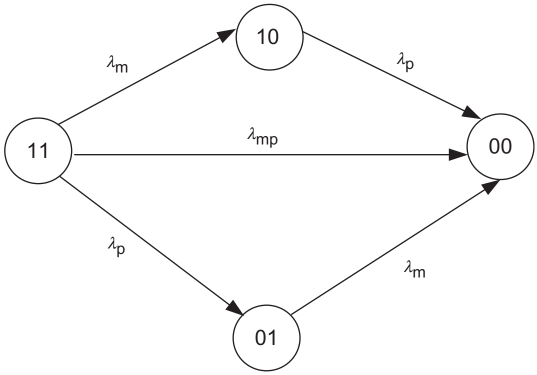
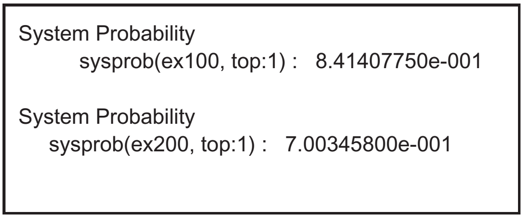


FIG. 16. CTMC model for a single board.



```
System Probability
        sysprob(ex100, top:1) :   8.41407750e-001

System Probability
    sysprob(ex200, top:1) :   7.00345800e-001
```

FIG. 17. SHARPE output for the multistate FT example.

3.1.2 VAXcluster System

Availability evaluation of a VAXcluster system using RBD is shown in Ref. [29]. This system consists of two or more VAX processors, a star coupler, one or more storage controllers (HSCs), and a set of disks (see Fig. 18). The RBD of a VAXcluster configuration is presented in Fig. 19 with N_p processors, N_h HSCs, and N_d disks. The system is working properly if at least one component of each set of devices is working properly. Since RBDs are easy to construct and solve, it is necessary to assume failure independence between system components and independent repair for components in order to solve the RBD (without generating the underlying state space). A detailed model using SRN can be found in Ref. [49], and a hierarchical model can be found in Ref. [41].

Figure 20 depicts the SHARPE input file for the VAXcluster system. On line 4, we begin the specification of the model with a keyword *block* which means RBD

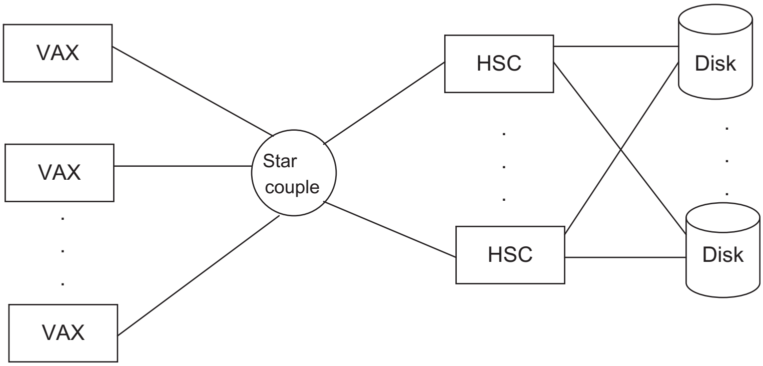


FIG. 18. VAXcluster system.

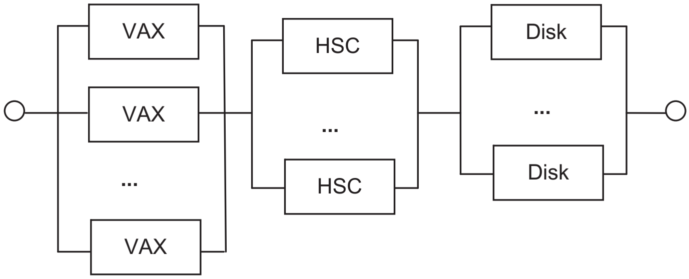


FIG. 19. Reliability block diagram for VAXcluster system.

and a name *Vax*. From lines 4 through 7, we define the blocks. Each line contains the keyword *comp*, the name of the component type, and a built-in function *exp*. For example, on line 5, the failure rate of each component of type *B* is 1/2000h. We could have used other built-in distributions such as Weibull. Lines 8 through 11 define the RBD structure. Each line consists of one of the keywords *series* or *parallel*, the name of the structure, and either the components or other structure being combined. Line 8 combines three components of type *A* in parallel. The name of this structure is *parallel2*. From lines 15 through 18, we compute the reliability of VAXcluster system for various values of *t*. Note that a loop is used to print the result, where *t* ranges from 100 to 1000 by using an increment of 100. Figure 21 shows the results. The system reliability decreases as *t* increases. We could also compute steady-state or instantaneous availability.

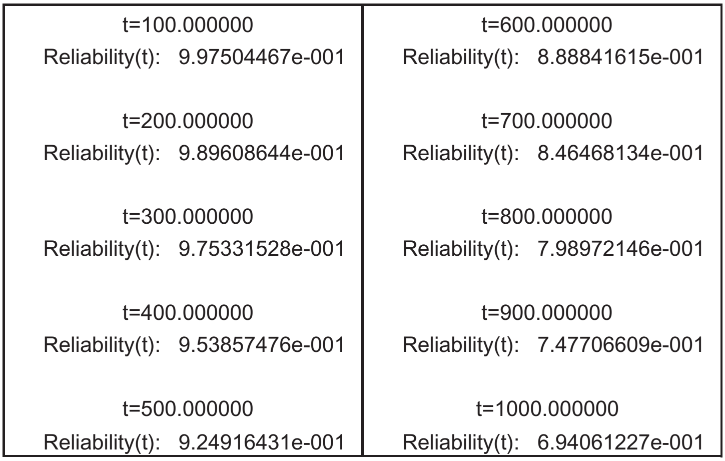
```
format 8
factor on

block Vax
comp B exp(1/2000)
comp A exp(1/3000)
comp C exp(1/1000)
parallel  parallel2  A A A
parallel  parallel3  B B
parallel  parallel5  C C C C
series  serie0  parallel2  parallel3  parallel5
End

*output
func Reliability(t) 1-tvalue(t;Vax)
loop t,100,1000,100
expr Reliability(t)
End

End
```

FIG. 20. SHARPE input for the VAXcluster system.

3.1.3 Telecommunication Switching System

Figure 22 presents the performance model of a telecommunication switching system by CTMC [50]. This system is composed of n trunks with an infinite caller population. We assumed call holding times are exponentially distributed with rate μ_h, and the call arrival process is Poison with rate λ_a. This model can be represented as $M/M/n/n$ queuing system. The blocking probability of new calls due to the lack of available trunks is computed by solving the CTMC. The blocking probability is given by π_n which is the probability that the CTMC is in state n in the steady state. As mentioned before, pure performance model does not consider the performance degradation caused by failures of system components.

Figure 23 shows the SHARPE input file for the telecommunication switching system. From lines 4 through 7, we assign values to the input parameters. The parameters are set to $\lambda_a = 5\,s^{-1}$ and $\mu_h = 0.3\,s^{-1}$. Lines 8 through 17 specify the



```
           t=100.000000                          t=600.000000
Reliability(t):   9.97504467e-001    Reliability(t):   8.88841615e-001

           t=200.000000                          t=700.000000
Reliability(t):   9.89608644e-001    Reliability(t):   8.46468134e-001

           t=300.000000                          t=800.000000
Reliability(t):   9.75331528e-001    Reliability(t):   7.98972146e-001

           t=400.000000                          t=900.000000
Reliability(t):   9.53857476e-001    Reliability(t):   7.47706609e-001

           t=500.000000                          t=1000.000000
Reliability(t):   9.24916431e-001    Reliability(t):   6.94061227e-001
```

FIG. 21. SHARPE output for RBD example.

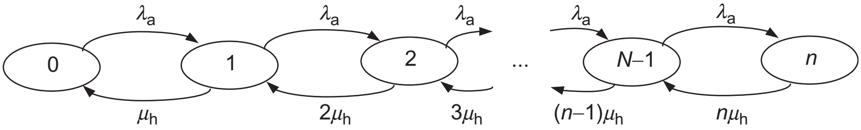


FIG. 22. Performance model of telecommunication switching system.

Markov reward model. Note that the Markov model is specified using a loop. This functionality is extremely powerful, since it allows creating complex models easily. As the blocking probability is given by π_n, the state n is assigned a reward rate 1 (see line 15). By default, the other states are assigned reward rate 0. On lines 19 through 21, the blocking probability is computed with the number of trunks varying from 1 to 10. The outputs are shown in Fig. 24. The outputs can be easily plotted by SHARPE as well.

3.2 Composite Performance and Availability Analysis

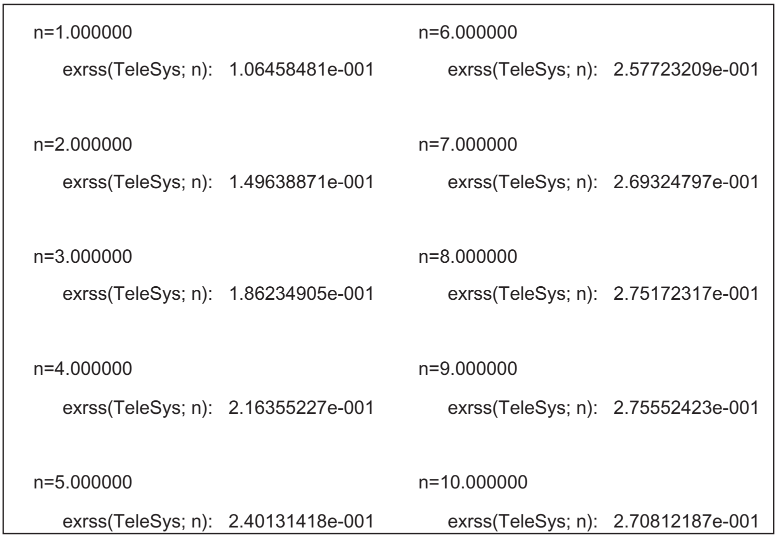
In the next subsections, we describe some examples of composite performance and availability analysis.

```
format 8

bind
mu_h   0.03
lambda_a   0.5
End

markov TeleSys(n)
loop i,0,n
    $(i) $(i+1) lambda_a
    $(i+1) $(i) (i+1)*mu_h
End
* Reward configuration:
Reward
$(n) 1
End
End

loop n,1,10,1
expr exrss(TeleSys; n)
End

```

FIG. 23. SHARPE input for the performance model.

3.2.1 Multiprocessor Systems

In Ref. [3], we discussed the use of MRM for combining performance and availability analysis of fault tolerant systems. Figure 25 presents the availability model of a multiprocessor system. This system is composed of n processors with covered and not covered failures. In covered failure case (whose probability is represented by c), the system must be reconfigured after the failure with a small delay (mean $1/\delta$). On the other hand, not covered failure (the probability of this case is $1-c$) denotes that the system must be rebooted with rate β after the failure. We assumed the system is down during the reconfiguration states (x_n, x_{n-1},. . .,x_2) and reboot states (y_n, y_{n-1},. . .,y_2). In this example, we were interested in computing unavailability and the normalized throughput loss (NTL), which are both instances of steady-state expected reward rate $E[X]$.



```
n=1.000000                                  n=6.000000
    exrss(TeleSys; n):   1.06458481e-001        exrss(TeleSys; n):   2.57723209e-001

n=2.000000                                  n=7.000000
    exrss(TeleSys; n):   1.49638871e-001        exrss(TeleSys; n):   2.69324797e-001

n=3.000000                                  n=8.000000
    exrss(TeleSys; n):   1.86234905e-001        exrss(TeleSys; n):   2.75172317e-001

n=4.000000                                  n=9.000000
    exrss(TeleSys; n):   2.16355227e-001        exrss(TeleSys; n):   2.75552423e-001

n=5.000000                                  n=10.000000
    exrss(TeleSys; n):   2.40131418e-001        exrss(TeleSys; n):   2.70812187e-001
```

FIG. 24. SHARPE output for the performance model.

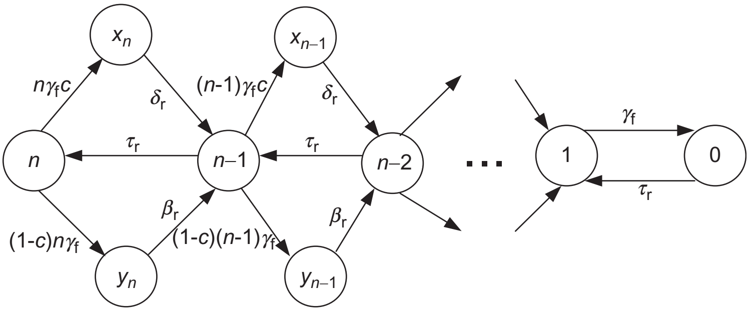


FIG. 25. Markov chain of a multiprocessor system.

Solving for the steady-state probabilities, we have [23]:

$$\pi_{n-i} = \frac{n!}{(n-i)!}(\gamma/\tau)^i \pi_n,$$

$$\pi_{x_{n-i}} = \frac{n!}{(n-i)!}\frac{\gamma(n-i)^c}{\delta}(\gamma/\tau)^i \pi_n,$$

$$\pi_{y_{n-i}} = \frac{n!}{(n-i)!}\frac{\gamma(n-i)(1-c)}{\beta}(\gamma/\tau)^i \pi_n,$$

$$i = 0, 1, 2, \ldots, n-2,$$

where

$$\pi_n = \left[\begin{array}{l} \displaystyle\sum_{i=0}^{n}(\gamma/\tau)^i \frac{n!}{(n-i)!} \\ \displaystyle+\sum_{i=0}^{n-2}(\gamma/\tau)^i \frac{\gamma(n-i)cn!}{\delta(n-i)!} \\ \displaystyle+\sum_{i=0}^{n-2}(\gamma/\tau)^i \frac{\gamma(n-i)(1-c)n!}{\beta(n-i)!} \end{array}\right]^{-1}.$$

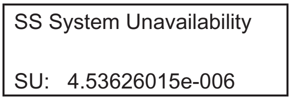
The unavailability of the system was computed by assigning a reward rate 1 to all down states (x_n, x_{n-1},. . .,y_n, y_{n-1}, and 0) and 0 to all the up states (n, n−1,. . .,1). Thus,

$$U_s = \sum_{i \in S_{rb}} \pi_i + \sum_{i \in S_{rp}} \pi_i + \sum_{i \in S_e} \pi_i$$

whereS_{rb}=$\{y_{n-1}|i$=0,1,. . .n−2}, S_{rb}=$\{x_{n-1}|i$=0,1,. . .n−2}, and S_e={0}.

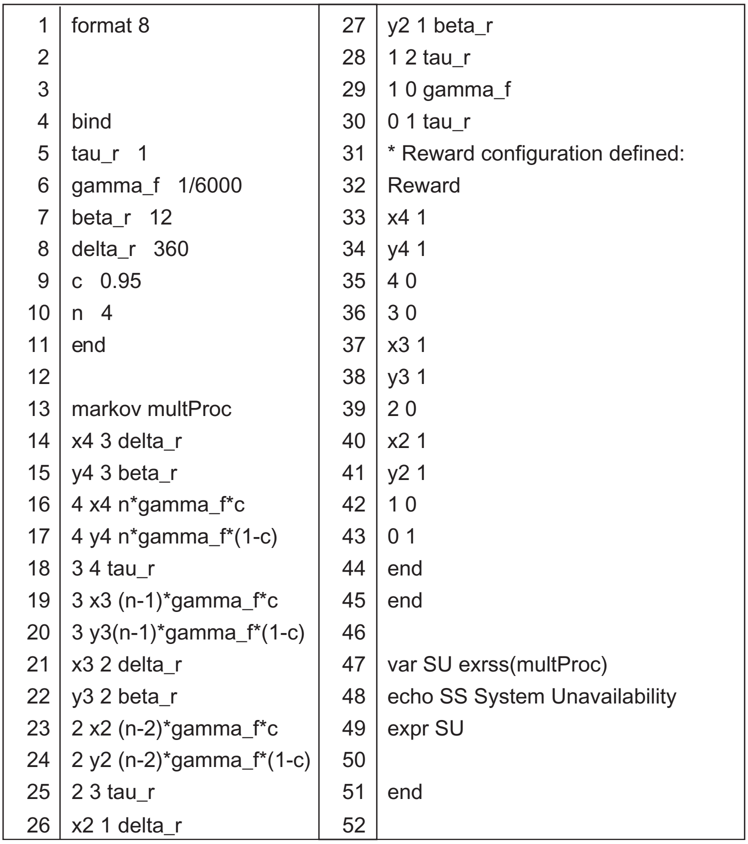
Figure 27 depicts SHARPE file to compute steady-state unavailability for the multiprocessor system. Lines 4 through 11 assign the values to the input parameters. The number of processors, n, is 4. The failure rate γ_f of a processor is 1/6000 failures per hours. The mean times to reboot, $1/\beta_r$, and reconfiguration, $1/\delta_r$, are set to be 5 min and 10 s, respectively. The mean time to repair a processor is 1 h. From lines 13 through 44, we define the Markov reward model. Note that the up states are assigned reward rate 0 and the down states are assigned reward rate 1 (see lines 32 thorough 44). From lines 47 through 49, we compute the steady-state system unavailability. Figure 26 presents the output for the Markov reward model.

To obtain the NTL representing the fraction of the jobs rejected, the up states were assigned with reward rate of a task being rejected caused by the fullness of the



```
SS System Unavailability

SU:   4.53626015e-006
```

FIG. 26. SHARPE output for multiprocessor system.



```
format 8


bind
tau_r   1
gamma_f   1/6000
beta_r   12
delta_r   360
c   0.95
n   4
end

markov multProc
x4 3 delta_r
y4 3 beta_r
4 x4 n*gamma_f*c
4 y4 n*gamma_f*(1-c)
3 4 tau_r
3 x3 (n-1)*gamma_f*c
3 y3(n-1)*gamma_f*(1-c)
x3 2 delta_r
y3 2 beta_r
2 x2 (n-2)*gamma_f*c
2 y2 (n-2)*gamma_f*(1-c)
2 3 tau_r
x2 1 delta_r
y2 1 beta_r
1 2 tau_r
1 0 gamma_f
0 1 tau_r
* Reward configuration defined:
Reward
x4 1
y4 1
4 0
3 0
x3 1
y3 1
2 0
x2 1
y2 1
1 0
0 1
end
end

var SU exrss(multProc)
echo SS System Unavailability
expr SU

end

```

FIG. 27. SHARPE input to compute the unavailability of the multiprocessor system.

buffers. Since arriving jobs are always rejected when the system is unavailable, the down states are assigned with reward rate 1. The computation of job loss probability due to full buffers can be carried out using a queuing model [23] or can be based on measurements. For this example, we assumed an $M/M/i/b$ queuing model (see Fig. 28).

The Markov chain for an $M/M/i/b$ queuing model is shown in Fig. 29. The state indices denote the number of jobs in the system. We assume that jobs arriving to the system form a Poisson process with rate λ and that the service requirements of jobs are independent, identically distributed according to an exponential distribution with mean $1/\mu$. We also assume that there is a limited number b of buffers available for

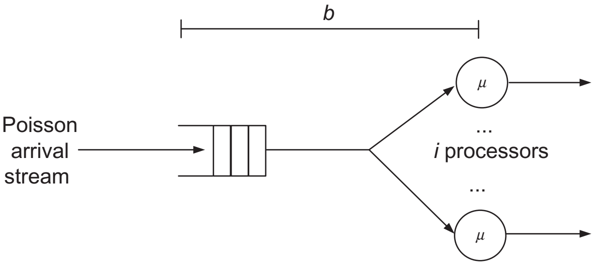


FIG. 28. Queuing system for the multiprocessor system.

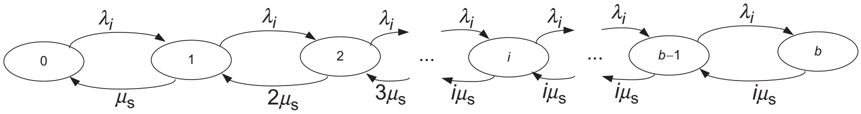


FIG. 29. Performance model for the multiprocessor system.

queuing the jobs. Tasks arriving when all the buffers are full are rejected. As explained earlier, to compute the NTL, we assign reward rates of a task being rejected to all the up state of the availability model by solving a sequence of lower-level performance models. Besides, we assign a reward rate 1 to all the down states. The task will be rejected whenever b tasks are in the system. Therefore, the NTL is given by the probability $q_b(i)$ that the CTMC (see Fig. 29) is in state b [5]:

$$q_b(i) = \begin{cases} \dfrac{\rho^b}{i^{b-i} i!} \left[\displaystyle\sum_{j=0}^{i-1} \frac{\rho^j}{j!} + \sum_{j=i}^{b} \frac{\rho^j}{i^{j-i} i!} \right]^{-1}, & b \geq i \\ \dfrac{\rho^b}{b!} \left[\displaystyle\sum_{j=0}^{b} \frac{\rho^j}{j!} \right]^{-1}, & b < i \end{cases}$$

where $\rho = \lambda/\mu$.

Figures 30 and 31 present the SHARPE input file to compute the NTL. Lines 4 through 13 define the input parameters. The parameter values for the high-level availability model are the same as the ones presented before to compute the steady-state unavailability. For the performance models, we assumed the jobs arrive at $\lambda_{\mathrm{j}} = 200$ jobs per second. Each job has a service rate of $\mu_{\mathrm{s}} = 100$ jobs per second. The number of buffer is $b = 3$. Thus,

```
format 8


bind
lambda_j   200
tau_r   1
gamma_f   1/6000
beta_r   12
mu_s   100
delta_r   360
c   0.95
n   4
end

markov perfMultProc4
0 1 lambda_j
1 2 lambda_j
1 0 mu_s
2 3 lambda_j
2 1 2*mu_s
3 4 lambda_j
3 2 3*mu_s
4 5 lambda_j
4 3 4*mu_s
5 6 lambda_j
5 4 4*mu_s
6 7 lambda_j
6 5 4*mu_s
7 6 4*mu_s
end
end

func R4()\
prob(perfMultProc4, 7)
markov perfMultProc3
0 1 lambda_j
1 2 lambda_j
1 0 mu_s
2 3 lambda_j
2 1 2*mu_s
3 4 lambda_j
3 2 3*mu_s
4 5lambda_j
4 3 3*mu_s
5 6 lambda_j
5 4 3*mu_s
6 5 3*mu_s
end
end

func R3()\
prob(perfMultProc3, 6)


markov perfMultProc2
0 1 lambda_j
1 0 mu_s
1 2 lambda_j
2 1 2*mu_s
2 3 lambda_j
3 2 2*mu_s
3 4 lambda_j
4 3 2*mu_s
4 5 lambda_j
5 4 2*mu_s
end
end

```

FIG. 30. SHARPE input for the multiprocessor system.

```
func R2()\
prob(perfMultProc2, 5)

markov perfMultProc1
0 1 lambda_j
1 2 lambda_j
1 0 mu_s
2 3 lambda_j
2 1 mu_s
3 4 lambda_j
3 2 mu_s
4 3 mu_s
end
end

func R1()\
prob(perfMultProc1, 4)


markov multProc
x4 3 delta_r
y4 3 beta_r
4 x4 n*gamma_f*c
4 y4 n*gamma_f*(1-c)
3 4 tau_r
3 x3 (n-1)*gamma_f*c
3 y3 (n-1)*gamma_f*(1-c)
x3 2 delta_r
y3 2 beta_r
2 x2 (n-2)*gamma_f*c
2 y2 (n-2)*gamma_f*(1-c)
2 3 tau_r
x2 1 delta_r
y2 1 beta_r
1 2 tau_r
1 0gamma_f
0 1 tau_r
* Reward configuration defined:
Reward
x4 1
y4 1
4  R4()
3  R3()
x3  1
y3  1
2  R2()
x2  1
y2  1
1  R1()
0 1
end
end

var NTL exrss(multProc)
echo NTL for the Multiprocessor System.
expr NTL

end
```

FIG. 31. SHARPE input for the multiprocessor system (continuation).

$$\mathrm{NTL} = \sum_{i\in S_{\mathrm{p}}} q_b(i) + U_{\mathrm{s}}$$

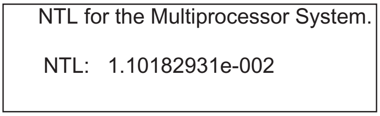
where $S_{\mathrm{p}}=\{i|1\leq i\geq n\}$.

Lines 15 through 105 define the performance models which are solved for each up state of the availability model. For instance, from lines 15 through 31, we define the performance model when 4 processors are operational. This performance model is solved to compute the job loss probability due to full buffers (function on lines 33 through 34). The function computes the probability that the CTMC is in state b. The result is passed as reward rates to the availability model (see line 110). This procedure is carried out for all the up states. Lines 122 through 124 compute the NTL for the multiprocessor system. Figure 32 shows the output for this model. Note that this model specification can be easily modified to consider different number of processors and different number of buffer using loops in the specification of CTMC as presented before for the telecommunication switching case.

3.2.2 Wireless Communication Network System

In Ref. [2], Ma et al. presented two techniques for combining performance and availability analysis in a wireless communication network system. The two modeling techniques used are an exact composite model and a hierarchical performability model. Figure 33 presents a monolithic CTMC model for a system with channel failure and repair. We assumed there are C idle channels and g guard channels. Note that the guard channels are reserved channels which are used by handoff calls only, since the dropping of a handoff call is more severe than the blocking of a new call. Interarrival times for the new and handoff calls are exponentially distributed with respective rates λ_n and λ_h^i. The departure of call due to termination or due to an ongoing call crossing a cell boundary is exponentially distributed with respective rates λ_d and λ_h^0. The channel failure and repair times are exponentially distributed with rates λ_f and μ_r, respectively. For the sake of simplicity, we assume $\lambda_t=\lambda_\mathrm{n}+\lambda_\mathrm{h}^i$, $\lambda_0=\lambda_\mathrm{d}+\lambda_\mathrm{h}^0$, $a=C-g$, $b=C-g+1$, and $q=C-1$.

Since the exact approach (see Fig. 33) generally faces largeness and stiffness problems, we advocate the use of hierarchical approaches. The performability model composed of two-level MRMs for the wireless communication network is shown in Fig. 34. The upper level model depicts the failure and repair behavior of the system (see Fig. 34A). The lower-level models describe the performance aspects of the system (see Fig. 34B and C). For each state i $(C,\ldots,C-g-1)$ on the upper level



```
NTL for the Multiprocessor System.

 NTL:   1.10182931e-002
```

FIG. 32. SHARPE output for the multiprocessor system.

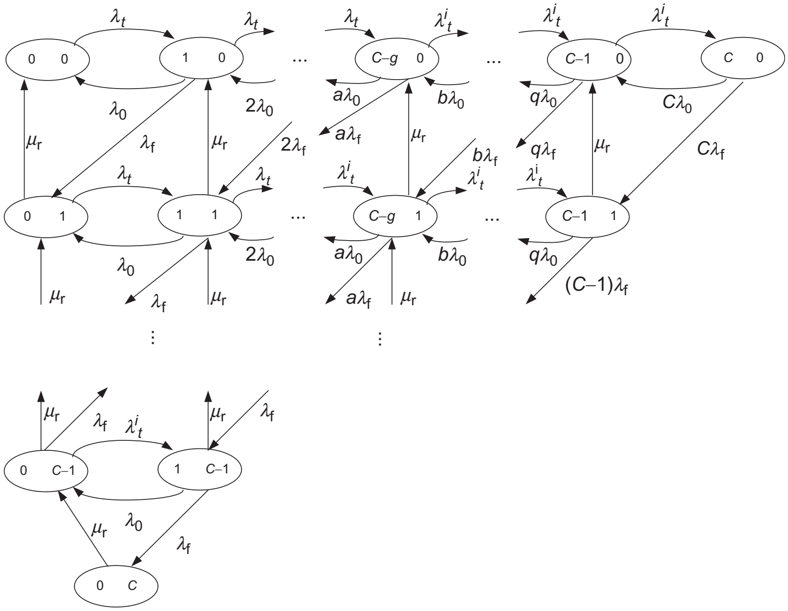


FIG. 33. Monolithic CTMC model for a wireless system with channel failure and repair.

model, the lower-level performance model described in Fig. 34B is solved to compute the dropping probability and blocking probability. These measures are then used as reward rates in the upper level model. Likewise, for each state *i* ($C-g$,. . .,1) on the availability model, the lower-level performance model described in Fig. 34C is solved, and the results are used as reward rates to the availability model. Note that the performance model in Fig. 34B represents the arrival of either new or handoff calls, while the performance model in Fig. 34C represents only handoff call arrival. This approach is an approximation of the exact model by assuming that in each state of the upper level, the lower level reaches steady state. Even though an approximation is involved in this process, the errors caused by the approximation are acceptable. Thus, the use of hierarchical models provides a good alternative for combining performance and availability analysis.

3.2.3 Job Completion Time

Yet another example of combined performance and reliability analysis is computation of the job completion time on a system subject to component failure and repair. The distribution of the job completion time on a computer system considering

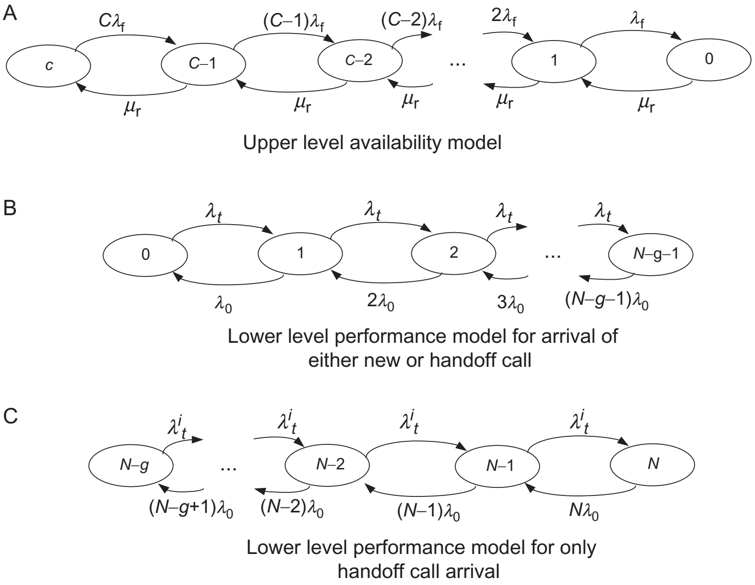


FIG. 34. Upper- and lower-level models for a wireless system.

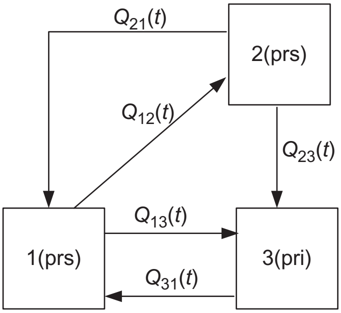


FIG. 35. Three-state CPU model.

CPU failure and repair was originally studied in Ref. [51]. The CPU or the server model used in the study was a three-state SMP, and the job completion time was analyzed in a general manner [29,52]. Figure 35 shows the SMP for the CPU model where state 1 represents the server up state; state 2 is the state where the server is recovering from a nonfatal failure; and state 3 is the state where the server is recovering from a fatal failure.

The state 1 and the state 2 are categorized as pre-emptive resume (prs) states in which the job execution is resumed from the interrupted point. On the other hand, the state 3 is categorized as pre-emptive repeat identical (pri) state in which the job execution is restarted from the beginning. A job that started execution when the server is in state 1 may encounter a nonfatal error that leads to the server state change from 1 to 2. The job execution also faces a fatal error that causes the server state change from 1 to 3. Both of nonfatal error and fatal error are repairable and their times to recovery follow general distribution $G_2(t)$ and $G_3(t)$, respectively. Assuming that failures are exponentially distributed with rate λ and each failure is either nonfatal with probability p_{nf} or fatal with probability $p_{\mathrm{f}}=1-p_{\mathrm{nf}}$, the SMP kernel distributions are given by following expressions.

$$Q_{12}(t) = p_{\mathrm{nf}}\cdot\left(1-\mathrm{e}^{-\lambda t}\right)$$

$$Q_{13}(t) = p_{\mathrm{f}}\cdot\left(1-\mathrm{e}^{-\lambda t}\right)$$

$$Q_{21}(t) = \int_0^t \mathrm{e}^{-\lambda p_{\mathrm{f}}\tau}\frac{\mathrm{d}}{\mathrm{d}\tau}G_2(\tau)\mathrm{d}\tau$$

$$Q_{23}(t) = \left(1-\mathrm{e}^{-\lambda p_{\mathrm{f}}\tau}\right) - \int_0^t \lambda\cdot p_{\mathrm{f}}\cdot\mathrm{e}^{-\lambda p_{\mathrm{f}}\tau}\frac{\mathrm{d}}{\mathrm{d}\tau}G_2(\tau)\mathrm{d}\tau$$

$$Q_{31}(t) = G_3(t)$$

Using the analysis method developed in Ref. [52] to the SMP model, we can obtain the Laplace–Stieltjes transforms (LSTs) of the job completion time distribution $F_1^{\sim}(s,x)$ for fixed work amount x:

$$F_1^{\sim}(s,x) = \frac{\mathrm{e}^{-\tau(s)x}}{1-\frac{\lambda p_{\mathrm{f}}}{\mathrm{s}+\lambda p_{\mathrm{f}}}\cdot\left[1-\mathrm{e}^{-\tau(s)x}\right]\cdot G_3^{\sim}(s)}$$

$$\tau(s) = s + \lambda\left(1 - p_{\mathrm{nf}}\cdot Q_{21}^{\sim}(s)\right)$$

where $Q_{21}^{\sim}(s)$ and $G_3^{\sim}(s)$ are the LSTs of $Q_{21}(t)$ and $G_3(t)$.

Then LST can be numerically inverted, or by taking derivatives expected completion time determined.

4. Conclusions

As a result of the proliferation and complexity of computer systems, the use of composite performance and availability analysis has become an important method to analyze degradable systems. Pure performance evaluation of a system tends to be optimistic since it ignores the failure–repair behavior of the system. On the other hand, pure availability analysis tends to be conservative, since performance considerations are not properly taken into account. In order to understand and predict the behavior of these systems, this chapter introduced the main techniques used in model construction and solution of composite performance and availability analysis, such as exact composite approach and hierarchical modeling approaches. We have also shown the basics of analytic models which are useful to compute availability and performance metrics by themselves. To show the applicability of these methods, practical examples have been detailed using the SHARPE software package.

References

[1] G. Ciardo, R. Marie, B. Sericola, K.S. Trivedi, Performability analysis using semi-Markov reward processes, IEEE Trans. Comput. 39 (10) (1990) 1251–1264.

[2] Y. Ma, J. Han, K. Trivedi, Composite performance & availability analysis of wireless communication networks, IEEE Trans. Veh. Technol. 50 (5) (2001) 1216–1223.

[3] K.S. Trivedi, J.K. Muppala, S.P. Woolet, B.R. Haverkort, Composite performance and dependability analysis, Perform. Eval. 14 (3–4) (1992) 197–215.

[4] R.A. Sahner, K.S. Trivedi, A. Puliafito, Performance and Reliability Analysis of Computer Systems: An Example-Based Approach Using the SHARPE Software Package, Kluwer Academic Publishers, Dordrecht/Boston/London, 1996.

[5] R.M. Smith, K.S. Trivedi, A.V. Ramesh, Performability analysis: measures, an algorithm, and a case study, IEEE Trans. Comput. 37 (4) (1988) 406–417.

[6] J.K. Muppala, S.P. Woolet, K.S. Trivedi, Real-time systems performance in the presence of failures, IEEE Comput. 24 (1991) 37–47.

[7] Y. Cao, H. Sun, K.S. Trivedi, Performability analysis of TDMA cellular systems, International Conference on the Performance and QoS of Next Generation Networking, P&QNet2000, Nagoya, Japan, 2000.

[8] R. Ghosh, K.S. Trivedi, V.K. Naik, D. Kim, End-to-end performability analysis for infrastructure-as-a-service cloud: An interacting stochastic models approach, Proceedings of the 16th IEEE Pacific Rim International Symposium on Dependable Computing (PRDC), 2010, pp. 125–132.

[9] S. Ramani, K. Goseva-Popstojanova, K.S. Trivedi, A framework for performability modeling of messaging services in distributed systems, Proceedings of the 8th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 02), Greenbelt, MD, pp.25–34, 2002.

[10] N. Lopez-Benitez, K.S. Trivedi, Multiprocessor performability analysis, IEEE Trans. Reliab. 42 (4) (1993) 579–587.

[11] K.S. Trivedi, X. Ma, S. Dharmaraja, Performability modeling of wireless communication systems, Int. J. Commun. Syst. 16 (6) (2003) 561–577.

[12] M. Lanus, L. Yin, K.S. Trivedi, Hierarchical composition and aggregation of state-based availability and performability models, IEEE Trans. Reliab. 52 (1) (2003) 44–52.

[13] D. Wang, W. Xie, K.S. Trivedi, Performability analysis of clustered systems with rejuvenation under varying workload, Perform. Eval. 64 (3) (2007) 247–265.

[14] B. Haverkort, R. Marie, G. Rubino, K.S. Trivedi, Performability Modeling Tools and Techniques, John Wiley & Sons, Chichester, England, 2001.

[15] M.D. Beaudry, Performance-related reliability measures for computing systems, IEEE Trans. Comput. C-27 (Jun. 1978) 540–547.

[16] J.F. Meyer, On evaluating the performability of degradable computing systems, IEEE Trans. Comput. 29 (8) (Aug. 1980) 720–731.

[17] K.S. Trivedi, D. Wang, D.J. Hunt, A. Rindos, W.E. Smith, B. Vashaw, Availability modeling of SIP protocol on IBM WebSphere, Proceeding Pacific Rim Dependability Conference, 2008, pp. 323–330.

[18] K.S. Trivedi, D. Wang, J. Hunt, Computing the number of calls dropped due to failures, Proceedings of the IEEE International Symposium on Software, Reliability Engineering, 2010, pp. 11–20.

[19] A. Bobbio, K.S. Trivedi, Computing cumulative measures of stiff Markov chains using aggregation, IEEE Trans. Comput. 39 (Oct. 1990) 1291–1298.

[20] M. Malhotra, J.K. Muppala, K.S. Trivedi, Stiffness-tolerant methods for transient analysis of stiff Markov chains, Microelectron Reliab. 34 (11) (1994) 1825–1841.

[21] F. Longo, R. Ghosh, V.K. Naik, K.S. Trivedi, A scalable availability model for Infrastructure-as-a-Service cloud, IEEE/IFIP DSN, 2011.

[22] G. Bolch, S. Greiner, H. Meer, K.S. Trivedi, Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications, Wiley Interscience, New York, NY, 1998.

[23] K.S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications, second ed., John Wiley and Sons, New York, 2001.

[24] D. Logothesis, K.S. Trivedi, A. Puliato, Markov regenerative models, Proceedings of the International Computer Performance and Dependability Symposium, Erlangen, Germany, 1995, pp. 134–143.

[25] B.R. Haverkort, Approximate performability and dependability analysing using generalized stochastic Petri nets, Perform. Eval. 18 (1993) 61–78.

[26] V. Mainkar, K.S. Trivedi, Fixed point iteration using stochastic reward nets, Proceedings of the 6th International Workshop on Petri Nets and Performance Models (PNPM), Durham, USA, pp. 21–30, 1995.

[27] ITU-T Recommendation E.800. Terms and definitions related to quality of service and network performance including dependability. http://wapiti.telecom-lille1.eu/commun/ens/peda/options/ST/RIO/pub/exposes/exposesrio2008-ttnfa2009/Belhachemi-Arab/files/IUT-T%20E800.pdf. Accessed 18 May 2011.

[28] M. Grottke, K.S. Trivedi, Fighting bugs: remove, retry, replicate and rejuvenate, IEEE Comput. 40 (2) (2007) 107–109.

[29] A. Sathaye, S. Ramani, K.S. Trivedi, Availability models in practice, Proceedings of the International Workshop on Fault-Tolerant Control and Computing (FTCC-1), 2000.

[30] X. Zang, D. Wang, H. Sun, K.S. Trivedi, A BDD-based algorithm for analysis of multistate systems with multistate components, IEEE Trans. Comput. 52 (12) (Dec. 2003) 1608–1618.

[31] D. Wang, K. Trivedi, T. Sharma, A. Ramesh, D. Twigg, L. Nguyen, Y. Liu, A new reliability estimation method for large systems, 2000 The Boeing Company patent application pending.

[32] M. Malhotra, K.S. Trivedi, Power-hierarchy of dependability-model types, IEEE Trans. Reliab. 43 (3) (1994) 34–42.

[33] S.S. Gokhale, M.R. Lyu, K.S. Trivedi, Analysis of software fault removal policies using a non-homogeneous continuous time Markov chain, Softw. Qual. J. 12 (3) (2004) 211–230.

[34] W.J. Stewart, Probability, Markov Chains, Queues, and Simulation, Princeton University Press, USA, 2009.

[35] M. Malhotra, K. Trivedi, Dependability modeling using Petri nets, IEEE Trans. Reliab. 44 (3) (1995) 428–440.

[36] K. Trivedi, G. Ciardo, M. Malhutra, R. Sahner, Dependability and performability analysis, in: Donatiello Lorenzo, Nelson Randolf (Eds.), Performance Evaluation of Computer and Communication Systems, Springer-Verlag, NY, USA, 1993.

[37] G. Ciardo, J. Muppala, K.S. Trivedi, Analyzing concurrent and fault-tolerant software using stochastic Petri nets, J. Parallel Distrib. Comput. 15 (1992) 255–269.

[38] M. Balakrishnan, K.S. Trivedi, Stochastic Petri nets for the reliability analysis of communication network applications with alternate-routing, Reliab. Eng. Syst. Safety 52 (3) (1996) 243–259 special issue on Reliability and Safety Analysis of Dynamic Process Systems.

[39] J.B. Dugan, K.S. Trivedi, V. Nicola, R. Geist, Extended stochastic Petri nets: applications and analysis, Proceeding Performance '84, North-Holland, Amsterdam, pp. 507–519, 1985.

[40] K.S. Trivedi, Sun Hairong, Stochastic Petri nets and their applications to performance analysis of computer networks, Proceedings of the International Conference on Operational Research, 1998.

[41] O. Ibe, R. Howe, K.S. Trivedi, Approximate availability analysis of VAXCluster systems, IEEE Trans. Reliab. 38 (1) (Apr. 1989) 146–152.

[42] R. Fricks, K.S. Trivedi, Modeling failure dependencies in reliability analysis using stochastic Petri nets, Proceedings of the European Simulation Multi-conference (ESM '97), Istanbul, 1997.

[43] R. German, C. Kelling, A. Zimmermann, G. Hommel, TimeNET—a toolkit for evaluating non-Markovian stochastic Petri nets, Perform. Eval. 24 (1995) 69–87.

[44] G. Chiola, G. Franceschinis, R. Gaeta, M. Ribaudo, GreatSPN 1.7: graphical editor and analyzer for timed and stochastic Petri nets, Perform. Eval. 24 (Nov. 1995) 47–68.

[45] K.S. Trivedi, R. Sahner, SHARPE at the age of twenty two, SIGMETRICS Perform. Eval. Rev. 36 (4) (2009) 52–57.

[46] C. Hirel, B. Tuffin, K.S. Trivedi, SPNP: stochastic Petri nets. Version 6.0, Lect. Notes Comput. Sci. 1786 (2000) 354–357.

[47] A. Bobbio, A. Puliafito, M. Telek, K.S. Trivedi, Recent developments in stochastic Petri nets, J. Circuits Syst. Comp. 8 (1) (Feb. 1998) 119–158.

[48] K.S. Trivedi, D. Kim, X. Yin, Multi-state availability modeling in practice, in: A. Lisnianski, I. Frenkel (Eds.), Recent Advances in System Reliability: Signature, Multi-state Systems and Statistical Inference, Springer, New York, 2011.

[49] J.K. Muppala, A. Sathaye, R. Howe, K.S. Trivedi, Dependability modeling of a heterogeneous VAXcluster system using stochastic reward nets, in: D. Avresky (Ed.), Hardware and Software Fault Tolerance in Parallel Computing Systems, Ellis Horwood Ltd, NJ, USA, 1992, pp. 33–59.

[50] Y. Liu, K.S. Trivedi, Survivability quantification: the analytical modeling approach, International J. Performability Eng. (2006) 29–44.

[51] X. Castillo, D.P. Siewiorek, A performance-reliability model for computing systems, Proceedings of the FTCS-10, Silver Spring, MD, IEEE Computer Society, 1980, pp. 187–192.

[52] P. Chimento, K. Trivedi, The completion time of programs on processors subject to failure and repair, IEEE Trans. Comput. 42 (10) (1993) 1184–1194.

About the Author

Kishor S. Trivedi holds the Hudson Chair in the Department of Electrical and Computer Engineering at Duke University, Durham, NC. He has been on the Duke faculty since 1975. He is the author of a well-known text entitled, *Probability and Statistics with Reliability, Queuing and Computer Science Applications*, published by Prentice-Hall; a thoroughly revised second edition (including its Indian edition) of this book has been published by John Wiley. He has also published two other books entitled, *Performance and Reliability Analysis of Computer Systems*, published by Kluwer Academic Publishers and *Queueing Networks and Markov Chains*, John Wiley. He is a Fellow of the Institute of Electrical and Electronics Engineers. He is a Golden Core Member of IEEE Computer Society. He has published over 450 articles and has supervised 42 Ph.D. dissertations. He is on the editorial boards of *Journal of Risk and Reliability*, *International Journal of Performability Engineering*, and *International Journal of Quality and Safety Engineering*. He is the recipient of IEEE Computer Society Technical Achievement Award for his research on Software Aging and Rejuvenation. His research interests are in reliability, availability, performance, performability, and survivability modeling of computer and communication systems. He works closely with industry in carrying our reliability/availability analysis, providing short courses on reliability, availability, performability modeling, and in the development and dissemination of software packages such as SHARPE and SPNP.

Ermeson C. Andrade graduated in Computer Science from Catholic University of Pernambuco in 2006 and received his M.Sc. degree in Computer Science at Federal University of Pernambuco in 2009. He is currently a Ph.D candidate in Computer Science at Federal University of Pernambuco and visiting scholar of Electrical and Computer Engineering at Duke University. His research interests include performability analysis, component-based modeling, and hard real-time systems.

Fumio Machida is an assistant manager in NEC Service Platforms Research Laboratories. He was a visiting scholar of Electrical and Computer Engineering in Duke University in 2010. His primary research interest is in availability management of large-scale ICT systems such as data centers for cloud computing services. He has experienced real industrial research projects on enterprise system management, server virtualization management, autonomic computing, and dependable computing. He has also contributed industrial standardization activities such as Open Virtualization Format (OVF), Integrated Access Control Policy Management (IAM) in Distributed Management Task Force (DMTF).