Reliability Modeling Using SHARPE

Robin A. Sahner
Gould CSD, Urbana
Kishor S. Trivedi, Member IEEE
Duke University, Durham

Key Words—**Combinatorial model, Hierarchical model, Markov model, Reliability modeling.**

Reader Aids—
Purpose: To demonstrate a reliability model
Special math needed for explanations: Elementary probability, Markov chains
Special math needed to use results: None
Results useful to: Reliability engineers and analysts

Summary & Conclusions—**Combinatorial models such as fault trees and reliability block diagrams are efficient for model specification and often efficient in their evaluation. But it is difficult, if not impossible, to allow for dependencies (such as repair dependency and near-coincident-fault type dependency), transient and intermittent faults, standby systems with warm spares, and so on. Markov models can capture such important system behavior, but the size of a Markov model can grow exponentially with the number of components in the system.**

This paper presents an approach for avoiding the large state-space problem. The approach uses a hierarchical modeling technique for analyzing complex reliability models. It allows the flexibility of Markov models where necessary and retains the efficiency of combinatorial solution where possible. Based on this approach a computer program called SHARPE (Symbolic Hierarchical Automated Reliability and Performance Evaluator) has been written.

The hierarchical modeling technique provides a very flexible mechanism for using decomposition and aggregation to model large systems; it allows for both combinatorial and Markov or semi-Markov submodels, and can analyze each model to produce a distribution function. The choice of the number of levels of models and the model types at each level is left up to the modeler. Component distribution functions can be any exponential polynomial whose range is between zero and one.

Examples show how combinations of models can be used to evaluate the reliability and availability of large systems using SHARPE.

1. INTRODUCTION

Combinatorial models (such as fault trees and reliability block diagrams) are widely used to predict analytically the reliability or availability of fault-tolerant systems. These models are easy to construct and understand. If component states are *s*-independent, reliability block diagrams with *series-parallel* (*well-nested*) structure and fault trees without repeated nodes can be analyzed using linear-time algorithms. However, if component states are not *s*-independent or the structure is not *series-parallel*, the analysis cannot in general be done in linear time. Methods for analyzing such structures include conditioning (using the theorem of total probability) [1] and Markov models. Both methods are expensive; if there are n components, the corresponding set of conditional cases or Markov chain states can have size up to 2^n. Since most real-world problems do not satisfy the assumptions of *s*-independence and *series-parallel* structure, it is important to investigate ways of dealing with systems that generate very large state spaces.

There are two aspects to the problem of handling a Markov model with a large number of states: construction and analysis. The problem of accurate construction can be handled by automatic generation of the Markov chain from a more concise (hence less error-prone) model such as a fault tree or Petri net. Examples of programs that do this are HARP (Hybrid Automated Reliability Predictor) [2], SAVE (System AVailability Estimator) [3] and DEEP (Duke Evaluator for Extended stochastic Petri nets) [4]. The transition rate matrices of large Markov chains are usually sparse, but are often also stiff (the transition rates can differ by many orders of magnitude). Numerical methods exist for solving large, sparse, possibly stiff systems of algebraic [3] or ordinary differential equations [5]. It is also possible to use matrix-level decomposition/aggregation methods [6].

An alternative way to deal with the large state-space problem is to avoid it. That is, the system under investigation is modeled in such a way that the large state space is never produced. One way to do this is to use model-level decomposition. This is the method used in CARE III [7] and HARP [2, 8], where reliability models are decomposed behaviorally along temporal lines. The fault-occurrence behavior (a slow submodel) and the fault/error-handling behavior (a fast submodel) are analyzed separately.

We have developed a hierarchical modeling technique that makes it possible to use mixtures of different kinds of models at different levels to avoid a state-space explosion. Our technique differs from models such as HARP and CARE III in several ways. HARP and CARE III assume a specific fixed hierarchy of models geared toward modeling a chosen class of systems. Our technique allows complete freedom in the number of levels in the hierarchy, which kinds of models to use at each level, and how to combine the models. The previous efforts have used numerical methods to obtain the reliability for a specified mission time. The results therefore had to be recomputed for each submodel, each set of parameters, and each value of the mission time. Our technique provides a result that is symbolic in the mission time variable, so each submodel need be analyzed only once for each set of model parameters.

Work is underway to improve and extend SHARPE. Possible extensions include adding loops to the input

0018-9529/87/0600-0186$01.00

language, allowing fault trees to have repeated nodes, and extending the list of model types to include irreducible semi-Markov chains and one or more queueing network models.

In the next section, we introduce our approach to hierarchical modeling. In section 3, we discuss the use of SHARPE to analyze simple reliability block diagrams and in section 4, we show how to use SHARPE to analyze a non-*series-parallel* structure. In the remaining sections, we discuss the use of hierarchical models to model repair dependency, near-coincident-fault dependency, and performance-reliability dependency.

The words *series, parallel, series-parallel* refer to connections in a logic diagram, not to a schematic or layout diagram.

2. THE SHARPE FRAMEWORK

We have developed a hybrid, hierarchical modeling framework, implemented as a software tool, that we call SHARPE. Basic, built-in model types can be combined hierarchically in a flexible manner, with the number and types of models at each level and the particular information carried between the models left up to the modeler. Components in each model type are assigned functions that are symbolic in the time variable t. The analysis of each model type is carried out symbolically, resulting in another function that is symbolic in t.

The SHARPE framework provides seven model types:

1. *series-parallel* reliability block diagrams
2. fault trees without repeated nodes
3. acyclic Markov chains
4. irreducible cyclic Markov chains
5. cyclic Markov chains with absorbing states
6. acyclic semi-Markov chains
7. *series-parallel* directed (acyclic) graphs

Block diagram and fault tree models are specialized for modeling reliability and availability. The other model types can be used to model performance as well. This paper focuses on reliability modeling; for examples of the use of SHARPE for performance modeling, see [9].

To model reliability, each component in a block diagram or fault-tree model can be assigned either a Cdf for its time-to-failure or a simple probability that the component has failed. The analysis of the system produces the Cdf of the time-to-failure of the system as a whole. To model availability, each component can be assigned its instantaneous unavailability as a function of t. The analysis of the system produces the instantaneous unavailability function for the system as a whole.

Markov and semi-Markov chains that have absorbing states are analyzed for the Cdf of the time to absorption. If such a chain is acyclic, the analysis also produces the probability of ever visiting each state. Irreducible cyclic Markov chains are analyzed for the steady-state probabilities of being in each state.

The *series-parallel* graph submodel is the most general model. It is similar to, but more general than, *series-parallel* stochastic PERT (Program Evaluation Review Technique) networks [10]. In the graph model, nodes represent activities and arcs represent precedence constraints placed on the activities. *Parallel* paths through the graph can be interpreted as being executed either concurrently or probabilistically (one path is chosen). Each node in the graph is assigned a Cdf, and the graph is analyzed for the Cdf of its time to completion.

Submodels are combined hierarchically by using part of the analysis of one submodel as part of the specification of another submodel. This mechanism allows for both exact and approximate decomposition/aggregation. One method for combining submodels is to assign the result Cdf from a submodel as the Cdf for a basic component in another model. This method allows us to extract non-*series-parallel* portions of a logic structure and pay the price of 2^n states to analyze them exactly using Markov chains. Then we use the results from those portions as the Cdf's of basic aggregate components in the remaining structure and analyze it using a combinatorial solution method. This results in an exact solution.

Information can also be passed between submodels by means of one or more of the scalar (as opposed to Cdf) quantities produced during the analysis of a submodel. SHARPE makes available the mean and variance of each distribution produced by the analysis of a system, the value of each Cdf at specified values of t (including $t = 0$ and $t = \infty$), and state probabilities for Markov and semi-Markov chains. These scalars can be used in other models as elements in the expressions that specify probability values, transition rates, and the parameters of Cdf's. This mechanism allows approximate decomposition.

To work with symbolic expressions for the Cdf's, we need a class of functions that is closed under convolution, order statistics [11] of s-independent (not necessarily identically distributed) random variables, and mixing of distributions (weighted averages). We have chosen to use the class of exponential polynomials that are valid Cdf's. An exponential polynomial is a function with the form:

$$\sum_{j=1}^{n} a_j t^{k_j} e^{b_j t} \tag{1}$$

where k_j is a nonnegative integer and a_j and b_j are real or complex numbers. (A Cdf must be real-valued, even if it contains complex terms). A cyclic Markov chain with absorbing states can have complex numbers in the Cdf for its time-to-absorption.

We allow the distributions to have a mass at zero and/or a mass at infinity. A distribution has a mass at infinity if it does not reach one in the limit, and in that case, it is called *defective*. Defective distributions are useful for modeling the performance of programs that might fail to complete because of either software or hardware faults. An example of such a model is given in section 7. Distributions

with a mass at zero can be assigned to components that can have a latent fault. It is possible to use distributions that have mass only at zero and infinity. Assigning such a distribution to a component is equivalent to assigning to the component a probability value (at some implicit time t) rather than a Cdf. Examples of the use of such distributions are given in sections 3 and 5.

The SHARPE program consists of about 4800 lines of code in *C* [12] and runs under UNIX or VMS. It is a full implementation of the modeling technique and has been tested extensively. The specification language supports general well-formed arithmetic expressions and user-defined functions and Cdf's of any number of parameters.

3. EMULATING RELCOMP

The Relcomp program [13] is a reliability block-diagram analyzer. With the exception that SHARPE cannot handle the Weibull distribution, SHARPE can analyze any Relcomp-type system, and the flexibility of the SHARPE interface allows it to present a user interface similar to that of Relcomp. This emulation uses only a small portion of the power of SHARPE, but it is a good introduction to its use.

Relcomp uses numerical integration (Simpson's rule) to compute the probability of survival until a user-specified time t for a *series* system in which there is no repair. Redundancy in the form of active or standby spare units is brought in by allowing the basic components in the *series* system to have reliability functions chosen from a set of pre-computed functions including those in table 1. A Relcomp-type system from [13] is shown in figure 1. Relcomp computes the probability that the system is functioning at time $t = 20$ as 0.99581.

TABLE 1
Relcomp Reliability Functions

$R(t) = \exp(-\lambda t)$	single exponential, with failure rate λ
$R(t) = 2\exp(-\lambda t) - \exp(-2\lambda t)$	active redundancy, 2 units with equal failure rates λ
$R(t) = (\exp(-\lambda_a t) + \exp(-\lambda_b t) - \exp[-(\lambda_a + \lambda_b)t]$	active redundancy, 2 units with unequal failure rates λ_a (primary) and λ_b (secondary)
$R(t) = \exp(-\lambda t) + \exp(-\lambda_s t)\lambda t \exp(-\lambda t)$	standby redundancy, 2 units with equal failure rates λ and a sensing switch with failure rate λ_s
$R(t) = \exp(-\lambda_a t) + \left[\frac{\lambda_a}{\lambda_b - \lambda_a}\right](\exp(-\lambda_a t) - \exp(-\lambda_b t))$	standby redundancy, 2 units with unequal failure rates λ_a (primary) and λ_b (secondary) and a sensing switch with failure rate λ_s
$R(t) = P$	single, one-shot system, with probability P of success
$R(t) = \sum_{i=0}^{m} \binom{n}{i} (\exp(-\lambda t))^{n-i} (1 - \exp(-\lambda t))^i$	binomial (m-out-of-n) system), with equal failure rates λ

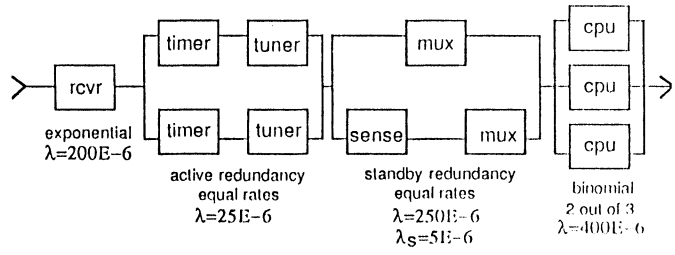

Fig. 1. A Relcomp-type System.

SHARPE can emulate the Relcomp program by using only type 1 models (reliability block diagrams). The file in table 2 is a SHARPE input file. (Line numbers are included only for the sake of exposition.) The SHARPE input language is described in detail in [14]. On lines 3 through 33, we define the Relcomp built-in distribution functions from table 1. (Relcomp uses reliability functions while SHARPE uses failure-time functions.) Each of the first four distributions is defined by giving all of the exponential polynomial terms a_j, k_j, b_j; see (1). Line 25 defines the component type *oneshot*, which is assigned a probability value (the probability of immediate failure) rather than a distribution function. The binomial distribution (line 32) is defined to be the Cdf of the system *KN*, which is a $(n - k + 1)$-out-of-n:F system.

Lines 35 through 41 define the system in figure 1. Line 43 asks that the Cdf for the time-to-failure of the system be printed. To obtain the probability of being operational at time $t = 20$, we must evaluate the Cdf at $t = 20$ and subtract from one. Line 44 asks SHARPE to do this. The result agrees with that produced by Relcomp. Table 3 shows the output for the input file of table 2. These results were produced in less than 0.1 second on a lightly loaded Gould PowerNode 9080 running UTX/32.

TABLE 2
Input File for Relcomp-type System

```
1  * emulating the RELCOMP program
2
3  poly activeE (lambda) gen\
4      1,0,0\
5      -2,0,-lambda\
6      1,0,-2*lambda
7
8  poly activeU (lambda,mu) gen\
9      1,0,0\
10     -1,0,-lambda\
11     -1,0,-mu\
12     1,0,-(lambda+mu)
13
14 poly standbyE (lambda, s) gen\
15     1,0,0\
16     -1,0,-lambda\
17     -lambda,1,-(lambda+s)
18
19 poly standbyU (lambda,mu,s) gen\
20     1,0,0\
21     -1,0,-lambda\
22     -lambda/(mu-lambda), 0, -(lambda+s)\
23     lambda/(mu-lambda), 0, -(mu+s)
24
25 poly oneshot(p) prob(1-p)
26
27 block KN (lambda, k, n)
28 comp x exp(lambda)
29 kofn top n-k+1,n, x
30 end
31
32 poly binomial (lambda, k, n)\
33     cdf (KN;lambda,k,n)
34
35 block DP
36 comp receiver exp(.0002)
37 comp tuner    activeE(.000025)
38 comp mux      standbyE(.00025, .000005
39 comp cpu      binomial(.0004, 2, 3)
40 series DP receiver tuner mux cpu
41 end
42
43 cdf(DP)
44 expr 1-value(20;DP)
45 end
```

The power of SHARPE can extend the Relcomp model to include k-out-of-n systems where the components are *not* identically distributed, and (by using Markov submodels) redundant systems with varying numbers of active and standby spares.

TABLE 3
Output from SHARPE for the File in Table 2

```
CDF for system DP:

   -1.5000e-03 t(  1) exp(-1.2800e-03 t)
 +  7.5000e-04 t(  1) exp(-1.3050e-03 t)
 +  1.0000e-03 t(  1) exp(-1.6800e-03 t)
 + -5.0000e-04 t(  1) exp(-1.7050e-03 t)
 +  1.0000e+00 t(  0) exp( 0.0000e+00 t)
 + -6.0000e+00 t(  0) exp(-1.2750e-03 t)
 +  3.0000e+00 t(  0) exp(-1.3000e-03 t)
 +  4.0000e+00 t(  0) exp(-1.6750e-03 t)
 + -2.0000e+00 t(  0) exp(-1.7000e-03 t)

mean: 1.4794e+03
variance: 1.2465e+06

----------------------------------------

 1-value(20;DP):    9.9581e-01
```

4. A NON-SERIES-PARALLEL BLOCK DIAGRAM

Suppose we want to compute the Cdf of the time-to-failure of the system shown in figure 2. Because it has two non-*series-parallel bridge* subsystems, this system cannot be analyzed using a common block diagram method like Relcomp. We need to analyze the *bridge* subsystems separately. We can then replace each *bridge* subsystem by a single aggregate component, resulting in a *series-parallel* block diagram that can be analyzed easily for an exact result.

There are several methods available for analyzing a *bridge* subsystem. If all of the components had exponential failure-time distributions, we could use a Markov chain. Instead, we have chosen to pre-analyze the bridge system by hand, using conditioning on whether the crossover component (3 in the bottom subsystem) is functioning. This method can be used whether the component time-to-failure distributions are exponential or not (here we assume they are exponential). A SHARPE specification for the system is shown in table 4. After defining the Cdf *bridge*, having five parameters, we define a reliability block diagram containing aggregate nodes A and B in place of the two *bridge* subsystems; the Cdf attached to each aggregate node is *bridge* called with appropriate parameters.

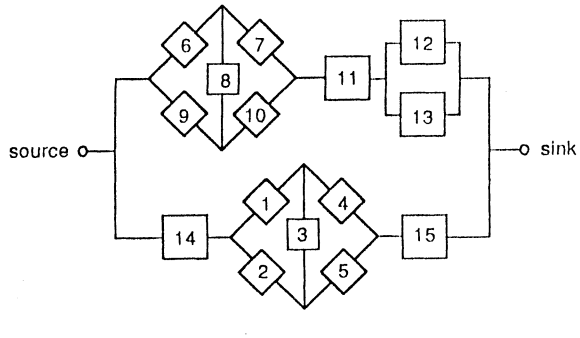

Fig. 2. A Non-Series-Parallel Reliability Diagram.

TABLE 4
Input for Non-Series-Parallel System

```
poly bridge(u1,u2,u3,u4,u5) gen\
 1,0,0\
 -1,0,-(u1+u4)\
 -1,0,-(u2+u5)\
 -1,0,-(u1+u3+u5)\
 -1,0,-(u2+u3+u4)\
 1,0,-(u1+u2+u3+u4)\
 1,0,-(u1+u2+u3+u5)\
 1,0,-(u1+u2+u4+u5)\
 1,0,-(u1+u3+u4+u5)\
 1,0,-(u2+u3+u4+u5)\
 -2,0,-(u1+u2+u3+u4+u5)
```

```
block b
comp 11 exp (u11)
comp 12 exp (u12)
comp 14 exp (u14)
comp 13 exp (u13)
comp 15 exp (u15)
comp A bridge(u1,u2,u3,u4,u5)
comp B bridge(u6,u7,u8,u9,u10)
parallel C 12 13
series D 11 B C
series E 14 A 15
parallel top D E
end
```

5. REPAIR DEPENDENCE

Consider the Carnegie Multiprocessor system, C.mmp, having 16 processors, 16 memories, and a switch [15]. The system is up if the switch, at least four processors, and at least four memory units are up. The condition under which the system is down is expressed by the fault tree (a type 2 model) in figure 3. Based on a parts count method and the use of MIL-HDBK data, the failure rate of each processor is 68.9 failures/10^6 hours, that of each memory unit 224/10^6 hours and that of the switch 202/10^6 hours [15].

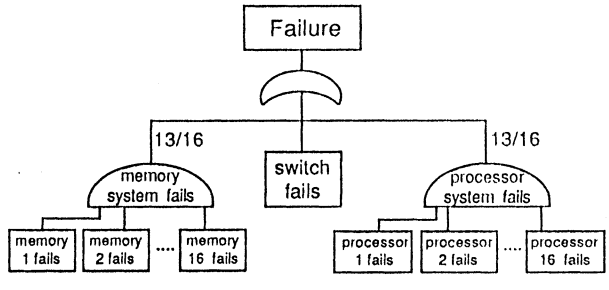

Fig. 3. Fault Tree for C.mmp System.

If the units are repairable and each individual unit has its own repair facility, we can assign to each component its instantaneous unavailability function [1]:

$$U_i(t) = \frac{\lambda_i}{\lambda_i + \mu_i} \exp[-(\lambda_i + \mu_i)t] \tag{2}$$

where λ_i is the failure rate, and μ_i is the repair rate. The $U_i(t)$ is defective. The SHARPE analysis of the system produces the system instantaneous unavailability. Setting $\mu_i = 0$ in (2) yields the Cdf for the time-to-failure of the component. We may allow some components to have a repair facility and others to be non-repairable by letting μ_i be zero for some components and nonzero for others. If we assign to each component its steady-state unavailability,

$$\lim_{t \to \infty} U_i(t) = \frac{\lambda_i}{\lambda_i + \mu_i} \tag{3}$$

the SHARPE analysis produces the system steady-state unavailability. In this case we are assigning to each component a probability value, instead of a probability function with a time parameter.

If repair facilities are shared within each subsystem instead of having a separate repair facility for each unit, component behavior is no longer *s*-independent. We can avoid a large Markov model by exploiting the *s*-independence across subsystems; we can model each subsystem as an independent Markov chain and still model the overall system by a fault tree. Figure 4 shows the Markov chain (a type 4 model) for the memory subsystem. Each state name gives the number of failed memories in the system. The steady-state unavailability for the subsystem is the sum of the steady-state probabilities for states 13-16. To incorporate this lower-level model into the fault tree, we replace the gate *memory system fails* and all of its inputs by a basic event assigned the above sum of state probabilities as its steady-state unavailability. The processor subsystem and switch are handled similarly.

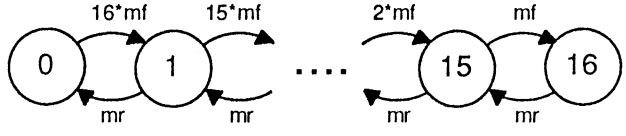

Fig. 4. Markov Chain for Memory Subsystem.

Table 5 compares the steady-state availability for repair-per-component versus repair-per-subsystem for various memory and processor average repair times (the average repair time for the switch was assumed to be 2 hours throughout). As anticipated, steady-state availability decreases as repair time increases, and the steady-state availability is higher for repair-per-component. In fact, we had to increase the repair times to unrealistic amounts to see the differences. This is because given the amount of redundancy and degree of reliability of the processors and memories, the availability of the system is determined almost entirely by the failure and repair times for the single switch. The results in table 5 were produced in about one second by SHARPE running on a Gould PowerNode 9080.

TABLE 5
Effects of Sharing Repair Facilities

average memory repair time (hours)	average processor repair time (hours)	steady-state availability (repair per component)	steady-state availability (repair per subsystem)
1	1	.999596163	.999596163
15	20	.999596163	.999596163
100	100	.999596163	.999596162
200	200	.999596163	.999592057
1000	1000	.999596080	.651090056

6. MODELING INTERMITTENT AND NEAR-COINCIDENT FAULTS

This example has two purposes: it shows how the dependency introduced by intermittent and near-coincident faults can be modeled, and it illustrates the flexibility of the SHARPE hierarchical combination mechanism. We begin with a single-level model and add levels as we show how the model can be refined and how aggregation can be expressed in the SHARPE framework.

We consider the flight control system modeled in appendix G of [16]. The system contains three inertial reference sensors (IRS), three pitch rate sensors (PRS), four computer systems (CS), and three secondary actuators (SA). At least two of each type of component must be operational for the overall system to function correctly. If all components operate (and fail) *s*-independently and system reconfiguration after a failure is perfect, we can model the system using the fault tree in figure 5. SHARPE calculates that the probability of failure during an interval of 10 hours is 1.02×10^{-6}.

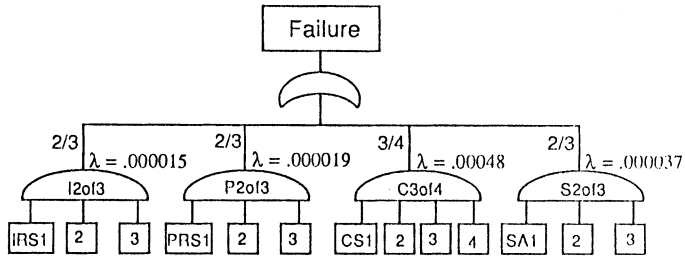

Fig. 5. Aircraft Control System: Fault Tree.

The fault-tree model does not take into account the possibility of intermittent faults or the possibility that the system might not be able to recover properly from an error. We can use a 2-level model to include these possibilities. Because in this system the computers are most susceptible to failure, we model the computer subsystem more exactly.

The Markov chain in figure 6 (a type 5 model) is a more detailed model of fault/error handling. A distinction is made between a *fault* and an *error* [17]. A fault may occur at any time, but it need not cause an error right away. For example, the failure of a memory module is a fault, but an error does not occur until one of the computers tries to read from that memory module. Each system monitors itself periodically to check for faults and errors.

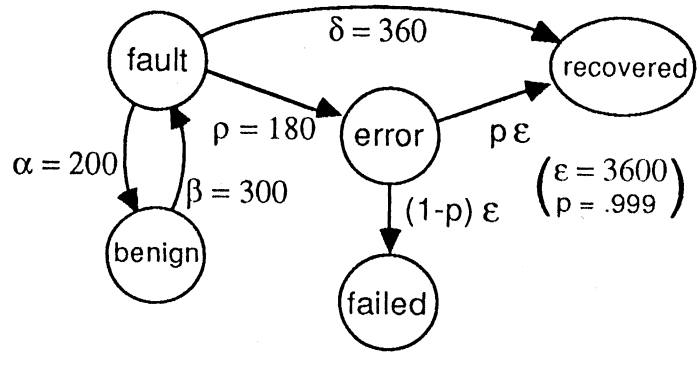

Fig. 6. Fault/Error-Handling Model.

In state *fault*, a fault has occurred in one of the computers. At this point, one of three things might happen. With rate δ, the system detects the fault and recovers, continuing to run with one less computer. With rate α, the fault (which is intermittent) becomes benign. While the system is in state *benign*, the fault will not cause an error. Eventually, the fault will become active again. With rate ρ, an error (owing to the fault) occurs. The error detection rate is ϵ. If the system can recover, the error is *covered,* and the system goes to state *recovered*; if not, the system goes to state *failed.* An error is covered with probability p.

The complete lower level model consists of two of these fault/error-handling chains strung together, one for handling a fault/error that occurs when 4 computers are operational, and one for when 3 computers are operational. A fault while two computers are operational causes immediate system failure.

To incorporate this lower-level Markov model into the fault tree we replace the gate *C3of4* and its four inputs by a basic event having the Cdf of the time to reach absorption in the Markov chain. When this two-level model is analyzed with $p = 0.999$, the probability of failure during an interval of 10 hours is computed to be 7.40502×10^{-6}. There is a noticeable increase in the likelihood of a failure, even with a high coverage value.

So far, we have assumed that a second (near-coincident) fault does not occur while a fault is being resolved. Suppose the occurrence of a near-coincident fault causes the system to fail. We could refine the model of figure 6 to include the possibility of a second fault while in states *fault* and *error.* Instead, we will expand to a 3-level model, with the bottom two levels expressing behavioral decomposition of the fault/error handling process [2, 8].

Behavioral decomposition is based on the premise that a long time passes between faults, but that the resolution of faults happens quickly. Based on this assumption, we model separately the fault/error-occurrence and fault/error-handling behavior of the system. Figure 7a shows the fault/error-handling model, which is the lowest-level model. The rate at which the second fault occurs is $k\lambda$, where k is the number of components that remain operational after the occurrence of the first fault.

Figure 7b shows the middle-level, fault/error-occurrence model (type 3) for this system. State k represents the system when k computers are operational. From state k, we go to state $k - 1$ if there is a covered failure and the number of operational computers has not fallen below 2. We go to state F if there is an uncovered failure or the number of operational computers falls below 2. The coverage value $c(k)$ (which is state-dependent) is the probability of reaching state *recovered* starting with state *fault* in the lower-level model of figure 7a with an appropriate value for k. The middle-level model replaces gate *C3of4* in the fault tree of figure 5. For the 3-level model, the probability of failure during 10 hours is computed to be 7.46150×10^{-6}. As anticipated this is a little higher than in the 2-level model.

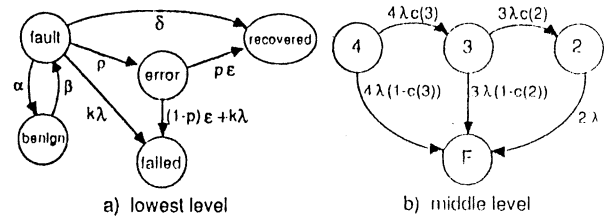

Fig. 7. Markov Chains for 3-level Model.

Because the coverage values $c(k)$ depend on k, SHARPE has to reanalyze the lowest-level model of figure 7a for each value of k. With a model of this size, the expense of doing that is negligible, but it is interesting to see how we can extend the model one further level (to 4 levels) to avoid some of that extra computation.

Let the bottom-most level consist of the single-fault/error-handling model of figure 6. At the next level up, we use the semi-Markov model (type 6) shown in figure 8. When a fault has occurred, two processes compete. The first is the fault/error-handling model of figure 6. This process completes when state *failed* or *recovered* is reached. If this process finishes first, we go to state *resolved* in figure 8. The second process is the occurrence of a second fault, happening at a rate of $k\lambda$ (a state-dependent rate). The fault is covered if the fault/error-handling process finishes first and if, within that process, the state *recovered* is reached rather than *failed.* Thus the coverage value is the probability of reaching state *resolved* in figure 8 multiplied by the probability of reaching state *recovered* in figure 6. We still have to analyze the model of figure 8 for each value of k, but we only have to analyze the single-fault system one time. The price for this savings is another level of approximation: in the 4-level model, a fault that occurs while another fault is in the *benign* state causes the system to fail, while in the 3-level model it does not. The probability of failure is 7.49546×10^{-6}, higher than that for the 3-level model.

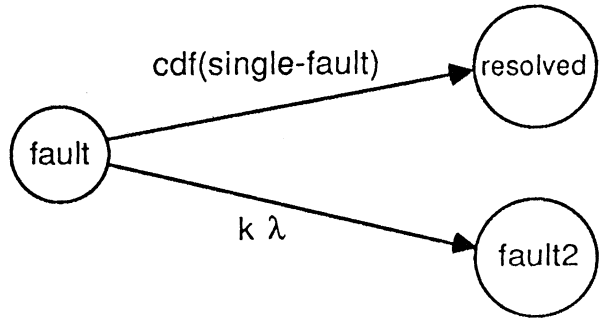

Fig. 8. Semi-Markov Chain in 4-level Model.

7. PERFORMANCE-RELIABILITY DEPENDENCE

When a program runs in a failure-prone environment, performance-related measures (such as response time) and

reliability measures (such as the probability of program completion) are *s*-dependent on each other. The task system in figure 9 (a type 7 model) is the producer-consumer model with two messages. Nodes P1 and P2 represent the tasks of producing messages 1 and 2, and nodes C1 and C2 represent the tasks of consuming those messages. There are two processors, one dedicated to producing messages and one to consuming them, each of which might fail. We want to compute the distribution of the finishing time of the task system and the probability that it finishes before a failed processor prevents further execution.

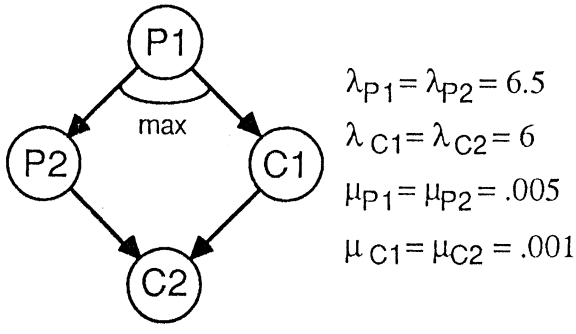

Fig. 9. Producer-Consumer Model.

If the time-to-completion of task i is exponentially distributed with parameter λ_i and the time-to-failure of the processor running the task is exponentially distributed with parameter μ_i, then the probability that task i finishes by time t is given by the defective distribution

$$F_i(t) = \left(\frac{\lambda_i}{\lambda_i + \mu_i}\right)(1 - \exp[-(\lambda_i + \mu_i)t] \qquad (4)$$

If each task i is assigned a function $F_i(t)$, then the overall time-to-completion distributed for the graph will be defective and its limit at infinity gives the probability that all of the tasks finish.

In the output file (table 6), lines 1 through 18 give the finish-time Cdf. Lines 4 through 6 tell us that the mass at zero of the Cdf is zero, the mass at infinity is 0.0018694, (this is the probability that a processor failure prevents the task system from completing), and the remaining mass (called the *continuous probability*) is 0.99813. This *continuous probability* is the probability that all tasks finish before their respective processors fail. Lines 8 through 18 give the Cdf for the task system and its mean and variance. The mean and variance are conditional on the finishing time being finite, because the actual mean and variance of a defective distribution are infinite and hence give no useful information.

TABLE 6
Results for Producer-Consumer Problem

```
 1
 2  CDF for producer-consumer:
 3
 4  probability at 0:              0.0000e+00
 5  probability at infinity:  1.8694e-03
 6  continuous probability:   9.9813e-01
 7
 8     -7.7309e+01 t(  1) exp(-6.0010e+00 t)
 9   +  7.7309e+01 t(  1) exp(-6.5050e+00 t)
10   +  9.9813e-01 t(  0) exp( 0.0000e+00 t)
11   + -9.9813e-01 t(  0) exp(-6.0010e+00 t)
12   + -9.9813e-01 t(  0) exp(-6.5050e+00 t)
13   +  9.9813e-01 t(  0) exp(-1.2506e+01 t)
14
15  mean and variance are conditional on finite time
16
17  mean:  5.6077e-01
18  variance:  8.3620e-02
```

```
19
20        system producer-consumer
21           t          F(t)
22
23  0.0000 e+00   0.0000 e+00
24  1.0000 e-01   6.8717 e-03
25  2.0000 e-01   6.1192 e-02
26  3.0000 e-01   1.7703 e-01
27  4.0000 e-01   3.2836 e-01
28  5.0000 e-01   4.8330 e-01
29  6.0000 e-01   6.2074 e-01
30  7.0000 e-01   7.3174 e-01
31  8.0000 e-01   8.1573 e-01
32  9.0000 e-01   8.7628 e-01
33  1.0000 e+00   9.1838 e-01
34
```

ACKNOWLEDGMENT

This work was supported in part by the US Air Force Office of Scientific Research under grant AFOSR-84-0132, by the US Army Research Office under contract DAAG29-84-0045, and by the US National Science Foundation under grant MCS-830200. UNIX is a registered trademark of AT&T Bell Laboratories. VMS is a trademark of Digital Equipment Corporation. PowerNode and UTX/32 are trademarks of Gould Inc.

REFERENCES

[1] K. S. Trivedi, *Probability and Statistics with Reliability, Queuing and Computer Science Applications,* Prentice-Hall, 1982.

[2] J. B. Dugan, K. S. Trivedi, M. Smotherman, R. Geist, "The Hybrid Automated Reliability Predictor", *AIAA Journal on Guidance Control and Dynamics,* 1986 May/Jun, pp 319-331.

[3] A. Goyal, W. C. Carter, E. de Sousa e Silva, S. S. Lavenberg, K. S. Trivedi, "The System AVailability Estimator", *Proc. IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-16),* 1986, pp 84-89.

[4] J. B. Dugan, K. S. Trivedi, R. M. Geist, V. F. Nicola, "Extended Stochastic Petri Nets: Applications and Analysis", in *PERFORMANCE '84* (E. Gelenbe, ed.), Elsevier Science Pub, 1985.

[5] A. Reibman, K. S. Trivedi, "Transient analysis of Markov dependability models", in review.

[6] A. Bobbio, K. S. Trivedi, "An aggregation technique for the transient analysis of stiff Markov systems", *IEEE Trans. Computers,* vol C-0, 1986 Sep, pp 803-814.

[7] J. J. Stiffler, L. A. Bryant, "CARE III Phase III Report-Mathematical Description", NASA Contractor Report 3566, 1982 Nov.

[8] K. S. Trivedi, R. Geist, M. Smotherman, J. B. Dugan, "Hybrid modeling of fault-tolerant systems", *Computers and Electrical Engineering, An International Journal,* vol 11, no. 2 and 3, 1985, pp 87-108.

[9] R. Sahner, K. S. Trivedi, "Performance and reliability analysis using directed acyclic graphs", *IEEE Trans. Software Engineering,* 1987 (forthcoming).

[10] B. Dodin, "Bounding the project completion time distribution in PERT networks", *Operations Research,* vol 33, no 4, 1985 Jul-Aug, pp 862-881.

[11] H. E. David, *Order Statistics,* John Wiley & Sons, 1981.

[12] B. Kernighan, D. Ritchie, *The C Programming Language,* Prentice-Hall, 1978.

[13] J. L. Fleming, "Relcomp: a computer program for calculating system reliability and MTBF", *IEEE Trans. Reliability,* vol R-20, 1971 Aug, pp 102-107.

[14] R. Sahner, K. S. Trivedi, "SHARPE User's Guide", 1986 Sept. Available from Computer Science Dept; Duke University; Durham North Carolina 27706 USA.

[15] D. P. Siewiorek, "Multiprocessors: reliability modeling and graceful degradation", in *Infotech State of the Art Conference on System Reliability,* London: Infotech International Ltd., 1977, pp 48-73.

[16] S. Bavuso, P. Peterson, D. Rose, "CARE III Model Overview and User's Guide," NASA Technical Memorandum Number 85810, 1984 Jun.

[17] J. Laprie, "Dependable computing and fault-tolerance: concepts and terminology", *Proc. IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-15)*, 1985 Jun, pp 2-7.

AUTHORS

Dr. Kishor S. Trivedi; Dept. of Computer Science; Duke University; Durham, North Carolina 27706 USA.

Kishor S. Trivedi: For biography see · p 189 in this issue.

Dr. Robin A. Sahner; Gould CSD; 1101 East University; Urbana, Illinois 61801 USA.

Robin A. Sahner is a Member of the Technical Staff at Gould Computer Systems Division, Urbana. She graduated in 1976 from the University of Hartford (mathematics and music), and received the AM degree in mathematics and the PhD degree in computer science from Duke University, Durham, NC in 1978 and 1986, respectively. Her work at Gould has included involvement in operating systems, network and interprocessor communications and secure systems. Her research interests include reliability and performance analysis and software engineering.

Manuscript TR86-202 received 1986 May 30; revised 1986 December 17.
IEEE Log Number 15255 ◄ TR ►

Workshop: R&M in Computer-Aided Engineering

Integrating Reliability & Maintainability into Computer-Aided Design & Engineering (CAD/CAE) Techniques

The IEEE Reliability Society, in cooperation with the IEEE Computer Society, is sponsoring a Workshop aimed at assuring that R&M concerns are addressed when the capabilities of engineering workstations are being defined. The workshop will bring together R&M engineers and workstation designers to exchange information on the current capabilities of CAE workstations and the additional capabilities needed to respond to R&M requirements.

Date	1987 August 25-26
Location	Xerox Training Center Leesburg, Virginia USA (Greater Washington DC area)
Further information:	Henry Hartt VITRO Corporation 14000 Georgia Avenue Silver Spring, Maryland 20906 USA *phone: 301-231-1431* ◄ TR ►