Modeling and analysis of computer system availability

by Ambuj Goyal
Stephen S. Lavenberg

The quantitative evaluation of computer-system availability is becoming increasingly important in the design and configuration of commercial computer systems. This paper deals with methods for constructing and solving large Markov-chain models of computer-system availability. A set of powerful high-level modeling constructs is discussed that can be used to represent the failure and repair behavior of the components that comprise a system, including important component interactions, and the repair actions that are taken when components fail. If time-independent failure and repair rates are assumed, then a time-homogeneous continuous-time Markov chain can be constructed automatically from the modeling constructs used to describe the system. Markov chains having tens of thousands of states can be readily constructed in this manner. Therefore, techniques that are particularly suitable for numerically solving such large Markov chains are also discussed, including techniques for computing the sensitivities of availability measures with respect to model parameters. A computer system modeling example is presented to illustrate the use of these modeling and analysis techniques. The modeling constructs, automatic Markov-chain construction, and model-solution methods have been implemented in a program package called the System Availability Estimator (SAVE).

©**Copyright** 1987 by International Business Machines Corporation. Copying in printed form for private use is permitted without payment of royalty provided that (1) each reproduction is done without alteration and (2) the *Journal* reference and IBM copyright notice are included on the first page. The title and abstract, but no other portions, of this paper may be copied or distributed royalty free without further permission by computer-based and other information-service systems. Permission to *republish* any other portion of this paper must be obtained from the Editor.

1. Introduction

System availability is becoming an increasingly important factor in evaluating the behavior of commercial computer systems. This is due to the increased dependence of enterprises on continuously operating computer systems and to the emphasis on fault-tolerant designs. Thus we expect quantitative evaluation of availability to be of increasing interest to computer-system manufacturers. Most of the modeling work in fault-tolerant computing has focused on models of mission-oriented systems with high reliability requirements such as space computers, avionics systems, wind-tunnel systems, and ballistic missile defense computers [1, 2]. For the mission to succeed, the system must not fail during the mission time. In addition, the repair or replacement of failed components is usually not possible during the mission. Hence, the probability that the system does not fail during the mission time, i.e., the system reliability, is a measure of interest. The modeling of continuously operating systems with high availability requirements such as telephone switching systems, general-purpose computer systems, transaction processing systems (e.g., airline reservation systems), and communication-network computers has received less attention and is the focus of this paper. For systems like these, system failures can be tolerated if they occur infrequently and result in short system downtimes. Failed components can be repaired or replaced either during operation or when the system is down. For such systems, the expected fraction of time the system is operational is an important availability measure.

From an availability-modeling point of view, a system consists of a collection of hardware and software components, each of which may be subject to failure, recovery, and repair. (Software components in operation can be modeled with constant failure rates, as described in [3].) As we discuss later, component interactions often have a substantial effect on system availability and must therefore be considered in addition to the individual component behaviors. A system is considered to be operational (available) if specified combinations of its components are operational. Different types of mathematical models are used to predict system availability and reliability measures, including combinatorial models, continuous-time Markov chain models, and semi-Markov process models [4]. In this paper we focus on continuous-time Markov chain models, which are perhaps the most commonly used. The state-space size of such models grows (often exponentially) with the number of components being modeled. Therefore, except for models that represent a very small number of components and hence have a small state-space size, the direct construction of the Markov chain is a tedious and error-prone process. Rather than requiring the modeler to construct the Markov chain directly, it would be better to provide a high-level modeling language containing constructs which aid in representing the failure, recovery, and repair behavior of components in the system, as well as important component interactions. The challenge in developing such a language lies in making it comprehensive enough so that important system details can be modeled, but simple enough so that the Markov chain can be automatically constructed from the modeling-language description.

In Section 2 we describe such a modeling language and illustrate its use with an example. The language contains a few simple but powerful constructs that are capable of representing both independent component behaviors and component interactions. The constructs provide a framework for the modeler to use in thinking about aspects of the system to be modeled, particularly the component interactions. The language has been incorporated in a system-availability-modeling program package called the System Availability Estimator (SAVE) [5]. SAVE automatically generates the Markov chain from the modeling language description. Other modeling packages (e.g., HARP [6]) provide simple fault-tree (or reliability-block-diagram)-oriented languages to generate Markov chains, but typically only recovery-based constructs are provided to model component interactions or dependencies.

Once the Markov chain has been generated, it must be solved numerically in order to compute system availability measures. The SAVE modeling language and automatic Markov chain construction allow Markov chains to be generated that have large state spaces, e.g., tens of thousands of states. Therefore SAVE incorporates numerical solution methods that are particularly suitable for large-state-space Markov chains. In Section 3 we first define several availability and reliability measures of interest and then briefly describe the solution methods that are implemented in SAVE to compute these measures. We also discuss solution methods for computing the sensitivities (derivatives) of some of these measures with respect to model input parameters (e.g., failure and repair rates and coverage [7]). Such sensitivities are very useful in suggesting design improvements for the system being modeled, as we demonstrate. More detailed discussions about solution methods are found in [4, 8–10].

In Section 4, we return to the example of Section 2 to further illustrate the features of the SAVE modeling language and to demonstrate the capabilities of the SAVE solution methods and the use of sensitivity analysis in design and reconfiguration of fault-tolerant systems. Section 5 contains concluding remarks and comments on future research directions.

2. An availability modeling language

Some modeling considerations

The uses of system availability models include comparing various fault-tolerant design alternatives and, for a particular design, identifying any availability bottlenecks that may require subsequent design improvements. A system availability model is an abstraction of the system in which we ignore certain design details to reduce the size of the model but include other details to be able to study the various design trade-offs. As mentioned in Section 1, for availability-modeling purposes a system is considered to be a collection of hardware and software components. Thus components are the basic entities that are represented in a model. The modeling language we present allows the modeler to describe for each component its failure and repair behavior, including its interactions with other components. In addition, the language allows the modeler to describe those subsets of the components that must be operational (nonoperational) for the system to be considered operational (nonoperational). The modeler must decide the level of details of the hardware and software to include for a component representation in an availability model. For example, in modeling a computer system each processor could be considered a component, or, at a lower level, each of the logic modules that comprise a processor could be considered a component. In the latter case there would be many more components and the model would have a much larger state space. The level of detail chosen should depend on the questions to be addressed using the model and/or on the obtainable failure and repair data, as well as on the resulting model size. Thus, if one wanted to determine the effect of a standby processor on system availability, it might be better to consider processors rather than the logic modules that comprise them to be components. However, if failure data were readily obtainable at the logic-module level, these data could be combined, using a combinatorial model, to estimate the processor failure rate. This is an example of hierarchical modeling, where lower-level data are input to a lower-level model which is solved to estimate the inputs to a higher-level model. In the rest of the paper we assume that the required inputs are available for the modeling level that is chosen.

Using the terminology developed in [11], a hardware fault may be transient, intermittent, or permanent. (Software faults are permanent.) A fault is not detected until the hardware component (or the section of the code) is exercised and causes an error which may be detected and recovered from or may lead to a component or system failure. We restrict ourselves to modeling at the error or failure level because error and failure data are much easier to gather than fault data. If the system implements some error-recovery techniques, modeling at the error level may be appropriate; otherwise modeling is at the failure level.

In the remainder of this section we describe the main constructs of the availability-modeling language which has been implemented in SAVE. (For completeness the entire syntax of the language is given in Appendix A.) We use the simple computer-system example shown in **Figure 1** to aid in this description. The system comprises a total of nine components consisting of three software components [operating system software (Mvs1), communication subsystem software (Vtam1), and database subsystem software (Ims1)]; three hardware components [processor (Proc1), power supply (Power1), and processor controller (Pc1)]; two components that are combinations of hardware and software [front-end network (Network) and storage subsystem (Storage)]; and one data component [database subsystem (Database)]. The constructs contained in the language are general enough to be able to describe the above types of components and their interactions. We also assume that there are three classes of repairmen available to repair the above system, namely Fieldengineer, Operator, and Softretry. The first does hardware repair, the second does software restart, and the third does automatic recovery. Note that the software restart and the automatic recovery are also considered a type of repair on a component. This leads to uniformity in language constructs, as explained later in this section.

We first present the constructs used to describe the failure and repair behavior of components, including the component interactions. Next, we present the constructs used to describe the actions taken by the various classes of repairmen, including the repair strategies followed. Finally, we present the constructs used to describe the conditions under which the system is considered to be operational (or nonoperational).

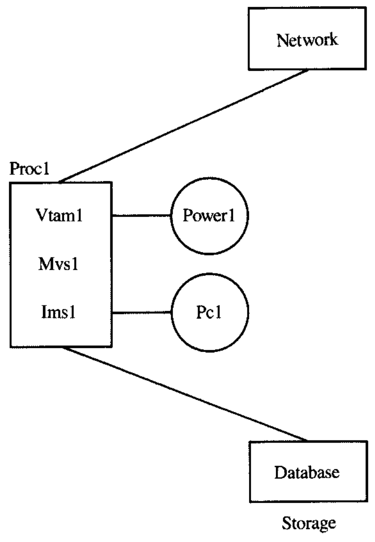

Figure 1

A computer-system example.

- *Failure and repair behavior of components*

The most important construct in the modeling language is the COMPONENT construct, which is used to model the failure and repair behavior of a component, including its interactions with other components. In the modeling language we use the term "failure" to refer to either an error or a failure, and we use the term "repair" to refer to any action that renders a "failed" component operational. Many types of repair are possible, including automatic recovery, operator restart of a subsystem or the entire system, and physical repair or replacement. In the modeling language a component is assumed to fail in possibly multiple modes, and a different type of repair can be associated with each mode. Also, each type of repair can be performed by a different class of repairman and can have a different repair rate. (The repair strategies followed by a repairman in a class are discussed in the following subsection.) Therefore, a component can be either in the operational state or in one of the many failed states depending upon the mode of failure, as shown in **Figure 2**. The transition rates from operational to failed and failed to operational states are affected by component interactions and the repair strategies followed by the repairmen. A distinct combination of the state of each

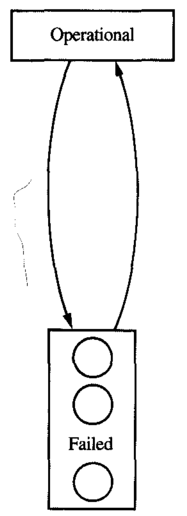

Figure 2

States of a single component.

component (Figure 2) determines the total system state. The state of each component is discussed in greater detail later in this section.

We first illustrate a simple use of the COMPONENT construct to model different types of repair using the example in Figure 1. Suppose that Proc1 has two failure modes, one in which software retry is successful and one in which physical repair is needed. This can be modeled as follows using the COMPONENT construct:

```
COMPONENT:                      Proc1
  FAILURE RATE:                 procfr
  FAILURE MODE PROBABILITIES:   p1, 1 − p1
  REPAIR RATE:                  procrr1, procrr2
  REPAIRMAN CLASS USED:         Softretry, Fieldengineer
```

The convention used in this paper is that all keywords are capitalized; all names, e.g., component names and repairman class names, have the first letter capitalized; and all parameters (variables) and constants are lowercase words. The constructs to declare parameters and constants are given in Appendix A. In the above example, we gave the component a name, defined its failure rate, failure-mode probabilities, and repair rates in each mode, and specified which entity (repairman class) does the repair in each mode. These repairman classes are further defined in the following subsection. The failure and repair rates are assumed to be time-independent. If there are multiple components that are identical in their failure and repair behaviors, e.g., multiple identical processors, this is expressed by appending the number of such components in parentheses after the component name, as shown in Appendix A. In general, then, the COMPONENT construct describes not just a single component instance but a single component type, where components of the same type are identical as far as their failure and repair behaviors are concerned.

In real systems the failure behavior of a component can be more complicated than we have described so far. Operational dependencies, failure/error propagation, and differences among dormant, spare, and operational states of a component affect the failure process. To capture such complex failure behavior of a component, we add two more states, namely spare and dormant, in our component model, as shown in **Figure 3**. Now, a component can be in one of four states: operational, failed (could be in one of many models), spare, and dormant. In the latter three states the component is considered nonoperational. A component is said to be dormant when it is not operating because its operation depends upon some other components which are nonoperational or because the whole system is nonoperational. For example, a processor is dormant when its power supply has failed. The component may fail when it is either operational, spare, or dormant, and its failure rate may be different in the three states. Thus we distinguish among operational, spare, and dormant failure rates of a component. A component may also fail if it is affected by the failure of other components in the system (failure propagation). In addition to the operation of a component depending on the operation of other components, the repair of a component could depend on the operation of other components. For example, in systems which have a service processor, the service processor must be operational in order for a failed processor to be repaired. We now illustrate, again using the example in Figure 1, how this more complicated failure and repair behavior can be modeled using the expanded COMPONENT construct shown below:

```
COMPONENT:                      Proc1
  OPERATION DEPENDS UPON:       Power1
  REPAIR DEPENDS UPON:          Power1, Pc1
  DORMANT WHEN SYSTEM DOWN:     NO
  DORMANT FAILURE RATE:         procdfr
  FAILURE RATE:                 procfr
  FAILURE MODE PROBABILITIES:   p1, 1 − p1
  REPAIR RATE:                  procrr1, procrr2
  REPAIRMAN CLASS USED:         Softretry, Fieldengineer
  COMPONENTS AFFECTED:          NONE, Database
    Database: 1 − coverage
```

In Figure 1, suppose that when Proc1 fails it could with some probability cause Database to fail by contaminating the data. This might happen in one or more of the failure modes of Proc1. This behavior is expressed using COMPONENTS AFFECTED where components that can be affected (caused to fail) in each failure mode are specified. Moreover, for each component that can be affected, the probability that it is actually affected is also specified. NONE is a keyword indicating that no components are affected in a particular failure mode. Thus no components are affected in the first failure mode of Proc1. Database can be affected in the second failure mode of Proc1 and there is a probability of 1 − coverage of doing so. (In general, in each mode more than one component can be affected; this can be specified by using the LISTS construct shown in Appendix A, where each component in a list can be affected with a different probability.) Database, thus affected (failed), must be repaired before becoming operational. This is in contrast to the operational dependency of Proc1 on Power1 expressed using OPERATION DEPENDS UPON in the example. When Power1 fails, Proc1 becomes dormant rather than failed, so that no repair on Proc1 is needed. When Proc1 is dormant, it can still fail, with a possibly different failure rate than when it is operational. This is expressed using DORMANT FAILURE RATE as shown. Suppose that Proc1 is not dormant when the system is nonoperational (see the subsection on availability specification for the definition of when a system is or is not operational). This is expressed using DORMANT WHEN SYSTEM DOWN and the keyword NO. Suppose that the repair of Proc1 depends upon both Power1 and Pc1 (processor controller) being operational. This is expressed using REPAIR DEPENDS UPON as shown.

The complete syntax of the COMPONENT construct, which is given in Appendix A, has not been fully illustrated using the simple example of Figure 1. If there are spares for a given component, the number of spares and their failure rate can be specified using SPARES and SPARES FAILURE RATE, respectively. The failure rate of a component may also be dependent upon the number of that type of component that are operational. For example, in a four-processor system, if one processor fails, the load of four processors must be handled by only three, which may cause the failure rate of each individual processor to increase. This can be expressed using the general form of FAILURE RATE that is shown in Appendix A, in which a different failure-rate expression can be given for each possible number of operational components of the type being described.

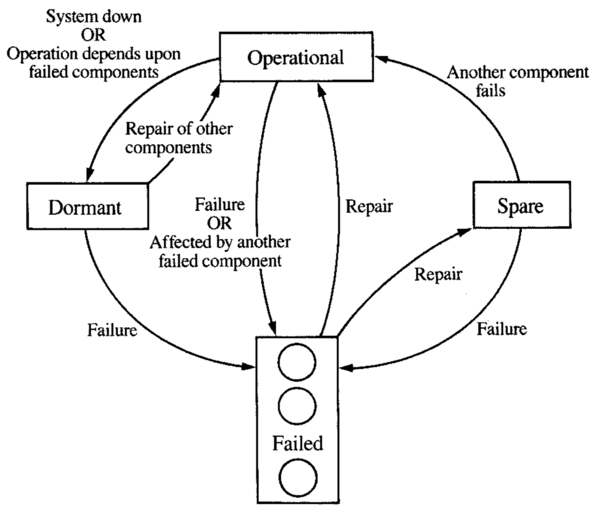
System down
OR
Operation depends upon failed components
Operational
Another component fails
Repair of other components
Dormant
Failure
OR
Affected by another failed component
Repair
Spare
Repair
Failure
Failure
Failed

Figure 3

All states of a single component.

• *Actions of repairmen*

We next discuss the REPAIRMAN CLASS construct that is used to provide information about the actions taken by the repairmen in each repairman class. As was mentioned earlier in this section, different types of repair, e.g., automatic recovery, operator restart, and physical repair or replacement, can be associated with the different failure modes of a component. Each type of repair can be performed by a different repairman class. A repairman class could represent operators who do restart, field engineers who do physical repair or replacement of the parts of the system, and either hardware or software that does automatic recovery. A repairman class is specified by assigning it a name, giving the number of repairmen in the class and the repair strategy used, and, if the repair strategy is based on priorities, assigning a priority to each component type that can be repaired by the repairman class. In some cases the number of repairmen in a class is effectively unlimited, in that queueing for repair does not occur. For example, this can be considered to be the case for software retry. The use of the REPAIRMAN CLASS construct is illustrated below for the example in Figure 1:

```
REPAIRMAN CLASS:        Softretry(UNLIMITED)
REPAIRMAN CLASS:        Fieldengineer(1)
  REPAIR STRATEGY:      PRIORITY
    Storage:            1
    Network:            1
    Proc1:              2
    Power1:             3
    Pc1:                4
```

Earlier we specified, using the COMPONENT construct, that Proc1 has two failure modes, one where software retry is successful and one where physical repair is required, and that a different repairman class, named Softretry and Fieldengineer, respectively, was used in each mode. The keyword UNLIMITED specifies that the Softretry repairman class has an effectively unlimited number of repairmen, and

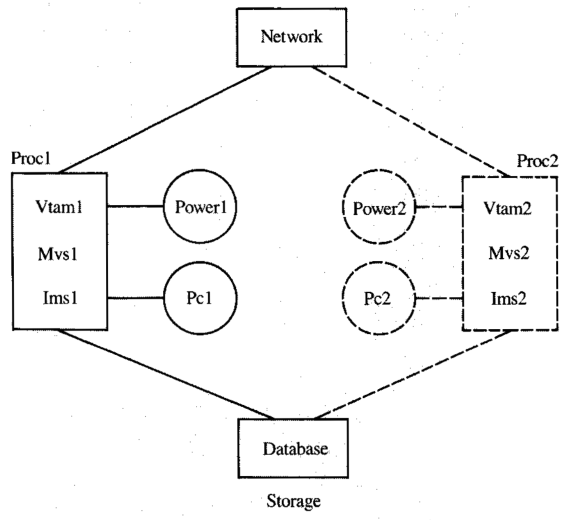

Figure 4

Duplexed-processor example.

hence queueing for this class cannot occur. In this case it is not necessary to specify a repair strategy. Otherwise the number of repairmen for the class is given in parentheses, and a repair strategy must be specified. In the example it is assumed that there is a single field engineer who can repair/replace any component that fails and that the repair strategy used when there is more than one component to be repaired is to give highest priority to Storage and Network, second highest to Proc1, and so on. Components that have equal priority are assumed to have equal probabilities of being chosen for repair. The priority repair strategy is assumed to be of the preemptive-resume type. This means that if a component fails while a component of equal or lower priority is being repaired, then the current repair is preempted and a new decision is made to repair a component based on the specified repair strategy [ROS (Random Order Service) or PRIORITY]. Eventually repair is resumed on the preempted component as determined by its priority. If all component types are assumed to have equal priorities, then the repair strategy can be specified as ROS and the priority specification omitted. With a preemptive-resume-type repair strategy and with the assumption of time-independent failure and repair rates, the state of the Markov chain that results can be represented by a concatenation of the states (operational, failed, spare, or dormant) of each component. While other types of repair strategies such as nonpreemptive priority and first-come-first-served could be defined, they would result in Markov chains having much larger state spaces than with preemptive resume. Therefore, the only repair strategies implemented in the SAVE package and the only ones we consider are PRIORITY and ROS.

The COMPONENT and REPAIRMAN CLASS constructs have been chosen to allow relatively easy description of system aspects that are important to incorporate in an availability model. This is illustrated further by the example in Section 4. They have also been chosen with the aim of allowing straightforward generation of the resulting Markov chain and limiting its state-space size.

• *Availability specification*

A system is considered to be available if specified subsets of its components are operational. This can be expressed using a reliability block diagram or, equivalently, a Boolean expression over the component names involving "and" and "or." Alternatively, it may be easier to specify conditions under which the system is unavailable by specifying subsets of its components that are nonoperational. This can be done using a fault tree or equivalent Boolean expression. Consider the example in Figure 1 where all components (except Pc1) need to be operational for the system to be considered operational. These conditions can be expressed using the EVALUATION CRITERIA construct shown below. The keyword BLOCKDIAGRAM indicates that system availability conditions will be specified and the keyword FAULTTREE indicates that system unavailability conditions will be specified. Both are illustrated and are logically equivalent:

```
EVALUATION CRITERIA: BLOCKDIAGRAM
  Exp1: Database and Storage and Network
  Exp2: Power1 and Proc1 and Mvs1 and Vtam1 and Ims1
  Exp1 and Exp2
EVALUATION CRITERIA: FAULTTREE
  Exp1: Database or Storage or Network
  Exp2: Power1 or Proc1 or Mvs1 or Vtam1 or Ims1
  Exp1 or Exp2
```

The final expression for availability specification (BLOCKDIAGRAM) or unavailability specification (FAULTTREE) can contain previously declared subexpressions. Note that we have omitted Pc1 from the specifications because it is only needed for repair and is not required for the system to be considered operational. We could also have omitted Power1 from the specifications because its effect has already been taken into account using OPERATION DEPENDS UPON while defining the Proc1 component. Therefore, its exclusion in the specifications would not change any state from operational to nonoperational or vice versa. If in Figure 1 we duplex the processor and its associated hardware and software as shown in **Figure 4**, the following BLOCKDIAGRAM expression could be used:

```
EVALUATION CRITERIA: BLOCKDIAGRAM
  Exp1: Database and Storage and Network
  Exp2: Power1 and Proc1 and Mvs1 and Vtam1 and Ims1
  Exp3: Power2 and Proc2 and Mvs2 and Vtam2 and Ims2
  Exp1 and (Exp2 or Exp3)
```

Clearly connectivity or lack of connectivity among components can affect the above Boolean specification and hence the system availability measures.

On the basis of the availability specification of the system, the state space of the Markov chain, which is determined by the failure and repair behavior descriptions of components and the actions of the repairmen classes, can be partitioned into an available set and an unavailable set. System availability measures can then be computed, as discussed in Section 3.

3. Computation of availability and reliability measures

In this section we review numerical methods that are suitable for computing system availability measures for large irreducible time-homogeneous continuous-time Markov chains of the type that can be generated using the modeling language described in Section 2. Such Markov chains typically have a sparse transition-rate matrix and, since repair rates are typically orders of magnitude larger than failure rates, they typically have order-of-magnitude differences in transition rates. We show that the numerical methods we discuss are also suitable for computing mean time to system failure and system reliability, so that we also consider these measures. The methods we describe have been implemented in the SAVE package and are discussed in more detail elsewhere [4, 8–10]. We include this review for completeness.

The state space of the Markov chain is determined from the failure and repair descriptions of the components and the actions of the repairman classes. On the basis of the availability specification of the system, the state space can be partitioned into a set of states for which the system is available and a set for which the system is unavailable. (This is done automatically in SAVE.) Let $A(t)$ denote the fraction of time the Markov chain is in the available set during the interval $[0, t]$. The availability measures we wish to compute are the interval availability

$$I(t) = E[A(t)],$$

the steady-state availability

$$A = \lim_{t\to\infty} I(t),$$

and the distribution of availability

$$F(t, x) = Pr[A(t) \le x].$$

Steady-state availability is probably the most commonly used availability measure. However, if failures occur very infrequently, steady state may not be reached during a finite time interval of interest. In that case, interval availability would be a more meaningful measure than steady-state availability. The distribution of availability is of interest when a vendor offers a computer system with a guaranteed level of availability, e.g., 0.95 or higher. If the guaranteed level is not met over a specified time interval, e.g., three months, the vendor pays a penalty. The distribution of availability can be used to compute the probability of not meeting the guaranteed level over the specified interval.

In system design studies it is often useful to be able to identify the model input parameters, e.g., the failure rates or repair rates to which an availability measure is most sensitive in the sense that a small fractional change in the parameter value would cause a large change in the availability measure. We define the sensitivity of an availability measure with respect to a model input parameter to be the derivative of the measure with respect to the parameter (assuming the derivative exists) normalized by multiplying by the value of the parameter. The normalization aids in comparing the absolute values of the sensitivities with respect to different parameters. A negative (positive) sensitivity value implies that a small increase in the parameter value will decrease (increase) the availability measure. A higher absolute value of the sensitivity with respect to a parameter value implies that a fractional change in the parameter value has more effect on the measure. In Section 4 we illustrate the use of sensitivities in design analysis.

We next show that the steady-state availability, its sensitivity with respect to a model input parameter, the mean time to system failure, and its sensitivity with respect to a model input parameter can be computed by solving four different sets of simultaneous linear equations. The stationary probabilities of an n-state (time-homogeneous) Markov chain satisfy the homogeneous system of n linear equations in n unknowns,

$$\boldsymbol{\pi}\mathbf{Q} = \mathbf{0}, \qquad \boldsymbol{\pi}\mathbf{e} = 1, \tag{1}$$

where $\mathbf{Q}$ is the transition rate matrix of the Markov chain, $\mathbf{e}$ is a column vector of all ones, and the row vector $\boldsymbol{\pi}$ is the stationary probability vector. Since the Markov chain is irreducible, $\boldsymbol{\pi}$ exists and is unique. Assume that the available states of the Markov chain are numbered from 1 to n'. Then the steady-state availability is

$$A = \sum_{i=1}^{n'} \pi_i . \tag{2}$$

To compute the derivative of A with respect to a model input parameter, say λ, we compute the derivative of the vector $\boldsymbol{\pi}$ with respect to λ and use (2). Differentiating (1) with respect to λ yields a nonhomogeneous system of n linear equations in n unknowns,

$$\frac{d\boldsymbol{\pi}}{d\lambda}\mathbf{Q} = -\boldsymbol{\pi}\frac{d\mathbf{Q}}{d\lambda}, \qquad \frac{d\boldsymbol{\pi}}{d\lambda}\mathbf{e} = 0, \tag{3}$$
$$\text{i.e., } \mathbf{x}\mathbf{Q} = \mathbf{b}, \qquad \mathbf{x}\mathbf{e} = 0,$$

where **b** is known after $\boldsymbol{\pi}$ is computed, and **x** is the vector of derivatives that is to be computed.

The mean time to failure of the system is the mean time until the Markov chain first exits the available set of states. It is obtained from the transient behavior of a modified Markov chain in which all unavailable states are replaced by a single absorbing state. This transient behavior is described by the set of linear differential equations

$$\frac{d\tilde{\boldsymbol{\pi}}(t)}{dt} = \tilde{\boldsymbol{\pi}}(t)\tilde{\mathbf{Q}}, \tag{4}$$

where $\tilde{\mathbf{Q}}$ is the $n' \times n'$ upper left submatrix of $\mathbf{Q}$ corresponding to the available states, and $\tilde{\boldsymbol{\pi}}(t)$ is the vector of state probabilities at time t for the available states in the modified Markov chain. All the rows of $\tilde{\mathbf{Q}}$ no longer sum to 0, as there is a finite transition rate from some of the available states to the unavailable states. Integrating (4) from 0 to infinity, we have

$$\tilde{\boldsymbol{\pi}}_\infty - \tilde{\boldsymbol{\pi}}_0 = \left[\int_0^\infty \tilde{\boldsymbol{\pi}}(t)dt\right]\tilde{\mathbf{Q}},$$

where $\tilde{\boldsymbol{\pi}}_0$ is the vector of initial state probabilities and $\tilde{\boldsymbol{\pi}}_\infty$ is the vector of final state probabilities. Since the probability of exiting the available set in finite time is 1, $\tilde{\boldsymbol{\pi}}_\infty = 0$. The components of the vector $\int_0^\infty \tilde{\boldsymbol{\pi}}(t)dt$ are the mean times spent in each available state before exiting the available set. Denoting this vector by **z**, we get the set of n' nonhomogeneous linear equations in n' unknowns

$$\mathbf{z}\tilde{\mathbf{Q}} = -\tilde{\boldsymbol{\pi}}_0. \tag{5}$$

The sum of the elements of **z** gives the mean time to failure. To compute the derivative of the mean time to failure with respect to a model input parameter, say λ, we compute the derivative of the vector **z** with respect to λ. Differentiating (5) with respect to λ yields a nonhomogeneous system of n' linear equations in n' unknowns,

$$\frac{d\mathbf{z}}{d\lambda}\tilde{\mathbf{Q}} = -\mathbf{z}\frac{d\tilde{\mathbf{Q}}}{d\lambda}, \quad \text{i.e., } \mathbf{y}\tilde{\mathbf{Q}} = \mathbf{c}, \tag{6}$$

where **c** is known after **z** is computed, and **y** is the vector of derivatives that is to be computed. The sum of the elements of **y** gives the derivative of the mean time to failure with respect to λ.

Numerical methods for computing the solutions of (1), (3), (5), and (6) must be suitable for Markov chains with large state spaces. Since **Q** is sparse, the state-space size problem can be alleviated to some extent by using sparse-matrix storage techniques. Iterative-solution methods are particularly suitable for use with sparse-storage techniques, since the iteration matrix is not altered on successive iterations. Direct methods, on the other hand, need much more sophisticated sparse-storage schemes, since allowance must be made for fill-in (zero elements which become nonzero as a result of operations upon the matrix) as well as for the elimination of nonzero elements. In some cases the fill-in can become excessive, which is hard to predict *a priori*. In addition, with iterative methods, advantage can be taken of good initial approximations, especially when a series of related models are being solved. Also, an iterative procedure can be halted once a prespecified tolerance criterion has been satisfied (e.g., a user may need three-decimal-place accuracy), whereas direct methods, by definition, perform a fixed amount of computation and yield the best accuracy they can. Finally, iterative methods do not suffer from problems of stability, for successive iterations always refer to the iteration matrix, which is not altered. For these reasons, we prefer iterative methods for solving large Markov-chain availability models. The method implemented in SAVE for solving (1), (3), (5), and (6) is successive overrelaxation (SOR). Other candidate methods are empirically compared with SOR in [10] and are shown to be inferior. Implementation issues for the SOR method are also considered in [10], including the choice of an appropriate relaxation parameter. It is also shown that the structure of the transition-rate matrix and the order-of-magnitude differences in the transition rates can be exploited to speed up convergence of the SOR method.

Next we consider the computation of interval availability and reliability, both of which are time-dependent measures which are computed in SAVE by a technique called randomization or uniformization. The interval availability during $[0, t]$ can be expressed as

$$I(t) = \sum_{i=1}^{n'} \frac{1}{t}\int_0^t \pi_i(u)du, \tag{7}$$

where $\boldsymbol{\pi}(t)$, the state probability vector at time t, satisfies the linear differential equations

$$\frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t)\mathbf{Q}, \qquad \boldsymbol{\pi}(t)\mathbf{e} = 1. \tag{8}$$

The reliability of the system at time t is given by

$$R(t) = \sum_{i=1}^{n'} \tilde{\pi}_i(t), \tag{9}$$

where $\tilde{\boldsymbol{\pi}}(t)$ satisfies the linear differential equations in (4). Randomization is an iterative method for computing time-dependent measures for a Markov chain. The mathematical basis for this technique was developed in [12]. A recent recommended reference is [13]. Randomization is particularly suitable for Markov chains with large state spaces, as has been demonstrated in [13, 14]. As shown in [13], $\boldsymbol{\pi}(t)$ can be computed using the expression

$$\boldsymbol{\pi}(t) = \sum_{k=0}^{\infty} \boldsymbol{\pi}(0)(\mathbf{Q}^*)^k \frac{e^{-qt}(qt)^k}{k!}, \tag{10}$$

where

$$\mathbf{Q}^* = \mathbf{I} + \mathbf{Q}/q, \tag{11}$$

and where

$q \geq \max(-q_{ii})$ and $\mathbf{I}$ is the identity matrix. The interval availability can be computed by symbolically integrating (10) and summing the first n' components of the resulting vector, which results in

$$I(t) = \sum_{i=1}^{n'} \frac{1}{qt} \sum_{k=0}^{\infty} \pi(0)(\mathbf{Q}^*)^k \sum_{m=k+1}^{\infty} \frac{e^{-qt}(qt)^m}{m!}. \tag{12}$$

We can also compute $\tilde{\pi}(t)$ using the expression in (10) with $\mathbf{Q}^*$ replaced by $\tilde{\mathbf{Q}}^*$, where $\tilde{\mathbf{Q}}^*$ is given in (11) with $\mathbf{Q}$ replaced by $\tilde{\mathbf{Q}}$. The reliability is computed from $\tilde{\pi}(t)$ using (9). There are several advantages in using randomization. Numerical problems are minimized, since all terms in (10) and (12) are nonnegative. It is easy to show that the errors obtained in the interval availability and the reliability obtained by approximating the infinite sum over k by the sum of the first K terms are upper-bounded by

$$\sum_{k=K+1}^{\infty} \frac{e^{-qt}(qt)^k}{k!}.$$

Thus, it is possible to achieve specified error tolerances. Other advantages include ease of exploiting sparse storage techniques and ease of implementation. The derivatives of reliability with respect to transition-rate parameters can also be computed using uniformization [15].

The numerical methods of computing the distribution of availability for Markov-chain models are fairly recent. One method implemented in SAVE is based on evaluating the joint probability that the cumulative operational time during an interval of length t is x and that the system is in a particular operational or nonoperational state at time t. Equations are obtained relating these joint probabilities for arguments t and x to those for arguments $t - \Delta$ and x and those for arguments $t - \Delta$ and $x - \Delta$, where Δ is chosen small enough so that the probability of more than one event in the Markov process in time Δ is negligible. These equations allow recursive computation of probability density functions of system availability which can be used to compute the distribution of availability by performing an appropriate integration step. The details are given in [9]. An alternative method implemented in SAVE for computing the distribution of availability is based on the randomization method [8]. The advantage of this method is that the global errors can be bounded, and it requires less computation time than the numerical method described in [9]. The disadvantage is that it requires a larger amount of storage than the numerical method.

4. Example continued

In this section we continue the discussion of the example of Section 2 and show that other types of components besides hardware (i.e., Proc1) can be modeled by the language constructs discussed in that section. Next, we assign numerical values for the parameters of the model and solve the model using SAVE. We then show how sensitivity analysis can be used to improve the design of the system.

Table 1 Parameter values for the model of Appendix B.

Component	*Mean time to failure* (h)	*Mean time to repair* (h)
Proc	960	1/10, 2
Power	9600	1
Pc	9600	1
Network	12000	1
Storage	12000	1
Mvs	7200	1/4
Vtam	7200	1/12
Ims	7200	1/12
Database	—	1/4

Returning to Figure 1, we select Mvs1 (a software component) for this discussion. (A complete SAVE language description for the example of Figure 1 is given in Appendix B.) When a software component fails, the operator typically takes a system dump and restarts the software. In such cases, the restart time is the only time considered in the repair time of software components. Another important point to note is that when Mvs1 fails, the software running on top of Mvs1 (i.e., Vtam1 and Ims1) must also be restarted after Mvs1 has been restarted. Therefore, whenever Mvs1 fails, it affects (or fails) components Vtam1 and Ims1 with probability one. If a component affects more than one component in any given failure mode, these affected components must be declared as part of a list using the LISTS construct given below:

```
LISTS: Cmplist2
 Cmplist2: Vtam1, Ims1
*
COMPONENT                        Mvs1
 OPERATION DEPENDS UPON:         Proc1, Power1
 REPAIR DEPENDS UPON:            Proc1, Power1
 DORMANT WHEN SYSTEM DOWN:       NO
 FAILURE RATE:                   mvsfr
 REPAIR RATE:                    mvsrr
 REPAIRMAN CLASS USED:           Operator
 COMPONENTS AFFECTED:            Cmplist2
  Cmplist2: 1, 1
```

Another interesting type of component is Database. This is a data component which does not fail by itself (i.e., it has a zero failure rate). It fails only if some other component affects (or contaminates) it. Therefore, in the Database component description no FAILURE RATE construct has been used, as shown in Appendix B. SAVE assumes natural default values for the omitted constructs. We solved the model of Appendix B using SAVE for the parameter values shown in **Table 1**.

The values for p1 and coverage were both 0.8. We also solved a similar model for the example of Figure 4, where the processor, power supply, processor controller, and all the

Table 2 Sensitivity of steady-state availability with respect to parameter values.

Component		*Simplex* ($A = 0.9990$) ($1 - A = 0.0010$)	*Duplex* ($A = 0.9998$) ($1 - A = 0.0002$)	*Duplex + storage* ($A = 0.99992$) ($1 - A = 0.00008$)
Proc	(f)	0.6×10^{-3}	0.2×10^{-4}	0.5×10^{-6}
Power	(b)	0.1×10^{-3}	0.4×10^{-6}	0.4×10^{-6}
Pc	(r)	0.3×10^{-6}	0.7×10^{-9}	0.1×10^{-9}
Network	(b)	0.8×10^{-4}	0.8×10^{-4}	0.8×10^{-4}
Storage	(f)	0.1×10^{-3}	0.1×10^{-3}	0.1×10^{-6}
Mvs	(r)	0.9×10^{-4}	0.1×10^{-6}	0.6×10^{-7}
Vtam	(r)	0.3×10^{-4}	0.5×10^{-7}	0.4×10^{-7}
Ims	(r)	0.3×10^{-4}	0.1×10^{-4}	0.3×10^{-5}
Database	(r)	0.3×10^{-4}	0.5×10^{-4}	0.2×10^{-7}
p1		0.2×10^{-2}	0.8×10^{-4}	0.1×10^{-5}
Coverage		0.2×10^{-4}	0.1×10^{-3}	0.1×10^{-6}

software are duplexed. To describe this model we used the same constructs for the duplexed components, which were named Proc2, Power2, Pc2, Mvs2, Vtam2, and Ims2, respectively. The component interactions were appropriately changed—for example, the operation of Proc2 depends upon Power2. The availability specification for this model is given at the end of Section 2. We also solved a third model, where we duplexed the storage and the database over and above the duplexing in Figure 4, by declaring storage as Storage1 and Storage2, and database as Database1 and Database2. [Component interactions determine when we can specify duplexed components as Cmpname(2) and when we have to specify them as Cmpname1 and Cmpname2. For example, Power1 supplies power to Proc1 and Pc1, and Power2 supplies power to Proc2 and Pc2. If Power1 and Power2 could supply power to both the processing subsystems, then we could have declared power as Power(2) because both the power supplies would have the same failure and repair behavior and the same component interactions.] The availability specification for the third model is as follows:

```
EVALUATION CRITERIA: BLOCKDIAGRAM
  Exp1: Database1 and Storage1 and Network
  Exp2: Exp1 and Power1 and Proc1 and Mvs1 and Vtam1 and Ims1
  Exp3: Database2 and Storage2 and Network
  Exp4: Exp2 and Power2 and Proc2 and Mvs2 and Vtam2 and Ims2
  Exp2 or Exp4
```

We call these three models simplex, duplex, and duplex + storage, respectively. The Markov chains constructed for these models have 264, 11616, and 34848 states, respectively (certainly not an easy task to do by hand). The steady-state availability (A) and its sensitivity with respect to various parameters are given in **Table 2**. The letter (f), (r), or (b) in parentheses after a component name indicates that the failure rate, the repair rate, or both for the component yielded the maximum absolute value of the sensitivity which is given in the table.

For the simplex system, the availability is most sensitive to p1, that is, to a processor failure where Softretry is successful. Therefore, if we want to improve the availability of the system, we should start by improving the software recovery mechanisms which could increase p1. Another alternative is to add a second processor, together with its associated hardware and software, as is done in the duplex system of Figure 4. Note that we get a major improvement in availability (that is, about five-times reduction in unavailability) by doing so, and the effect of processor failure (or a failure of any of its associated hardware and software) on the system availability has been drastically reduced. Network, Storage, and Database still have the same effect on availability as in the simplex system, as no availability improvements were made to these components. Note that coverage has a slightly higher sensitivity value because now two processors and two IMS software components can affect the database, as opposed to one of each in the simplex system. Further availability improvement can be achieved by duplexing the storage and the database, which now have the highest effect on availability. We did that for our third model. The results are shown in the last column of Table 2, which shows another two-and-a-half-times reduction in unavailability. Further improvement would have to come from duplexing the other parts of the system and simultaneously increasing the coverage value by improving the database recovery facility.

Besides design improvement, sensitivity analysis can also be very helpful in other ways. Note that for the simplex system, the availability is most sensitive to a processor failure, while it is least sensitive to a processor-controller failure. A few remarks can be made about that. First, we must estimate failure- and repair-rate parameters very carefully for the processor because they affect the availability the most. On the other hand, minor errors in the processor controller's parameter estimates are not going to affect the availability very much. In fact, since the processor controller's parameters have three orders of magnitude less effect on the availability than those of other components in the system, the processor controller can be removed from our system model to reduce the size of the model without significantly affecting the model results. We did this and found that the availability as well as the parameter sensitivities of the components in the three systems did not change significantly. Thus, sensitivity analysis is useful not only in design improvement, but also in model reduction and the identification of the critical parameters to be estimated.

5. Conclusions

The example discussed in this paper shows that the proposed modeling language has powerful modeling capabilities, particularly with regard to modeling component interactions. Large and complex models can be constructed using the language without having the modeler deal directly with the underlying Markov-chain models. This task would be very difficult if the Markov-chain transition-rate matrix had to be entered directly by the modeler. The numerical methods presented in the paper are particularly suited to solving large Markov-chain models. The storage requirements have been taken into account by exploiting sparse-matrix storage. The numerical methods, except for the computation of the distribution of availability, are capable of solving Markov-chain models with tens of thousands of states. We have solved models with up to 35000 states when the transition-rate matrix is stored in symbolic form, and much larger models when this matrix is stored in numerical form. The symbolic form of the transition-rate matrix is needed for sensitivity analysis because it requires symbolic differentiation of the transition-rate matrix with respect to the model parameters. The main limiting factor for the size of the model solved is the available memory space. Solution speed has not been a problem, nor has numerical stability.

For large, complex models it is hard to identify which model parameters affect the availability the most. Therefore, sensitivity analysis is an extremely useful aid in the design-improvement process. Moreover, sensitivity analysis can help identify the critical parameters to estimate; it can also identify noncritical components that may be removed from the model to reduce its size.

There are several directions for further work in this area. One is to improve the modeling capabilities of the language, for example by providing constructs for spare switch-in times. A second is to compute other types of measures, for example performability [16–19]. We are investigating both of these directions. A third is to develop methods for solving even larger models than are currently solved by the numerical methods presented in this paper. In this regard we are experimenting with simulation [20] and Markov-chain truncation, aggregation, and lumping techniques [4].

Appendix A: Syntax of SAVE language

```
MODEL: ⟨modelname⟩
  METHOD:⟨NUMERICAL|MARKOV|COMBINATORIAL|SIMULATION⟩
*
PARAMETERS: ⟨parameter-name⟩, ⟨parameter-name⟩, . . .
*
CONSTANTS: ⟨constant-name⟩, ⟨constant-name⟩, . . .
  CONSTANT-NAME: ⟨constant-value|expression⟩
  CONSTANT-NAME: ⟨constant-value|expression⟩

*
LISTS: ⟨list-name⟩, ⟨list-name⟩, . . .
  LIST-NAME: ⟨comp-name⟩⟨(no.-of-comps)⟩, . . .
  LIST-NAME: ⟨comp-name⟩⟨(no.-of-comps)⟩, . . .

*
COMPONENT:                       ⟨comp-name⟩⟨(no.-of-comps)⟩
  SPARES:                        ⟨no.-of-spares⟩
  SPARES FAILURE RATE:           ⟨expression⟩
  OPERATION DEPENDS UPON:        ⟨comp-name⟩⟨(no.)⟩, . . .
  REPAIR DEPENDS UPON:           ⟨comp-name⟩⟨(no.)⟩, . . .
  DORMANT WHEN SYSTEM DOWN:      ⟨YES|NO⟩
  DORMANT FAILURE RATE:          ⟨expression⟩
  FAILURE RATE:                  ⟨expression⟩, ⟨expression⟩, . . .
  FAILURE MODE PROBABILITIES:    ⟨prob-value⟩, . . .
  REPAIR RATE:                   ⟨expression⟩, . . .
  REPAIRMAN CLASS USED:          ⟨class-name⟩, . . .
  COMPONENTS AFFECTED:             ⟨NONE|list-name|comp-name(⟨no.⟩)⟩, . . .
    ⟨LIST-NAME|COMP-NAME⟩:         ⟨affect-prob-val⟩, . . .
    ⟨LIST-NAME|COMP-NAME⟩:         ⟨affect-prob-val⟩, . . .
COMPONENT:

*
EVALUATION CRITERIA: ⟨ASSERTIONS|BLOCKDIAGRAM|FAULTTREE|PERFORMANCE⟩

*
REPAIRMAN CLASS:          ⟨class-name⟩(⟨number⟩|UNLIMITED)
  REPAIR STRATEGY:        ⟨PRIORITY|ROS⟩
    COMPONENT-NAME:       ⟨priority-level⟩
    COMPONENT-NAME:       ⟨priority-level⟩
REPAIRMAN CLASS:

*
END
```

Appendix B: SAVE language description of the example

```
MODEL: Example
*
METHOD: NUMERICAL
*
CONSTANTS: procfr, procdfr, netfr, storagefr, powerfr, pcfr
  procfr:      1/960
  procdfr:     1/960
  netfr:       1/12000
  storagefr:   1/12000
  powerfr:     1/9600
  pcfr:        1/9600
*
CONSTANTS: mvsfr, vtamfr, imsfr
  mvsfr:   1/2400
  vtamfr:  1/4800
  imsfr:   1/4800
*
CONSTANTS: procrr1, procrr2
  procrr1:  10
  procrr2:  1/2
*
CONSTANTS: netrr, storagerr, powerrr, pcrr
  netrr:       1
  storagerr:   1
  powerrr:     1
  pcrr:        1
*
CONSTANTS: mvsrr, vtamrr, imsrr, dbrr
  mvsrr:    4
  vtamrr:  12
  imsrr:   12
  dbrr:     4
*
PARAMETERS: p1, coverage
*
LISTS: Cmplist1, Cmplist2
  Cmplist1: Database, Mvs1, Vtam1, Ims1
  Cmplist2: Vtam1, Ims1
*
```

```
COMPONENT:                        Proc1
  OPERATION DEPENDS UPON:         Power1
  REPAIR DEPENDS UPON:            Power1, Pc1
  DORMANT WHEN SYSTEM DOWN:       NO
  DORMANT FAILURE RATE:           procdfr
  FAILURE RATE:                   procfr
  FAILURE MODE PROBABILITIES:     p1, 1 − p1
  REPAIR RATE:                    procrr1, procrr2
  REPAIRMAN CLASS USED:           Softretry, Fieldengineer
  COMPONENTS AFFECTED:            NONE, Cmplist1
    Cmplist1: 1 − coverage, 1, 1, 1
*
COMPONENT:                        Mvs1
  OPERATION DEPENDS UPON:         Proc1, Power1
  REPAIR DEPENDS UPON:            Proc1, Power1
  DORMANT WHEN SYSTEM DOWN:       NO
  FAILURE RATE:                   mvsfr
  REPAIR RATE:                    mvsrr
  REPAIRMAN CLASS USED:           Operator
  COMPONENTS AFFECTED:            Cmplist2
*
COMPONENT:                        Ims1
  OPERATION DEPENDS UPON:         Mvs1, Proc1, Power1
  REPAIR DEPENDS UPON:            Mvs1, Proc1, Power1
  DORMANT WHEN SYSTEM DOWN:       YES
  FAILURE RATE:                   Imsfr
  REPAIR RATE:                    Imsrr
  REPAIRMAN CLASS USED:           Operator
  COMPONENTS AFFECTED:            Database
    Database: 1 − coverage
*
COMPONENT:                        Vtam1
  OPERATION DEPENDS UPON:         Mvs1, Proc1, Power1
  REPAIR DEPENDS UPON:            Mvs1, Proc1, Power1
  DORMANT WHEN SYSTEM DOWN:       YES
  FAILURE RATE:                   vtamfr
  REPAIR RATE:                    vtamfr
  REPAIRMAN CLASS USED:           Operator
*
```

```
COMPONENT:                          Storage
  DORMANT WHEN SYSTEM DOWN:         NO
  FAILURE RATE:                     storagefr
  REPAIR RATE:                      storagerr
  REPAIRMAN CLASS USED:             Fieldengineer
  COMPONENT AFFECTED:               Database
*
COMPONENT:                          Network
  DORMANT WHEN SYSTEM DOWN:         NO
  FAILURE RATE:                     netfr
  REPAIR RATE:                      netrr
  REPAIRMAN CLASS USED:             Fieldengineer
*
COMPONENT:                          Database
  OPERATION DEPENDS UPON:           Storage
  REPAIR DEPENDS UPON:              Storage
  DORMANT WHEN SYSTEM DOWN:         YES
  REPAIR RATE:                      dbrr
  REPAIRMAN CLASS USED:             Operator
*
COMPONENT:                          Power1
  DORMANT WHEN SYSTEM DOWN:         NO
  FAILURE RATE:                     powerfr
  REPAIR RATE:                      powerrr
  REPAIRMAN CLASS USED:             Fieldengineer
*
COMPONENT:                          Pc1
  OPERATION DEPENDS UPON:           Power1
  REPAIR DEPENDS UPON:              Power1
  DORMANT WHEN SYSTEM DOWN:         NO
  FAILURE RATE:                     pcfr
  REPAIR RATE:                      pcrr
  REPAIRMAN CLASS USED:             Fieldengineer
EVALUATION CRITERIA: blockdiagram
  Exp1: Mvs1 and Vtam1 and Ims1
  Exp1 and Database and Storage and Network
*
*
REPAIRMAN CLASS: Softretry(UNLIMITED)
REPAIRMAN CLASS: Operator(UNLIMITED)
*
REPAIRMAN CLASS: Fieldengineer(1)
  REPAIR STRATEGY: PRIORITY
    Network:  1
    Storage:  1
    Proc1:    2
    Power1:   3
    Pc1:      4
*
END
```

Acknowledgments

The work reported on in this paper and the SAVE package itself grew out of IBM SUR-SST-sponsored joint research between Duke University and the IBM Research Division. The choice of measures to compute and the choice of solution methods to incorporate in SAVE were strongly influenced by K. S. Trivedi of Duke University and W. J. Stewart of North Carolina State University. We would like to thank them for their important contributions. We would also like to thank W. C. Carter for providing us insights into real system behavior and E. de Souza e Silva, T. Risse, and A. N. Tantawi for helping us design and implement portions of the SAVE package. We are also pleased to acknowledge the help of C. Bading, P. Pignal, and G. Ahrens; their invaluable feedback from their use of the SAVE package resulted in a much improved language design.

References

1. A. Costes, J. E. Doucet, C. Landrault, and J.-C. Laprie, "SURF—A Program for Dependability Evaluation of Complex Fault-Tolerant Computing Systems," *Proc. FTCS-11* (Symposium on Fault-Tolerant Computing), Portland, ME, 1981, pp. 72–78.
2. S. V. Makam, A. Avizienis, and G. Grusas, "UCLA ARIES 82 User's Guide," *Technical Report No. CSD-820830*, University of California at Los Angeles, August 1982.
3. J.-C. Laprie, "Dependability Evaluation of Software Systems in Operation," *IEEE Trans. Software Eng.* **SE-10,** No. 6, 701–714 (November 1984).
4. A. Goyal, S. S. Lavenberg, and K. S. Trivedi, "Probabilistic Modeling of Computer System Availability," *Ann. Oper. Res.* **8,** 285–306 (1987).
5. A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg, and K. S. Trivedi, "The System Availability Estimator," *Proc. FTCS-16*, Vienna, Austria, 1986, pp. 84–89.
6. J. B. Dugan, K. S. Trivedi, M. K. Smotherman, and R. M. Geist, "The Hybrid Automated Reliability Predictor," *J. Guidance, Control, & Dynam.* **9,** No. 3, 319–331 (1986).
7. W. G. Bouricius, W. C. Carter, and P. R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems," *Proceedings of the ACM National Conference*, San Francisco, August 1969, pp. 295–309.
8. E. de Souza e Silva and H. R. Gail, "Calculating Cumulative Operational Time Distributions of Repairable Computer Systems," *IEEE Trans. Comput.* **C35,** No. 4, 322–332 (1986).
9. A. Goyal and A. N. Tantawi, "Numerical Evaluation of Guaranteed Availability," *Proc. FTCS-15*, Ann Arbor, June 1985, pp. 324–329.
10. W. J. Stewart and A. Goyal, "Matrix Methods in Large Dependability Models," *Research Report RC-11485*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, November 1985.
11. J.-C. Laprie, "Dependable Computing and Fault Tolerance: Concepts and Terminology," *Proc. FTCS-15*, Ann Arbor, June 1985, pp. 2–13.
12. A. Jensen, "Markov Chains as an Aid in the Study of Markov Processes," *Skand. Aktuarietidskr.* **36,** 87–91 (1953).
13. D. Gross and D. R. Miller, "The Randomization Technique as a Modeling Tool and Solution Procedure for Transient Markov Processes," *Oper. Res.* **32,** No. 3, 343–361 (1984).
14. W. K. Grassman, "Transient Solutions in Markovian Queueing Systems," *Comput. in Oper. Res.* **4,** 47–56 (1977).
15. P. Heidelberger and A. Goyal, "Sensitivity Analysis of Continuous Time Markov Chains Using Uniformization," *Proceedings of the Second International Workshop on Applied Mathematics and Performance/Reliability Models of Computer/Communication Systems*, Rome, Italy, 1987, pp. 93–104.
16. L. Donatiello and B. Iyer, "Analysis of a Composite Performance Reliability Measure for Fault Tolerant Systems," *Research Report RC-10325*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1985.
17. A. Goyal and A. N. Tantawi, "Evaluation of Performability for Degradable Computer Systems," *IEEE Trans. Comput.* **C-36,** No. 6, 738–744 (1987).

18. V. G. Kulkarni, V. F. Nicola, R. M. Smith, and K. S. Trivedi, "Numerical Evaluation of Performability and Job Completion Time in Repairable Fault-Tolerant Systems, "*Proc. FTCS-16*, Vienna, Austria, 1986; pp. 252–257.
19. J. F. Meyer, "On Evaluating the Performability of Degradable Computing Systems," *IEEE Trans. Comput.* **C-29,** No. 8, 720–731 (August 1980).
20. A. Conway and A. Goyal, "Monte Carlo Simulation of Computer System Availability/Reliability Models," *Proc. FTCS-17*, Pittsburgh, 1987, pp. 230–235.

Received November 13, 1986; accepted for publication August 11, 1987

Ambuj Goyal *IBM Research Division, P.O. Box 704, Yorktown Heights, New York 10598.* Dr. Goyal is a member of the Systems Analysis Group at the IBM Thomas J. Watson Research Center. He received his B.Tech. degree from the Indian Institute of Technology, Kanpur, in 1978, and the M.S. and Ph.D. degrees from the University of Texas at Austin in 1979 and 1982, respectively. His current interests are in computer architecture, communication networks, performance evaluation, and fault-tolerant computing. Dr. Goyal has served as the Program Committee Chairman of the Workshop on Reliability/Availability Modeling Tools and Their Applications. He is coauthor of a tutorial entitled "Effectiveness Evaluation of Multiprocess Systems" presented at the International Conference on Parallel Processing. Dr. Goyal is a member of the Institute of Electrical and Electronics Engineers and its Computer Society. He has received an IBM Outstanding Innovation Award for SAVE.

Stephen S. Lavenberg *IBM Research Division, P.O. Box 704, Yorktown Heights, New York 10598.* Dr. Lavenberg received the B.E.E. degree in 1963 from Rensselaer Polytechnic Institute, Troy, New York, and the M.S.E.E. and Ph.D. degrees in electrical engineering in 1964 and 1968, respectively, from the California Institute of Technology, Pasadena. He joined IBM in 1968 at the San Jose Research Laboratory and in 1976 transferred to the IBM Thomas J. Watson Research Center, where he has been the Senior Manager of the Systems Analysis Department since 1982. He was a Visiting Professor in the Computer Science Department at UCLA for the 1982–83 academic year. Dr. Lavenberg's research interests include computer performance modeling, queueing theory, and statistical aspects of simulation. He is the editor and a principal author of the *Computer Performance Modeling Handbook* (Academic Press, 1983). He was an associate editor of the *IEEE Transactions on Computers* for Performance Evaluation from 1979 to 1983 and since 1983 has been the *JACM* area editor for Computer System Modeling and Analysis. Dr. Lavenberg is a member of the Association for Computing Machinery and a senior member of the Institute of Electrical and Electronics Engineers. He has received IBM Outstanding Innovation Awards for Mean Value Analysis and for SAVE.