Modeling and Analysis of System Dependability Using the System Availability Estimator

Alvin M. Blum[§], Ambuj Goyal[§], Philip Heidelberger[§] [*], Stephen S. Lavenberg[§]
Marvin K. Nakayama[§§] [†], Perwez Shahabuddin[§]

[§] IBM T.J.Watson Research Center
Yorktown Heights, NY 10598

[§§] School of Management, Rutgers University
Newark, New Jersey 07102

Abstract

This paper reviews the System Availability Estimator (SAVE) modeling program package. SAVE is used to construct and analyze models of computer and communication systems dependability. The SAVE modeling language consists of a few constructs for describing the components in a system, their failure and repair characteristics, the interdependencies between components, and the conditions on the individual components for the system to be considered available. SAVE parses an input file and creates a Markov chain model. For small models numerical solution methods can be used, but for larger models (the state space grows exponentially with the number of components in the model), fast simulation techniques using importance sampling have to be used. We provide software demonstrations using both these techniques.

1 Introduction

This paper provides an overview of the System Availability Estimator (SAVE) program package. SAVE provides the capabilities to construct and solve continuous time Markov chain models of systems or subsystems in order to predict their dependability characteristics. (SAVE can also be used to construct and solve combinatorial models, but we do not discuss this aspect of SAVE here.) Using SAVE, it is not necessary to specify the details of the Markov chain, although this can be done. Rather, SAVE provides a higher level modeling language. It automatically constructs a Markov chain model from the user's modeling language description of the system.

When modeling a system using SAVE, the system is considered to be a collection of components, each of which is subject to failure and repair. The term "component" is used generically and can refer to hardware, (e.g. a processor), software (e.g. an operating system), a data structure (e.g. a locking table), or an environmental component (e.g. power or cooling unit). The term "repair" is used generically to refer to whatever is required to get the failed component to be functioning properly again, and could mean physical repair or replacement in the case of a hardware component or restart in the case of a software component. The term "failure" (with respect to a component) is also used generically and can refer to the onset of an unplanned outage of a component, or to the onset of a planned outage of a component (e.g. due to a hardware or software upgrade). In addition to components failing and being repaired independently of each other, various types of component dependencies and interactions are possible. For example, the failure of one component could cause another component to fail (e.g. a failing processor contaminates a data structure) or the repair of a component could depend on the proper functioning of another component (e.g. the restart of a failed database management subsystem requires the underlying operating system to be functioning properly). or the repair of a component is delayed because some other component is being repaired by the single repair person.

A system is considered to be available (operational and functioning properly) when either all, or a specified subset, of its components are available. For fault tolerant systems, it is usually the case that not all components need be available for the system to be considered available and the system can operate in various degraded performance levels depending on which subsets of components are failed. The purpose of the model is to predict the system's dependability characteristics (e.g. mean fraction of time the system is available, mean time until the system first becomes unavailable, etc.) from the failure and repair characteristics of the components that comprise the system including component interactions. This is done by solving the model to compute the dependability measures of interest. In addition to computing dependability measures,

[*]Currently on a sabaticcal leave at ICASE, NASA Langley Research Center, Hampton, Virginia 23665.

[†]The author's contribution was made when he was a Post Doctoral Fellow at IBM Research.

0363-8928/94 $3.00 © 1994 IEEE

SAVE computes the *sensitivity* of each of the measures with respect to the input parameters of the model, e.g. failure rates and repair rates. The sensitivity of a measure A with respect to an input parameter p is the derivative of the measure with respect to the parameter, normalized by multiplying by the parameter, i.e., $(dA/dp)p$. Sensitivity analysis can be used to determine which component parameters have the greatest effect on system dependability and hence where design improvements may be needed.

SAVE is written in standard FORTRAN 77. It runs on IBM's VM family of S/370 and S/390 systems and under AIX on the RS/6000. The first version of SAVE was released in November 1985 and was used in IBM, along with other tools, to model high availability system designs being proposed to the FAA for the Advanced Automation System. It has since been used in other parts of IBM and has been significantly enhanced. The current version was released in June 1993.

2 The Save Modeling Language

Specifying a SAVE model consists of three main parts:

- Describing the properties of individual components and their interactions.
- Specifying the resources and strategies available for the repair/recovery of failed components of the system.
- Relating the states of the individual components to the overall state of the system.

We will now explain some of the syntax that is used for accomplishing the above. We will illustrate the use of these constructs in availability modeling with an example. For a complete description of the constructs and several other examples the reader is referred to [1].

Example: Consider a shared memory multiprocessor (proc) running a common operating system (os) The system has four databases (disk1, disk2, disk3 and disk4), each residing on three different disks. The data in each disk are replicated such that any one of the three disks containing each database can fail without causing any loss of data. There are two operational control units (cu) connected to the databases. Hence for all data to be safe and accessible, one processor, one control unit and two of the three disks in each database must be operational. The system is shown in Figure 1. The complete SAVE model specification is given in the appendix (lines starting with '*' are comment lines) and should be referred to in the following discussion.

2.1 Preliminary Constructs

The first few constructs are preliminary. The MODEL construct gives the model a name, and the METHOD construct indicates the method to be used in dependability evaluation. One can use either of four choices in the METHOD construct: NUMERICAL, SIMULATION, MARKOV, COMBINATORIAL. The constructs described here are valid for the SIMULATION and NUMERICAL methods (only a subset of these constructs can

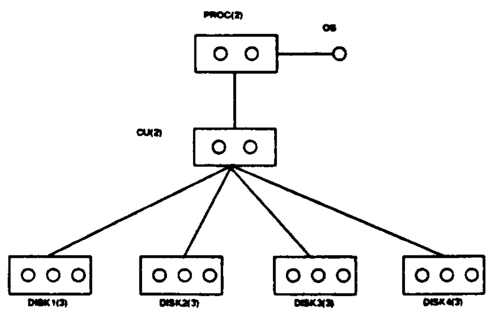

Figure 1: Computer system model of example

be used with the NUMERICAL method). Constants and parameters to be used in the model are specified using the CONSTANTS and PARAMETERS construct, respectively. Failure rates, repair rates and other quantities of interest can be expressed in terms of arithmetic expressions involving numbers and these constants and parameters. For ease in specifications components may be grouped into lists. These are specified by the LISTS construct. In the example we have a list called proclist consisting of the os and any one of the proc.

2.2 Properties of Components

The properties of components in SAVE are specified using the COMPONENT construct. In the example we have seven component constructs, one each for the proc, os, cu, disk1, disk2, disk3 and disk4. The specific properties of the component type are specified through other subconstructs that follow the COMPONENT construct. The FAILURE RATE construct is used to define the failure rate of the component type when the component is operational. (In SAVE there is a default that the failure rates of all components are zero when the system is down. Constructs for overriding this default can be found in [1].) A component may fail in different modes distinguished by different repair rates and/or different classes of repairmen used for the repair process and/or different impact on other components. The probabilities of failing in each mode are specified by the FAILURE MODE PROBABILITIES construct. The REPAIR RATE construct is used to specify the repair rate in each failure mode and the REPAIRMAN CLASS USED construct is used to specify which type of repairman is used to repair the component when it fails in each of its failure modes.

In SAVE we can also specify redundancies of components in the system using the SPARES construct. Spares may be assigned a different failure rate than operational components by using the SPARES FAILURE RATE construct. Unless otherwise specified, it is assumed that the time it takes to switchover from a failed component to one of its spares (if available) is zero. For non-zero switchover times one has to use a set of additional "switchover" constructs, details of which can be obtained from [1].

2.3 Component Interactions

SAVE allows the user to model several types of component interactions. For instance, in the example, the operation of the os may depend on one of the proc being operational. This is specified by the OPERATION DEPENDS UPON construct. The simplest type of specification in an OPERATION DEPENDS UPON construct consists of one or more *component state variables*, each of which consists of a component type name and an associated number. The value of the component state variable is true if the number of operational components of the type is equal to or exceeds the specified number. A component is operational if all component state variables in its OPERATION DEPENDS UPON construct have the value true. Otherwise, it is said to be in the DORMANT state. The repair of a component type may also depend on other components being operational, and may be specified in a similar fashion by using the REPAIR DEPENDS UPON construct.

More general dependency conditions are also possible. We can have component state variables linked together by the OR logical operator. Moreover, we can use simple logical expressions instead of component state variables. For example, we can give conditions of the type that the number of components in a certain state (operational, dormant, spare etc.) is " $=$ ", " $\neq$ ", " $>$ ", " $>=$ ", " $<$ ", or " $<=$ " a certain specified number. Refer to [1] for a more detailed description of this feature and how it can be used in modeling complex systems.

It is common in complex systems that when a component fails it causes other components to fail with certain probabilities (distinguish this from causing another components to become dormant; a dormant component does not need repair but a failed component does). This is specified by the COMPONENTS AFFECTED construct. In the example, the proc can fail in two modes. The failure of a proc in the first mode may crash both the elements of proclist (the os and the other proc), the former with probability 1 and the latter with probability 0.25. In the second mode no components are affected (meaning that the operating system continues to function on the remaining operational processor).

2.4 Repairman Classes and Repair Discipline

There can be several different repairman classes in a system. The name of a repairman class and the number of repairmen in that class are declared by the REPAIRMAN CLASS construct. In the example we have two classes: field engineers (fe) and software repairmen (swrep). The REPAIR STRATEGY construct determines in what order failed components will be repaired. Any one of five options can be specified in this construct. If all components repaired by the repairman class are of the same priority level (e.g. swrep), then we can use either the FCFS (first come first served) or ROS (random order service) option. In the case of different priority levels, we can use either PRIORITY, FCFSPR or FCFSNP. PRIORITY is used to indicate a preemptive repair discipline with random order service used for repairing components of the same priority. FCFSPR and FCFSNP are used to indicate preemptive and non-preemptive priority disciplines respectively, with first come first served used for repairing components of the same priority. The priority levels of the components are specified on succeeding lines.

2.5 Evaluation Criteria: When Is The System Considered Available?

The user can describe conditions defining when the system is considered to be up (available) or down (unavailable). Whether the system is up or down depends on the states of the individual component types, i.e., which components are operational and which are not. The EVALUATION CRITERIA construct specifies the conditions on the states of the components under which the system is considered up. One can specify this in one of three ways. ASSERTIONS are the simplest way to specify system up conditions. For each component type a number is specified which indicates the number of components of the given type that are *required* to be operational for the system to be considered up. For more complex system up conditions, the BLOCKDIAGRAM option can be used. In this option, a Boolean expression describing the system up condition is specified. The Boolean expression is in terms of component state variables and *list state variables*. List state variables are similar to component state variables, except now we use the total number of operational components among all component types specified in the list. The system is considered available if the value of the Boolean expression is true. Sometimes the expression in the BLOCKDIAGRAM method may get too long and cumbersome. To simplify it one may predefine some part of the expression as a subexpression and use it in the final expression. In our example we predefine a subexpression called disks.

2.6 Evaluation Criteria: Performance Levels

Sometimes, even though the system may be up, it may not be performing at full efficiency but rather may be in one of several degraded modes of operation. In such cases different user defined performance levels can be assigned to the degraded modes of operation using the PERFORMANCE option in the EVALUATION CRITERIA. Consider the case where the system above is now considered to have 3 operational levels of performance: levels 3, 2 and 1 (plus level 0 for system down configurations). Level 3 consists of the state in which all the components in the system are operational. Level 2 consists of the state in which at least one proc, one cu and 10 out of the total of twelve disks of the databases are operational. At the same time, no database has more than 1 out of 3 disks failed. Level 1 is the bare minimum configuration which we require for the system to be considered up. Define a list consisting of disk1, disk2, disk3 and disk4 (using the LISTS construct) and call it lsdisk. Then the new evaluation criteria is:

```
EVALUATION CRITERIA: PERFORMANCE
   PERFORMANCE LEVEL: 3
```

```
proc(2) and os and cu(2) and lsdisk(12)
PERFORMANCE LEVEL: 2
disks: disk1(2) and disk2(2) and disk3(2) and disk4(2)
proc(1) and os and cu(1) and disks and lsdisk(10)
PERFORMANCE LEVEL: 1
proc(1) and os and cu(1) and disks
```

Note that the list state variable lsdisk(10) has the value true if 10 out of the total of 12 disks are operational. If we used only component state variables we would have to write down all combinations (see [1]).

3 Dependability Measures

Dependability measures that evaluate the behavior of the system within a fixed time horizon are called transient; otherwise, they are called non-transient. SAVE evaluates the following transient and non-transient measures:

- Expected Interval Unavailability = mean fraction of time during the time interval $(0, t]$ that the system is unavailable.
- Unreliability = probability that the system becomes unavailable during the time interval $(0, t]$.
- Guaranteed Availability = distribution of the interval availability.
- Steady State Unavailability = fraction of time during an arbitrarily long time interval that the system is unavailable (limit of interval availability as $t \to \infty$).
- Steady State Performance Probabilities = fraction of time in each performance level during an arbitrarily long time interval.
- Steady State Performance = mean performance level during an arbitrarily long time interval.
- Mean Time to Failure = mean time until the system first becomes unavailable.
- Mean Time to Performance Levels= mean time to reach each performance level.

As mentioned in the Introduction, in addition to computing dependability measures, SAVE computes the *sensitivity* of each of the measures with respect to the input parameters of the model, e.g. failure rates and repair rates.

4 Solution Methods

The main assumption in SAVE is that all component failure times and component repair times are exponentially distributed, so that the system can be modelled as a continuous time Markov chain (CTMC). The numerical method in SAVE parses the model specification and generates the states and the explicit rate matrix of this Markov chain. Numerical techniques are then used to solve for the various dependability measures. The reader is referred to [2] and [3] for details of the techniques used.

Note that the size of the state space grows exponentially with the number of component types. Hence for larger models, the numerical method tends to consume a significant amount of CPU time and memory to generate the rate matrix of the Markov chain. This can be made more manageable by representing components whose behavior is identical by a single component type (in effect exploiting symmetry in the system and lumpability to reduce the state space size). In addition state space truncation can be used to obtain an approximate solution. Alternatively, one can simulate the model (without explicitly generating the rate matrix).

The main problem with simulation is that since the component failure rates are much lower than the component repair rates, very rarely are enough components failed for the system to be considered failed. Hence SAVE incorporates an importance sampling based fast simulation technique called failure biasing ([5]). All transitions of the Markov chain are either a component failure transition or a component repair transition. In the original Markov chain, from any state of the Markov chain, except the state in which all components are operational, the total probability of repair transitions is almost 1 and the total probability of failure transitions is almost 0. In failure biasing the probability of failure transitions is increased to p which is some number much greater than zero (in SAVE we use p = 0.5) and thus the total probability of repair transitions is decreased to 1-p. The resulting estimate is then adjusted using a correction factor known as a likelihood ratio to give an unbiased estimate. The type of failure biasing which we use in SAVE is called balanced failure biasing ([7],[4]) in which the conditional probabilities of individual failure transitions, given that a failure transition has happened, are all made equal. This technique was shown to have the bounded relative error property (the relative error of the simulation estimate remains bounded as failure rates tend to zero, unlike standard simulation in which it tends to infinity) in the estimation of non-transient measures ([8]) and their sensitivities ([6]). For transient measures, if the time horizon t is small, then the first component failure transition typically happens after t. In a technique called forcing ([5]) we sample the time of the first transition from the conditional distribution of it being less than t. Forcing combined with balanced failure biasing gives bounded relative error in the estimation of transient measures and their sensitivities, when t is small ([6], [10]).

5 Planned Demonstration

We plan to take the example in this paper, and use SAVE for its dependability evaluation. We will also examine a case of sensitivity analysis.

6 Acknowledgements

We wish to acknowledge several people who played a role in building the earlier (numerical) version of SAVE ([2], [3]) which laid the foundation for the current version. These include W.C. Carter, E. de Souza e Silva, T. Risse, W.H. Stewart, A.N. Tantawi and K. Trivedi. We are also grateful to G. Ahrens, M. Kanthnathan, L. Spainhower and C. Lin for valuable discussions based on their experience with the SAVE package. Finally, we wish to thank S. Juneja for testing some of the new constructs.

References

[1] A. Blum, P. Heidelberger, S. S. Lavenberg, M. K. Nakayama, P. Shahabuddin (1993), "System Availability Estimator (SAVE) Language Reference and User's Manual Version 4.0", *IBM Research Report RA 219S.*

[2] A. Goyal, W. C. Carter, E. de Souza e Silva, S. S. Lavenberg and K. S. Trivedi (1986), "The System Availability Estimator", *Proceedings of the Sixteenth Symposium on Fault-Tolerant Computing*, IEEE Press, 84-89.

[3] A. Goyal, and S. S. Lavenberg (1987), "Modeling and Analysis of Computer System Availability", *IBM Journal of Research and Development*, 31, 6: 651-664.

[4] A. Goyal, P. Shahabuddin, P. Heidelberger, V. F. Nicola and P. W. Glynn (1992), "A Unified Framework for Simulating Markovian Models of Highly Reliable Systems", *IEEE Transactions on Computers*, C-41: 36-51.

[5] E. E. Lewis and F. Bohm (1984), "Monte Carlo Simulation of Markov Unreliability Models", *Nuclear Engineering and Design*, 77: 49-62.

[6] M. K. Nakayama (1991), "Asymptotics of Likelihood Ratio Derivative Estimators in Simulations of Highly Reliable Markovian Systems", IBM Research Report RC 17357, Yorktown Heights, New York. To appear in *Management Science.*

[7] P. Shahabuddin (1990), "Simulation and Analysis of Highly Reliable Systems", Ph.D. Thesis, Department of Operations Research, Stanford University, California.

[8] P. Shahabuddin (1994), "Importance Sampling for the Simulation of Highly Reliable Markovian Systems", *Management Science*, 40, 3: 333-352.

[9] P. Shahabuddin (1993), "Fast Transient Simulation of Markovian Models of Highly Dependable Systems", *Proceedings of the PERFORMANCE '93 Conference.* To appear in special issue of *Performance Evaluation.*

[10] P. Shahabuddin and M. K. Nakayama (1993), "Estimation of Reliability and its Derivatives for Large Time Horizons in Markovian Systems", *1993 Winter Simulation Conference Proceedings.*

Appendix

```
MODEL: Example
METHOD: SIMULATION
* Failure Rates (per day)
PARAMETERS: osfr
CONSTANTS: diskfr, cufr, procfr
  procfr: 1/365
  diskfr: 1/730
  cufr: 1/365
* Repair Rates (per day)
CONSTANTS: hwrr, osrr
  hwrr: 1
  osrr: 24/2
* Failure Mode Probabilities
CONSTANTS: pc
  pc: 0.25
LISTS: proclist
  proclist: os, proc(1)
COMPONENT : proc(2)
  FAILURE RATE : procfr
  FAILURE MODE PROBABILITIES: pc, 1-pc
  REPAIR RATE : hwrr, hwrr
  REPAIRMAN CLASS USED : fe, fe
  COMPONENTS AFFECTED : proclist, none
              proclist: 1.0, 0.25
COMPONENT : os
  FAILURE RATE : osfr
  REPAIR RATE : osrr
  REPAIRMAN CLASS USED : swrep
  OPERATION DEPENDS UPON: proc(1)
  REPAIR DEPENDS UPON: proc(1)
COMPONENT : cu(2)
  FAILURE RATE : cufr
  REPAIR RATE : hwrr
  REPAIRMAN CLASS USED : fe
COMPONENT : disk1(3)
  FAILURE RATE : diskfr
  REPAIR RATE : hwrr
  REPAIRMAN CLASS USED : fe
COMPONENT : disk2(3)
  FAILURE RATE : diskfr
  REPAIR RATE : hwrr
  REPAIRMAN CLASS USED : fe
COMPONENT : disk3(3)
  FAILURE RATE : diskfr
  REPAIR RATE : hwrr
  REPAIRMAN CLASS USED : fe
COMPONENT : disk4(3)
  FAILURE RATE : diskfr
  REPAIR RATE : hwrr
  REPAIRMAN CLASS USED : fe
EVALUATION CRITERIA: BLOCKDIAGRAM
  disks: disk1(2) and disk2(2) and disk3(2) and disk4(2)
  proc(1) and os and cu(1) and disks
REPAIRMAN CLASS: swrep
  REPAIR STRATEGY: FCFS
REPAIRMAN CLASS: fe(2)
  REPAIR STRATEGY: PRIORITY
  proc: 1
  cu: 2
  disk1: 3
  disk2: 3
  disk3: 3
  disk4: 3
END
```