*** RELATÓRIO TÉNICO ***

PERFORMABILITY ANALYSIS OF COMPUTER SYSTEMS: FROM MODEL SPECIFICATION TO SOLUTION

Edmundo de Souza e Silva*

H. Richard Gail**

NCE 06/91

Abril/91

Universidade Federal do Rio de Janeiro

Núcleo de Computação Eletrônica

Caixa Postal 2324

20001 - Rio de Janeiro - RJ

BRASIL

* Universidade Federal do Rio de Janeiro

** IBM Research Division

T.J.Watson Center - NY/USA

PERFORMABILITY ANALYSIS OF COMPUTER SYSTEMS: FROM MODEL SPECIFICATION TO SOLUTION

RESUMO

Modelagem de disponibilidade/confiabilidade de sistemas de computação diz respeito a representação de mudanças na estrutura do sistema sendo modelado, geralmente causadas por falhas, e como essas mudanças afetam a disponibilidade do sistema.

Modelagem de desempenho, por outro lado, é voltada para a representação da natureza probabilística das demandas dos usuários e tenta prever a capacidade do sistema para realizar tarefas, supondo-se que a estrutura do sistema permanece constante.

Com o advento de sistemas degradáveis, o sistema pode ser reestruturado em consequência a falhas e pode continuar a realizar tarefas, mesmo a uma capacidade reduzida.

Análise de desempenhabilidade ("performability") considera o efeito das mudanças estruturais e o seu impacto no desempenho global do sistema.

A complexibilidade dos atuais sistemas de computação, e a variedade de diferentes problemas a serem analisados, incluindo a avaliação simultânea de desempenho e disponibilidade, demonstram a necessidade de ferramentas sofisticadas que permitam a especificação de uma classe geral de problemas além de incorporarem técnicas poderosas analíticas e/ou de simulação.

No que diz respeito a especificação do modelo, é discutido um paradigma orientado a objeto, recentemente proposto, que acomoda uma grande variedade de aplicações.

As principais vantagens deste paradigma são discutidas e é feita uma comparação com outras técnicas.

No que diz respeito a métodos de solução, é feito um breve apanhado de desempenhabilidade de modelos de Markov. É mostrado então que muitas medidas relacionadas a desempenhabilidade podem ser calculadas usando-se a técnica de aleatorização (uniformização), marcando-se estados e/ou transições do modelo Markoviano do sistema sendo estudado. Finalmente, o problema de explosão de estados é abordado, e várias técnicas para lidar com o problema são discutidas.

RC 16961 (#75239) 6/20/91
Computer Science 56 pages

Research Report

Performablity Analysis of Computer Systems: from Model Specification to Solution

Edmundo de Souza e Silva

Federal University of Rio de Janeiro, NCE
Rio de Janeiro, Brazil

H. Richard Gail

IBM Research Division
T. J. Watson Research Center
Yorktown Heights, NY 10598

Performability Analysis of Computer Systems: from Model Specification to Solution

Edmundo de Souza e Silva[1]
Federal University of Rio de Janeiro, NCE
Rio de Janeiro, Brazil

H. Richard Gail
IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598

Abstract

Computer system reliability/availability modeling deals with the representation of changes in the structure of the system being modeled, which are generally due to faults, and how such changes affect the availability of the system. On the other hand, performance modeling involves representing the probabilistic nature of user demands and predicting the system capacity to perform useful work, under the assumption that the system structure remains constant. With the advent of degradable systems, the system may be restructured in response to faults and may continue to perform useful work, even though operating at lower capacity. Performability modeling considers the effect of structural changes and their impact on the overall performance of the system. The complexity of current computer systems and the variety of different problems to be analyzed, including the simultaneous evaluation of performance and availability, demonstrate the need for sophisticated tools that allow the specification of general classes of problems while incorporating powerful analytic and/or simulation techniques. Concerning model specification, a recently proposed object oriented modeling paradigm that accommodates a wide variety of applications is discussed and compared with other approaches. With respect to solution methods, a brief overview of past work on performability evaluation of Markov models is presented. Then it is shown that many performability related measures can be calculated using the uniformization or randomization technique by marking distinguished states and/or transitions of the Markov model of the system being studied. Finally, the state space explosion problem is addressed and several techniques for dealing with the problem are discussed.

[1]The work of E. de Souza e Silva was supported by National Science Foundation grant INT-8514377 and by a grant from CNPq(Brazil).

1 Introduction

During the last twenty years, the modeling and analysis of computer systems has received increasing attention from researchers and practitioners who wish to understand and predict the behavior of these systems. The greater the dependence of society on computer and communication systems, the greater the need for efficient and reliable machines. Therefore, it is crucial that accurate answers be given to questions such as: "How long can the system be expected to work without interruption?"; "How much work can the system be expected to accomplish before a failure?"; "What is the probability that the system operates above a certain level of efficiency during an observation period?"

In the past, most modeling work has concentrated on developing tools and techniques to analyze (exclusively) either: (a) the reliability or the availability of the system to the user; or (b) the system performance under the assumption that it is perfectly reliable. Availability/reliability (*dependability*) modeling represents the changes that may occur in the system structure, typically caused by faults in its components. Events that cause faults to occur are probabilistic in nature, and the model captures the resulting effect in the components as well as the way the system handles the faults (fault recognition, isolation of faulty components with possible substitution of spares, etc.). Measures such as the probability that the system operates successfully during a period of time or the percentage of time that the system can perform useful work are among the many important measures of interest to the analyst. Performance modeling, on the other hand, represents the capacity of the system to perform useful work under the assumption that no structural changes occur. In other words, the model should capture the probabilistic nature of user demands for a set of system resources and the consequent contention for these resources. The effect of resource contention is measured in terms of quantities such as throughput, average task completion time, etc.

With the advent of degradable systems, some capacity may be lost when structural changes occur in maintaining operation after a fault. Yet the system may still perform useful work after a reconfiguration, although it may operate at a different "capacity level." Thus its performance cannot be accurately evaluated without taking into account the impact of the structural changes. It is therefore desirable to define new combined measures of dependability and performance. For example, consider a multiprocessor system with several CPUs that process the submitted tasks. When a CPU fails, the system may continue to operate (if the fault is properly recognized and isolated), but clearly performance degrades in such a case. As another example, consider a database system having many disks, with the replication of data for availability purposes. The loss of a disk may not affect data availability, but it may affect performance during read accesses, since the load generated by read requests to a data unit is balanced over fewer disks.

Early modeling studies on gracefully degradable systems can be found in the work of

Borgerson and Freitas [7], who developed a model for reliability analysis of such systems. Meyer [66] introduced the notion of "computational success" and showed how it differs from the usual reliability measures. Beaudry [4] and Meyer [67] were among the first who developed measures to consider the interaction between reliability and performance. In the pioneering work of Meyer [68], a general modeling framework was introduced for the definition and evaluation of new measures that are called *performability* measures. Meyer's definition of performability encompasses many different measures. In broad terms, consider a stochastic process describing, for example, the evolution of the system structure over a finite time interval $(0, t)$. For each sample path u, a function $\gamma_t(u)$ that indicates the performance level associated with u is defined. The performability is the probability that the system performs at a level in a given subset of the performance levels during the specified time interval. For instance, $\gamma_t(u)$ may depend on the total accumulated performance during $(0, t)$, on the number of times a certain event occurs in u over $(0, t)$, etc.

There are several issues involved in relating the performance levels to sample paths in the structural model, i.e. how does one obtain $\gamma_t(u)$ in order to calculate the desired performability measure. Clearly, it is not practical to assign a level to each sample path in the model. Instead, a performance level usually is associated with a state of the structural system model. In carrying out such a procedure, some assumptions are commonly made. To clarify certain issues, consider a simple model of a distributed system as shown in Figure 1. In this system, submitted tasks are dispatched to one of the two existing processors for

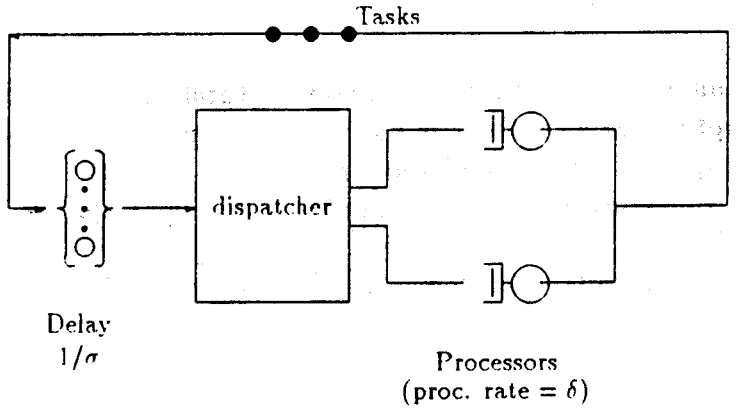

Figure 1: A simple distributed system model.

execution. Once submitted, a dispatcher is in charge of selecting a processor to execute the task according to the following rule. The processor with the smallest queue is chosen to execute a new task, and ties are broken randomly with equal probability. Each processor fails with exponential rate λ and is repaired with exponential rate μ. Once a processor fails, the tasks in its queue are instantaneously transferred to the other processor. In this example, we assume that there is a fixed number of N tasks in the system, which are independently submitted after an exponential delay with mean $1/\sigma$, and that each task is processed at an

exponential rate δ. Our interest is in calculating the probability that the total throughput during $(0, t)$ is above a certain level, i.e. $P[\text{thr}(0, t) > y]$. Figure 2 shows the transition rate matrix of the model of a system with $N = 3$ tasks and for which only processor 1 can fail. The state of the system is a triple that indicates: (a) the number of tasks queued at processor

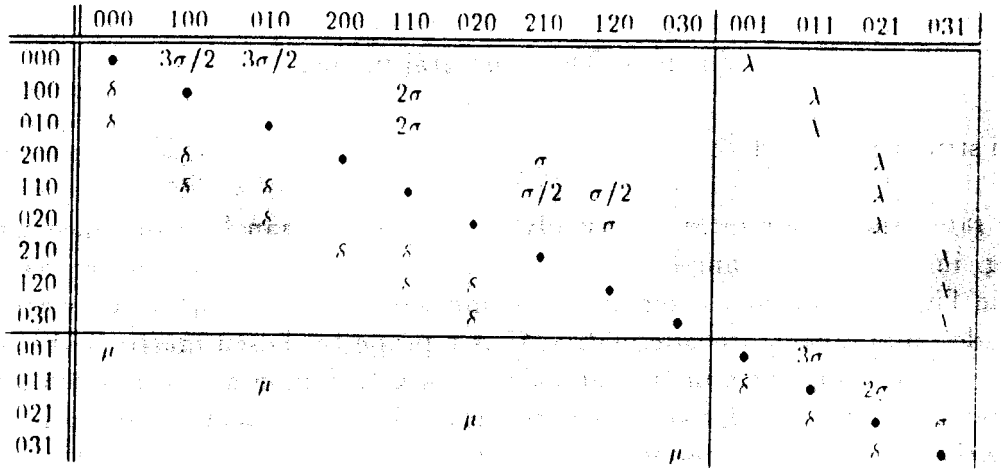

Figure 2: The transition rate matrix of the distributed system model.

1, (b) the number of tasks queued at processor 2 and (c) if processor 1 is operational (1) or failed (0). Clearly, the model corresponding to Figure 2 captures both the structural changes that occur in the system (a processor fails) and the performance obtained from the number of tasks in each queue. In order to calculate the throughput probability discussed above, a reward r_i is associated with each state a_i in the system. The reward rate in state a_i is equal to the throughput for that state. The performance measure we want to calculate is simply the distribution of the cumulative reward averaged over t. In terms of Meyer's definition of performability, we have

$$\gamma_t(u) = \begin{cases} 1 & \text{if } \frac{1}{t}\int_0^t r_{u(s)}ds > y \\ 0 & \text{otherwise} \end{cases}$$

where $u(s)$ is the state of the system at time s for sample path u, and we wish to find the probability that $\gamma_t = 1$.

Solving the performance-structural model of Figure 2 is more costly than first solving separate models which represent the throughput of the system for a given structure and the evolution of the system structure (number of working processors) over time, and then combining the results. In other words, we would like to decompose the combined model into distinct models. In Figure 2, the states are differentiated according to whether they represent both processors working or only one processor working. The structural model is obtained after aggregating states of Figure 2 with the same system structure and has only two states as in Figure 3. It remains to find the reward associated with each state of the

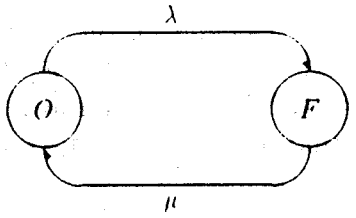

Figure 3: The structural model.

aggregated structural model.

Failure rates and repair rates are usually much smaller than "performance rates" in a system, e.g. in the above example, $\lambda, \mu \ll \sigma, \delta$. From this observation, we see that the matrix $\mathbf{Q}$ in Figure 2 is nearly completely decomposable, i.e. $\mathbf{Q} = \mathbf{Q}^* + \epsilon\mathbf{C}$, where $\mathbf{Q}^*$ is a completely decomposable stochastic matrix, $\mathbf{C}$ is a properly chosen matrix and ϵ is a small constant, $\epsilon \ll$ transition rates in the submatrices of $\mathbf{Q}^*$. Simon and Ando [90] formalized the decomposition technique for stochastic models. They have shown that an approximate solution for the state probabilities of the complete system can be found from the solution of the submatrices of $\mathbf{Q}^*$ and the matrix obtained after aggregating states in each submatrix of $\mathbf{Q}^*$ (see also Courtois [17]). For $t \to \infty$, Courtois [16] has shown that the error resulting from this approximation is $O(\epsilon)$. In our example, this is equivalent to first solving performance models for each of the two structures, determining the steady state throughput rates r_O and r_F and then using these rates in the aggregate two state structure model to obtain the overall solution. Unfortunately, the error that results from this approach is only bounded as $t \to \infty$. We know of no bounds that exist for finite t. Nevertheless, this approach has also been used for t finite and "large enough", i.e. under the assumption that the performance measure of interest reaches steady state between changes in the system structure.

The above example illustrates several issues concerning the evaluation of performability measures. First, depending on the rates in the model and the length t of the observation interval, one may not be able to apply decomposition as described above. Second, even if the usual decomposition assumptions are satisfied (which happens in most practical cases), the analyst must deal with solving performance models as well as dependability models and properly combine their results. Since these models may have many states it is very important that the model construction is automated, and thus it is highly desirable that the modeling paradigm used for system specification can properly describe both system performance and dependability behavior. Third, an important issue is how to properly merge the models, i.e. how are the rewards calculated (from a high level specification of a measure) using the performance model and how are they then associated with states in the structural model. Since the structural model may have hundreds or thousands of states, it is important that the reward assignment be specified at the system level, not at the state level. Finally, the evaluation of a measure itself raises important issues. Among these we mention:

computational complexity, numerical problems, tightness of error bounds, etc. Simplicity of the algorithm that is chosen may also be important, not only for didactic reasons, but also for its ease of use by practitioners.

It is the purpose of this paper to discuss the issues mentioned above, from model specification to solution techniques. We begin in section 2 by introducing notation and formally defining various performability measures. The measures include those commonly used in dependability analysis, since they are special cases of "strict" performability measures, i.e. those for which performance is taken into account. In section 3 issues related to the specification of performability models are discussed. The section includes a brief survey of tools that have been used for that purpose, with a concentration on a new modeling paradigm (object oriented) that has been recently proposed. Section 4 is devoted to model solution techniques, and it is divided into several subsections. We first briefly survey results in the literature for solving Markov chain performability models. We then describe in detail the uniformization or randomization technique, which has been used successfully to calculate several performability measures, and show that many of these measures can be calculated using the same framework. Finally we consider issues that are specifically concerned with the state space explosion problem, and we survey some of the techniques proposed to alleviate this problem, including both those for transient and steady state analysis. Our conclusions are presented in section 5.

2 Notation and Measure Definitions

Most models that have been used for performability evaluation are based on Markov analysis. A Markov process describes the structural changes in the system as faults and/or repairs occur [35]. For performance evaluation, queueing models are widely used, and results from queueing theory and networks of queues are generally applied. However, for models that do not satisfy product form requirements nor have some other closed form solution, the most general approach is still to build and numerically solve a Markov chain performance model. In the example presented in the introduction, the Markov transition matrix of Figure 2 describes both resource contention and structural changes for performability evaluation. If decomposition/aggregation is used, the two state model of Figure 3 represents the structural changes. In order to calculate the rewards that are assigned in the two state model, two queueing models have to be solved. The first model corresponds to a fully operational system (two working processors). However, since this model represents a shortest queue routing, it does not possess a product form solution, and a Markov chain model is used to obtain the throughput. The second model (only one processor working) is a simple queueing model and has a closed form solution.

Before defining various performability measures, we first describe some basic notation used throughout the paper. Since, as mentioned above, Markov chain models are used most frequently, we introduce the notation in this context. Consider a homogeneous continuous time Markov process $X = \{X(t), t \geq 0\}$ that describes the behavior of the system (structural, queueing or both), and let $S = \{a_i, i = 1, \ldots, M\}$ be the finite state space associated with the model. We assume there are $K + 1$ rewards $\rho_1 > \cdots > \rho_{K+1}$ which may be associated with states or transitions. The special dependability case is of particular interest, and we let S_O be the set of states that represent an operational system and S_F be the remaining set of states that represent a failed system. A vector is designated $\mathbf{v} = (v_i)$, while a matrix is written $\mathbf{A} = [A_{ij}]$. Usually $\mathbf{Q}$ denotes a transition rate matrix of a continuous time Markov chain, while $\mathbf{P}$ represents a (stochastic) transition probability matrix of a discrete time Markov chain. Given two vectors $\mathbf{u}$ and $\mathbf{v}$ of the same length, recall the inner product notation $\mathbf{u} \cdot \mathbf{v} = \sum_i u_i v_i$. We also write $\|\mathbf{v}\| = \sum_i v_i = \mathbf{1} \cdot \mathbf{v}$.

We now define various performability measures, and this also serves to introduce further notation. Note that, as mentioned in the introduction, such measures can be described in terms of the sample path framework of Meyer, although that approach is not followed here.

- Point availability.

  Point availability $PAV(t)$ is defined as the probability that the system is operational at time t. Formally, define an indicator random variable (representing the instantaneous availability) by

$$I(t) = \begin{cases} 1 & \text{if } X(t) \in S_O \\ 0 & \text{otherwise.} \end{cases}$$

  Then the point availability is

$$PAV(t) = P[I(t) = 1] = E[I(t)].$$

The point availability reflects the state of the system (operational or failed) at a given point in time, and for repairable systems it is not as useful as measures related to the amount of time the system is operational during a specified interval.

- Cumulative operational time.

  The random variable $O(t)$ is the total amount of operational time during $(0, t)$:

$$O(t) = \int_0^t I(s)ds.$$

  Its expected value and distribution are often of interest.

- Availability.

  The (interval) availability $A(t)$ is a random variable defined as the fraction of time the system is operational during $(0, t)$:

$$A(t) = \frac{O(t)}{t}.$$

  Note that

$$E[A(t)] = \frac{1}{t} E\left[\int_0^t I(s)ds\right] = \frac{1}{t}\int_0^t PAV(s)ds,$$

  and this formula can be used to calculate $E[A(t)]$. Another random variable of interest is the limiting or steady state availability, which is the fraction of operational time as $t \to \infty$, i.e. $\lim_{t\to\infty} A(t)$. In many cases, the name limiting availability or simply "availability" is given to $\lim_{t\to\infty} PAV(t)$, which is also equal to $\lim_{t\to\infty} E[A(t)]$.

Several measures can be defined in terms of the time of first system failure during $(0, t)$. In this case, all failed states in the model are considered as absorbing states. Analogous to $I(t)$, define a random variable $\tilde{I}(t)$ for this absorbing state model which simply indicates whether or not the system has failed by time t.

- Reliability.

  The reliability $R(t)$ is defined as the probability that the system is operational during the entire observation period. It is defined in a manner similar to the point availability as:

$$R(t) = P[\tilde{I}(t) = 1] = E[\tilde{I}(t)].$$

  It can also be defined in terms of the cumulative operational time as:

$$R(t) = 1 - \lim_{s\to t} P[O(t) \leq s].$$

- Lifetime.

  The lifetime $L(t)$ over an observation period is a random variable that is equal to the time of the first system failure, if such occurs before t, and is equal to t otherwise. That is, $L(t)$ can be thought of as the cumulative operational time for the model with all failed states made absorbing:

$$L(t) = \int_0^t \tilde{I}(s)ds.$$

  Note that the expected lifetime can be obtained from the reliability as:

$$E[L(t)] = \int_0^t R(s)ds.$$

- Mean time to failure.

  This measure is the limiting expected lifetime:

$$\text{MTTF} = \lim_{t \to \infty} E[L(t)].$$

Assume that rewards are associated with the states of $\mathcal{S}$, where the reward r_i is associated with state a_i (thus r_i is one of the possible system rewards ρ_l, $l = 1, \ldots, K+1$). The reward r_i may represent a performance measure when the system has the structure indicated by a_i. For instance, in the example of section 1, the reward r_0 for state a_0 is the throughput rate when two processors are operational, and the cumulative reward is the number of job completions during the observation period.

- Point performability

  Let $IR(t) = r_i$ if $X(t) = a_i$, so that this random variable is the instantaneous reward at time t. The point performability $PPF(t)$ is its expected value:

$$PPF(t) = E[IR(t)] = \sum_{i=1}^{M} r_i P[X(t) = a_i].$$

- Cumulative reward.

  The cumulative reward during $(0, t)$ is

$$CR(t) = \int_0^t IR(s)ds.$$

  The expected value of this random variable is given by:

$$E[CR(t)] = \int_0^t PPF(s)ds = \sum_{i=1}^{M} r_i \int_0^t P[X(s) = a_i]ds.$$

  The cumulative reward averaged over the length of the observation period is $ACR(t) = CR(t)/t$. Note that when only two rewards are assigned, $r_1 = 1$ for operational states and $r_0 = 0$ for failed states, then $CR(t)$ becomes the cumulative operational time and $ACR(t)$ the interval availability.

- Time to achieve a reward level.

  Another random variable of interest is $\Theta(r)$, the time to achieve a given level r of reward. Since $CR(t) > r$ is equivalent to $\Theta(r) < t$, the distribution of $\Theta(r)$ can be directly obtained from the distribution of the cumulative reward. As an example, the distribution of the time to finish a job that requires r units of work to complete is obtained from the distribution of cumulative reward. As another example, the distribution of the time to the first system failure is given by $P[L(t) > s] = R(s)$ (for $s < t$).

- Time above a performance measure.

  As before, associate a reward level with each state of $\mathcal{S}$. For instance, rewards may be assigned to states according to the capability of the system to execute different tasks, or according to throughputs, expected queue lengths, etc. Let $IR(t)$ be defined as above. Then, let

$$I(t) = \begin{cases} 1 & \text{if } IR(t) > r \\ 0 & \text{otherwise,} \end{cases}$$

  where r is a specified performance level. Define the total time above level r as:

$$O(t) = \int_0^t I(s)ds.$$

  The cumulative operational time is the case of 0,1 rewards and $0 < r < 1$.

In many cases it is important to evaluate the number of certain types of events during an observation period. For instance, if the chosen event is the failure of a certain component (e.g. the CPU) which causes the system to go down, then the measure to be obtained is the number of CPU failures that brought the system down during $(0, t)$. An obvious generalization of this measure is to associate rewards with transitions (pairs of states) instead of with states and then to obtain the total reward over the period.

- Number of events of a given type.

  For a pair of states (a_i, a_j) associate the reward r_{ij}. Let $N(t)$ be a random variable that gives the number of transitions occurring during $(0, t)$. Let τ_n be the time of the nth transition of the process X, and set $H(n) = r_{ij}$ if $X(\tau_n^-) = a_i$ and $X(\tau_n^+) = a_j$. Then the total reward due to transitions over $(0, t)$ is

$$TR(t) = \sum_{n=1}^{N(t)} H(n).$$

- Cost measures.

  Cost is an important measure to obtain, and it may be a function of different parameters of the model [14]. Two cost functions are of main importance: (a) the cost incurred in maintaining and repairing the system, and (b) the cost due to system unavailability per unit time. The first function is an example of assigning rewards to particular transitions in the model, e.g. a fixed cost is incurred each time a repair is performed. The second function can be viewed as a cumulative reward measure obtained by assigning rewards to certain states in the system, e.g. states that represent a down system.

- Other measures.

  Cumulative measures are related to the total amount of time spent in certain states between transitions of X during an observation period or to the total number of such transitions. However, there are other measures of interest in addition to these cumulative measures. One example, which is covered later in the paper, is the probability that two failures occur during a small period of time Δt, when the system may be vulnerable to a second failure while attempting to recover from the first failure. That is, a measure of interest is the probability of a "near coincident fault." If we let Y_l be the length of time between two consecutive faults, $l = 1, \ldots, NF(t)$, where $NF(t)$ is the number of times that two consecutive faults occurred, and we define the random variable $CF(t) = \min(Y_l)$, then the measure of interest is $P[CF(t) \leq \Delta t]$.

3 Model Specification

3.1 Introduction

In the previous section several measures of interest to the analyst were defined. The measures may reflect the effect of contention for resources and/or structural changes in the system. To evaluate the measures, two major questions have to be answered: (a) How does one specify the model? (b) How does one solve for the measures of interest? The first question is addressed in this section, while the second question is considered in section 4.

In a modeling tool it is desirable from the user point of view that the mathematical definition of the system being modeled and the details concerning the solution techniques be hidden. Yet the tool should be sophisticated enough to allow the system specification of the model and the measures in a manner as close as possible to the "natural" system definition, i.e. to the way the user "thinks" of his system. Many tools have been developed over the past few years, e.g. ARIES-82 [59], SURF [15], CARE III [3], HARP [33, 93], SAVE [34], SHARPE [87], METFAC [8], GreatSPN [10], METASAN [71, 88], TANGRAM [5, 80], SPNP [12], DyQNtool [47]. Some allow only specification similar to the mathematical definition of the system (e.g. Markov chain), while a major concern of others is in developing a "natural" definition language. Modeling capability, the type of measures that can be obtained and the solution techniques employed also serve to differentiate the many tools. Johnson and Malek [53] present a comprehensive survey of many existing tools which includes a discussion of the user interface and the measures that are obtainable.

Several important issues are involved in the development of a tool. Of these, the user interface is one of the main issues. The interface should be "friendly," "high level" and

tailored to the problem the user wishes to solve. For instance, in the example of section 1, it is desirable for the user to be able to describe the system in terms of processor modules and their queues, the dispatcher and its policy, etc., and/or the components that can fail, the repair policy, etc. The underlying Markov chain (the mathematical model) should be hidden below the high level interface. An example of a high level interface tailored to availability models is the language in the SAVE tool.

In addition to a high level interface language, the power of a tool also depends on the flexibility of adding new features and the ease with which it can be tailored to specific problems. For instance, although the interface of the SAVE tool allows the specification of many common availability models, the basic constructs in SAVE are not of sufficient generality. New features in the language may not be modeled using existing constructs, and the tool does not allow the user to define additional constructs. Furthermore, the high level user interface is tailored to availability modeling, and performance models cannot be specified.

Another issue of concern in the design of a modeling tool is the choice of a proper solution method, e.g. should a bounding technique be employed in determining steady state availability? Consider again the example of section 1. Depending on the user specified value for the length of the observation period, decomposition techniques may not be of use, e.g. the period of interest may be too short.

How to obtain a user specified measure from a "basic" model solution also constitutes an important issue to be addressed, and the answer may be nontrivial, depending on the problem. Consider the example of section 1, and suppose that the user, after specifying the system, asks for its steady state throughput. Although this seems like a simple request, satisfying it requires some effort. If decomposition is not used, the steady state solution of the Markov chain of Figure 2 can provide the answer. However, the tool needs to "understand" the meaning of each state as well as the output rates, in order to calculate the throughput from the steady state probabilities. If decomposition is used, performance models have to be independently solved to obtain reward rates. Then, rates must be properly assigned to each of the structural model states, which also may require knowledge of the meaning of each state, e.g. in Figure 3, state F (O) represents a system with one processor (two processors) working. For a model with thousands of states, such a task is nontrivial.

We begin this section by first presenting examples of model specification, which serves to briefly survey some of the existing modeling tools and to further illustrate the issues mentioned above. Then we discuss a modeling paradigm recently proposed for model specification and present some examples of its use.

3.2 Examples of Model Specification

A model specification language that is frequently used in performability modeling to generate the underlying Markov chain is that of *stochastic Petri nets* (SPN). The use of stochastic Petri nets for modeling specification was proposed independently by Molloy [73] and Natkin [77] and has subsequently seen widespread application [1]. A stochastic Petri net is based on the notion of a Petri net [82]. Briefly, a Petri net model consists of *places* and *transitions* connected via a set of directed arcs. Places may contain *tokens* which move through the network (i.e. from place to place) according to certain rules. One basic rule dictates that a transition can *fire* if all of its input places contain one or more tokens. When a transition fires, it removes tokens from its input places and adds a token to each of its output places. The current state of the model is given by the number of tokens in each place and is called a *marking*. Figure 4 depicts a Petri net model, where places are represented by circles, transitions are represented by bars and tokens are represented by dots.

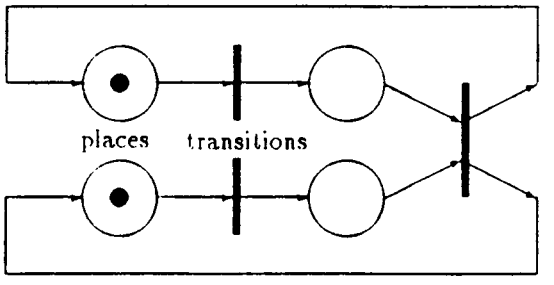

Figure 4: A Petri net model.

Various extensions to Petri net models have been proposed [1, 82]. One such extension is the notion of *inhibitor* arcs. An inhibitor arc from a place to a transition indicates that the transition is enabled when no tokens are present in the input place. Another extension is that of multiple arcs from an input place to a transition. Equivalently, one may define *counter arcs*, so that an integer k is associated with a counter arc to indicate that the transition is enabled when at least k tokens are present in the input place.

In stochastic Petri net models, an exponentially distributed firing time is associated with each transition. The firing rule is as follows: once a transition is enabled, an exponentially distributed amount of time elapses at the end of which the transition fires if it is still enabled. It was shown in [73] that an SPN model is equivalent to a continuous time Markov chain, the state space of which is the reachability set of the SPN (i.e. the set of markings reachable from an initial marking). Therefore, a Markov chain can be built and solved from a given

SPN model.

Several extensions have been proposed to increase the modeling power of SPNs. One of these extensions allows the model to have transitions that fire in zero time (called *immediate* transitions) in addition to exponentially distributed transitions. Models that include such transitions are called *generalized stochastic Petri nets* (GSPN) and were proposed by Ajmone Marsan *et al.* [2]. In a GSPN model, some of the states (called *vanishing* states) have zero holding times, and it is easy to show that an equivalent Markov chain with no such states can be built. Immediate transitions can be used to represent events with rates that are orders of magnitude higher than those of other events in the model. By representing "fast events" with immediate transitions, the state space of the associated Markov chain can be significantly reduced. Immediate transitions are also useful in representing "logical" structures in the model.

Another set of extensions was proposed by Dugan *et al.* [31], and the associated models are called *extended stochastic Petri nets* (ESPN). An ESPN model allows the firing times of certain transitions to be generally distributed. The embedded chain defined at points where marking changes occur must exhibit the Markov property, and there are restrictions concerning which transition may be allowed to have a general distribution. For instance, a transition can have a generally distributed firing time if, when it is enabled, no other transition is also enabled. However, if there is a marking that simultaneously enables two transitions and the firing of one of these does not disable the other, then these transitions must have exponentially distributed firing times. Other extensions involve modifying the transition firing rules. For example, a *probabilistic* arc from a transition to a set of places allows a probabilistic choice of the output places that receive a token after the transition fires.

Several tools utilize SPN (and its extensions) as part of the model specification language. Among these we mention HARP [93], SPNP [12], GreatSPN [10] and METASAN [88]. Some of the main advantages of SPN models are: (a) modeling power that is equivalent to Markov chains; (b) a graphical representation of the model; and (c) "general" language constructs (e.g. places, transitions), which permit new features to be incorporated in the model without requiring changes in the language. However, these models also have disadvantages. Perhaps the main disadvantage is that the basic SPN constructs are quite primitive, so that not only is a significant burden placed on the analyst in order to specify complex models, but in addition the graphical representation may become too complex to be useful. Another disadvantage is that the representation of priorities or ordering is hard to manage, although priority queues are important in performability modeling (e.g. a priority repair queue or queues for resources such as CPUs).

SPN interfaces are flexible, but they cannot be easily tailored to particular application domains. However, as mentioned in the introduction, a desirable property of an interface

is the ability of allowing users to specify the system in terms of how they think of it, i.e. in terms of the components of the particular application and the interactions among them. The SAVE tool provides an example of such a specialized interface.

Availability models are described in SAVE by specifying its components and the repair strategy that is used when a unit fails. All components are described using the same basic constructs, and they differ only in the parameters specified by the user. Basically, a SAVE specification of a set of component units of the same type (i.e. with the same set of parameters) includes: the failure rate of units in the set; the number of spare units; "failure mode" probabilities; for each failure mode, the repair rate of units that failed in that mode; the (possibly empty) list of components affected when a unit fails in each of the failure modes; repair dependencies; operational dependencies. The behavior of a generic component is shown in Figure 5, which is taken from [34]. As is apparent from this figure, an operational unit that fails or is affected by other components goes to a down state. When an operational unit goes to a down state, a spare unit (if there is any left) immediately replaces the failed unit. A component may be in a *dormant* state if its operation depends upon components that are not working.

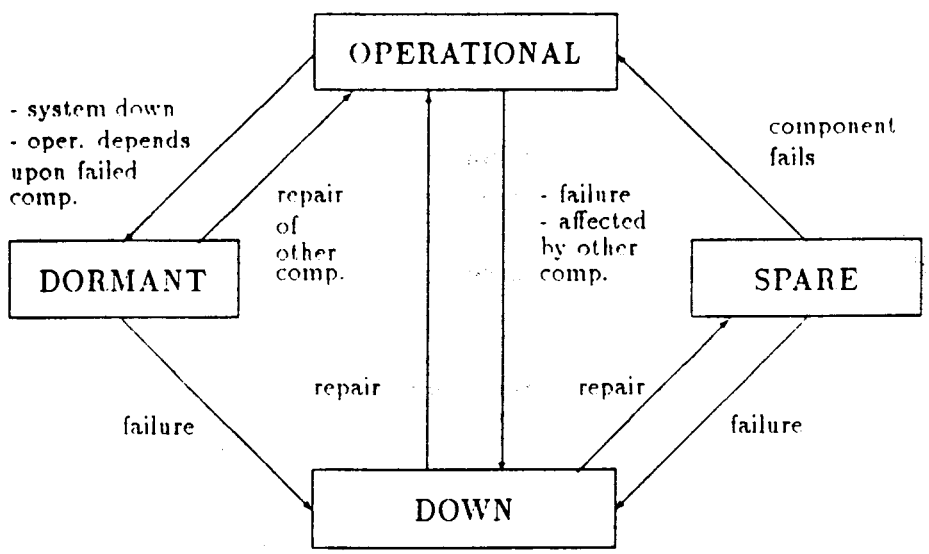

Figure 5: The behavior of a SAVE generic component.

Once the model is specified, the associated Markov chain is generated. An important advantage of this type of interface is the high level language tailored to availability modeling. Unfortunately, the language is not "general" in the sense that new features may require a change in the language constructs. Examples of features that cannot be obtained from the basic language constructs include: the modeling of "components affected" constructs that

differ from those that are provided; the modeling of resource contention other than the repair queue; the modeling of certain fault detection/recover mechanisms.

Movaghar and Meyer [74] and subsequently Meyer *et al.* [71] developed an SPN-based modeling paradigm called *stochastic activity networks* (SAN). SAN includes several extensions to SPN, some of which are similar to those presented above (GSPN and ESPN), but which were independently developed. Similar to Petri nets, SAN primitives include places and transitions (called activities). New primitives include input and output *gates*, which are used to describe how an activity is enabled and how the activity affects the next marking after it finishes. In more detail, corresponding to each gate is a rule that describes the conditions under which a transition associated with the gate is enabled. There is also a rule that describes the resulting marking at the places connected by the gate after the transition completes. Since its inception, the developers of SAN have addressed issues concerning performability evaluation such as the development of models that capture the performance of a system with varying structure. The method proposed in [71] follows the classical decomposition approach. A model that captures the overall behavior (performance and varying structure) is developed, and then a performance submodel and a structural submodel are identified. Reward rates are calculated by solving performance submodels with different initial markings obtained from the structural submodel. Performability measures are then determined from these rates and the structural submodel. It is interesting to note the attention given in developing a paradigm to obtain rewards and associate them with the structural model from a high level (Petri net) system description.

Later in [70] Meyer developed a methodology for specifying performability measures directly from a high level system description (a SAN model). By working jointly with the performance and structure submodels, reward structures are obtained that allow the determination of rewards for each structural configuration of the system. The reward structures are defined in terms of functions of SAN primitives.

Haverkort and Niemegeers [44, 45, 46] also address the issue of obtaining a Markov reward model from a high level system description. The approach, called the *dynamic queueing network* concept, follows the classical decomposition technique. First, a parameterized queueing network is built with a set of parameters $\mathcal{P}$ which are not yet assigned. Each of the queueing networks is to model the performance behavior under given structures. Second, a GSPN model that represents the structural changes in the system is built, and a function Φ of the possible markings of the GSPN is specified. This function returns a set of values to $\mathcal{P}$ for each possible marking. Then, for each such set, a queueing network is solved and the performance results are mapped as rewards to the corresponding GSPN marking or, equivalently, to the corresponding state of the final Markov reward model. In the example of section 1, the values of $\mathcal{P}$ would determine the scheduling policies of the queueing network of Figure 1: if only one CPU is operating, then the scheduler would send jobs only to the working CPU. More complex interdependencies between the performance and availability

models can also be specified. For example, GSPN transition rates can be made dependent on the queueing network model solution. This is useful in modeling failure rates that depend on the system load. Furthermore, structural dependencies on performance can also be modeled. In this case, the generation of the reachability set is done in conjunction with the solution of the performance models. That is, once the performance associated with a given marking is solved, the GSPN parameters are updated and the next marking reflects these new changes. These dependencies are useful in modeling certain failures that are related to the system load. Note, however, that these are dependencies on average load. To our knowledge, dependencies on peak load, for example, cannot be modeled.

As noted previously, many model specification languages for performability evaluation are based on SPN, and thus they possess the disadvantage of relatively "low level" constructs (e.g. places, transitions). The SAVE tool provides a convenient high level interface, but it lacks the flexibility of incorporating new constructs. METFAC [8] is an example of a tool which attempts to achieve a relatively high level of specification and yet preserve generality. Its proposed methodology is based on *production rules*. Basically, the global state is specified, and then rules are given that specify the conditions under which an action is taken that results in the changing of (global) state variables. One disadvantage of this approach is that the rules operate in the global system state variables.

Berson *et al.* [5] propose a novel paradigm for model specification, which has been implemented in the TANGRAM tool [80] developed at UCLA. Informally, the system to be modeled is viewed as a collection of *objects* which interact by exchanging *messages*. Each object has an internal state which can evolve over time. Changes in the internal state may be due to: (a) the occurrence of an *event*; and (b) the arrival of a *message*. Events are generated internally to an object with a given rate, and the occurrence of an event causes a set of *actions* to be taken. Actions are preconditioned on the state of the object and may cause a change in the object state and the sending of messages to other objects. Such actions are also taken when an object receives a message. Actions are executed in zero time, and the delivery of messages also occurs instantaneously. The system state is the set of internal states of the objects and the list of messages not yet delivered. Therefore, two sets of states are identified: (a) *tangible* states, which have positive holding time, i.e. no messages to be delivered; and (b) *vanishing* states, which have messages yet to be delivered and so have zero holding time. For analysis purposes, only tangible states need to be considered (analogous to GSPN).

The notion of object types is used in order to facilitate the specification of models. Object types are parameterized definitions, and an object is simply an instance of an object type, i.e. every object in the model is declared to be of a certain type, and parameters are specified for each. This modeling paradigm is quite flexible, since object types can be constructed for different application domains, and a higher level interface tailored to the needs of the user can be built from the types appropriate to the particular domain. A tool based on

this modeling paradigm can be enhanced simply by the addition of new object types. Thus, both the flexibility of SPN description tools and the convenience of tailored interfaces are obtained. The language chosen to implement the object oriented paradigm is Prolog. The main advantages include the possibility of using untyped data structures, which give the user freedom in the specification of object states, and the powerful pattern matching feature of Prolog which is used in the precondition of rules.

Examples that illustrate the power of the modeling paradigm are presented in section A of the appendix. In the examples, an object type that behaves like a SAVE generic component is defined, and an object type priority queue is also described. From these objects, structural models and performance models can easily be built using the same paradigm. In [64] extensions were made which include the ability to create and destroy objects dynamically, to prioritize execution among objects, and other additional features. These extensions are used by an interface which was developed to allow an Estelle [49] specification of a communication protocol to be automatically translated to the object oriented paradigm for analysis purposes. This is another example that demonstrates the power of the model description paradigm.

In this section the main issues concerning the specification of performability models have been addressed and existing tools that illustrate solutions to some of the problems have been briefly described. The next section concentrates on solution techniques developed for Markov performability models.

4 Model Solution

4.1 Introduction

In the previous section, our interest was in issues related to the specification of performability models via a high level description language. The (low level) mathematical description that represents the model (in our case the Markov transition rate matrix) is generated from the specification. In this section we describe several solution techniques which have been used to solve performability models. Although model specification and model solution are treated in separate sections of the paper, these two subjects interact naturally during the modeling and analysis process. For example, the state space explosion problem that we treat below in section 4.4 is of importance, not only in obtaining numerical solutions, but also in the generation of states as part of the specification process. Thus the two are inherently tied together.

As in the specification part of the paper, instead of presenting an exhaustive survey of so-

lution methods for dependability and performability models of computer systems, we will try to give the reader a brief overview of the approaches that have been proposed and the models for which they have been used. In particular, we will spend much of the section discussing the application of the uniformization or randomization technique to dependability and performability modeling. Other well-known methods of solution, such as transform methods, will receive less attention. However, the many references that are included should lead the interested reader to more details on these other approaches and models. As an example, a survey on performability analysis which concentrates on Laplace transform methods can be found in [50]. We will discuss both Markov and semi-Markov models of system behavior, with the emphasis on the former type of model.

Basically, the various solution techniques can be subdivided into those developed to calculate specific measures related to the time until failure, those applicable only to models of systems that cannot undergo repair during the observation period and those applicable to general repairable systems. In the first case, both repairable and nonrepairable system models may be considered, and the corresponding Markov chain contains absorbing states that are usually related to system failure. In the second case, the corresponding Markov chain is acyclic, i.e. no state is visited twice for any of the possible sample paths. The third case involves the solution of "general" chains that represent systems with repairable components. The techniques employed may take advantage of the particular Markov chain structure.

We begin this section with a brief overview of some of the techniques used in problems involving the classes of chains mentioned above, with an emphasis on transform approaches. As we will see, most Laplace transform methods which have been used to determine performability measures of repairable and nonrepairable systems utilize recursive equations obtained by conditioning on the time of the first transition of the Markov model. Next, a solution methodology based on the uniformization or randomization technique is described in detail, and it is shown that many performability measures can be calculated using a unified framework, i.e. by marking or coloring certain subintervals of the observation period based on the underlying stochastic process and then associating rewards to these intervals. Finally, we review some techniques that have been proposed in the context of the state space explosion problem.

4.2 A Brief Overview of Solution Techniques

The work of Beaudry [4] is an example of one of the earliest methods developed to calculate performability measures of Markov chains with absorbing states. Performability measures calculated include the probability of executing a task of a given length before system failure (equivalently, the distribution of accumulated reward until failure) and the expected accu-

mulated reward. The basic idea is to transform the original Markov chain with rewards into an equivalent Markov chain for which all reward rates of nonabsorbing states are equal to 1, while ensuring that the accumulated reward in each state for both models is identical. The performability measures of interest are then obtained from the reliability and the mean time to failure of the second chain. That is, an exponential sojourn time time τ in state a_i of the original chain is increased (if $r_i > 1$) or decreased (if $0 < r_i < 1$) to compensate for the unit reward of the second model. The transformation can be accomplished by dividing any rate from a state a_i to a state a_j, say q_{ij}, by r_i. This procedure applies directly for the case of positive rewards, but nonpositive rewards can be easily handled when the expected accumulated reward is the measure of interest. A constant is added to transform the problem to the positive reward case, and then the constant is subtracted back out of the final result. However, dependencies between the accumulated reward and the time until absorption cause this "trick" to be unsuccessful when the measure of interest is the distribution of accumulated reward. Recently, Ciardo *et al.* [11] extended the approach of Beaudry to handle the case of zero reward rates in determining the distribution. States with zero rewards are transformed into vanishing states in the equivalent model, similar to the GSPN case. Extensions to semi-Markov models are also included in [11].

We now briefly discuss several solution techniques for nonrepairable systems. An important characteristic of such models is that the system states may be labeled in a manner so that the probability of a transition from state a_i to state a_j is zero when $i < j$. Thus, the generator $\mathbf{Q}$ of the Markov process that describes the behavior of the system is lower triangular. Several papers have appeared that address the issue of performability evaluation for such models, and the acyclic structure has been exploited to obtain recursive solutions. Meyer [69] considered a model of a nonrepairable multiprocessor system with N processors and a fixed number of buffers to store submitted tasks. Each processor may fail, thus causing the system to degrade. A fault in a buffer or a nonrecoverable fault in a processor causes the system to fail. The state transition diagram of the Markov chain that models the structural changes in the system is given in Figure 6, where the state is the number of operating processors. The measure of interest is the distribution of the fraction of arrived tasks processed in

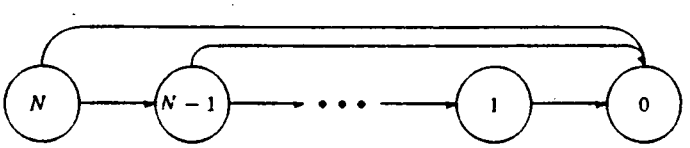

Figure 6: Markov chain structure model for the processor-buffer example.

an observation interval (let Y be the random variable that represents this fraction of tasks). After the rewards that are assigned to the states of the system are calculated, the solution is obtained by calculating the joint density of residence times in each state, conditioned

on the imbedded Markov chain at state transition times. This procedure is easy to carry out because of the special structure of the chain (see Figure 6). From the conditional joint density of residence times, the conditional distribution of Y is obtained by integrating over the set for which Y is below a given level of performance. The final solution is obtained by unconditioning, which requires determining the probability of the possible sample paths. In Meyer [69] this procedure was used for a system with two processors. Later, Furchtgott and Meyer [32] extended the technique to the case of general acyclic chains (i.e. multidimensional acyclic chains). The approach works for rewards that are monotonic in the states. The technique is applicable to semi-Markov processes, but its computational requirements are exponential in the number of states.

Donatiello and Iyer [30] also studied the model of Figure 6, i.e. single dimensional chains. They first obtained a recursive expression for calculating the cumulative reward by conditioning on the time τ of the first transition out of state N. The resulting equation relates the performability measure during $(0, t)$ with the measure during $(t - \tau, t)$ assuming that the system starts in a state reachable from N. By transforming the recursive expression (i.e. taking the double Laplace transform in the performability level and time variables), performing a partial fraction expansion and then inverting the result, a closed form expression for the cumulative reward is obtained (coefficients of the expression are found recursively). The cost of calculating the distribution of cumulative reward is shown to be $O(N^3)$. In [30] it is also shown that the approach can be extended to the case of general acyclic chains (see also Grassi *et al.* [37]). In this case the computational complexity is $O(M^3 d)$, where M is the number of states and d is the maximum number of nonzero elements in a row of the transition rate matrix. Ciciani and Grassi [13] solved the same model, but they work with the general equation (5) below.

Goyal and Tantawi [36] studied general acyclic chains with monotonic rewards and obtained a recursive equation by conditioning on the time of first failure. Unlike previous work, their recursion is carried out in the time domain. The resulting algorithm is polynomial in the number of states of the model. Note that the approach of [13, 30, 37] is not limited to monotonic rewards. Other results related to acyclic chains can be found in [60, 86].

Determining performability measures for repairable systems is a more difficult problem. In such systems, components can fail and then be repaired, so that the corresponding Markov chain (or semi-Markov process) is not acyclic. Thus the techniques outlined above are not directly applicable. However, as in the case of acyclic chains, transform methods have been applied by several authors to obtain performability measures. In order to illustrate the basic idea of such transform methods, consider a Markov chain model where a reward rate r_i is associated with state a_i. Let λ_i be the total rate out of state a_i, and let p_{ij} be the probability of a transition from a_i to a_j. Let $F_i(y, t)$ be the distribution function of the cumulative reward up to time t given that the chain started in state a_i, i.e. $F_i(y, t) = P[CR(t) \leq y | X(0) = a_i]$. An expression for the double Laplace transform of $F_i(y, t)$ is obtained by conditioning on τ,

the time of the first transition. We have

$$F_i(y,t) = u(y - r_i t)e^{-\lambda_i t} + \int_{\tau=0}^{t} \lambda_i e^{-\lambda_i \tau} \sum_{j=1}^{M} p_{ij} F_j(y - r_i\tau, t-\tau)d\tau. \quad (1)$$

where $u(x)$ is the unit step function. Equation (1) is the basic expression that is used in many transform approaches, including those for acyclic chains. Taking the Laplace-Stieltjes transform of $F_i(y,t)$ in the y variable gives

$$h_i^*(s,t) = e^{-(\lambda_i + r_i s)t} + \lambda_i \sum_{j=1}^{M} p_{ij} \int_{\tau=0}^{t} e^{-(\lambda_i + r_i s)\tau} h_j^*(s, t-\tau)d\tau. \quad (2)$$

Now, taking the Laplace transform in the t variable, we have

$$h_i^{**}(s,\delta) = \frac{1}{\lambda_i + \delta + r_i s} + \frac{\lambda_i}{\lambda_i + \delta + r_i s} \sum_{j=1}^{M} p_{ij} h_j^{**}(s,\delta). \quad (3)$$

Equation (3) can be written in matrix form as

$$\mathbf{h}^{**}(s,\delta) = \mathbf{w}(s,\delta) + \mathrm{diag}\{\mathbf{w}(s,\delta)\}\mathrm{diag}\{\lambda\}\mathbf{P}\,\mathbf{h}^{**}(s,\delta), \quad (4)$$

where all of the above vectors are column vectors. Here $\mathbf{w}(s,\delta) = \langle 1/(\lambda_i + \delta + r_i s)\rangle^{\mathrm{T}}$, $\lambda = \langle \lambda_i \rangle^{\mathrm{T}}$, and $\mathrm{diag}\{\mathbf{x}\}$ is a diagonal matrix with the vector $\mathbf{x}$ as its diagonal. Rewriting (4) gives

$$\mathbf{h}^{**}(s,\delta) = [\mathbf{I} - \mathrm{diag}\{\mathbf{w}(s,\delta)\}\mathrm{diag}\{\lambda\}\mathbf{P}]^{-1}\mathbf{w}(s,\delta).$$

Noting that $\mathbf{P} = \mathbf{I} + (\mathrm{diag}\{\lambda\})^{-1}\mathbf{Q}$, we have

$$\begin{aligned}\mathbf{h}^{**}(s,\delta) &= [\mathbf{I} - \mathrm{diag}\{\mathbf{w}(s,\delta)\}\mathrm{diag}\{\lambda\} - \mathrm{diag}\{\mathbf{w}(s,\delta)\}\mathbf{Q}]^{-1}\mathbf{w}(s,\delta) \\ &= [\mathrm{diag}\{\mathbf{w}(s,\delta)\}(\delta\mathbf{I} + s\,\mathrm{diag}\{\mathbf{r}\}) - \mathrm{diag}\{\mathbf{w}(s,\delta)\}\mathbf{Q}]^{-1}\mathbf{w}(s,\delta)\end{aligned}$$

where $\mathbf{r} = \langle r_i \rangle^{\mathrm{T}}$. Since $(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1}$, and defining $\mathbf{1} = \langle 1, \ldots, 1\rangle^{\mathrm{T}}$, we finally have

$$\mathbf{h}^{**}(s,\delta) = [\delta\mathbf{I} + s\,\mathrm{diag}\{\mathbf{r}\} - \mathbf{Q}]^{-1}\mathbf{1}. \quad (5)$$

Equation (5) was obtained by Puri [83] (see also Kulkarni *et al.* [56]). In [56] $h_i^{**}(s,\delta)$ is expanded in partial fractions, and the result is analytically inverted with respect to δ. The final solution is then computed by numerically inverting the resulting transform in the s variable. Later Smith *et al.* [91] improved the computational time of the algorithm. Kulkarni *et al.* [57] used the duality between the cumulative reward and the time to achieve a reward level mentioned in section 2 to derive equation (5) (see also Nicola *et al.* [79]).

Iyer *et al.* [51] noticed that equation (5) can be directly inverted in the δ variable to obtain

$$\mathbf{L}^*(s,t) = \left[e^{\mathbf{Q}t - s\,\mathrm{diag}\{\mathbf{r}\}t}\right]\mathbf{1},$$

and standard numerical techniques can then be used to invert the above transform in the remaining variable. In [51] a double Laplace transform expression for semi-Markov models was also found. The equation that was obtained is similar to equation (5), and its derivation follows along the same lines as the one above. For Markovian models, a recursion between the $(n+1)$st and nth moments of the cumulative reward was derived in [51]. The derivation involved rearranging equation (5), differentiating $(n+1)$ times with respect to s and setting $s = 0$. The result is a simple Laplace transform expression relating the $(n+1)$st and nth moments. Noting that $e^{\mathbf{Q}t}$ is the inverse of the transform $[\delta\mathbf{I} - \mathbf{Q}]^{-1}$, the Laplace transform expression can be inverted, and the final result is obtained from the spectral representation of $e^{\mathbf{Q}t}$. Recently, Pattipati and Shah [81] considered the case of a nonhomogeneous Markov chain and showed how to calculate moments of cumulative reward using numerical integration techniques.

In the next section we concentrate on a detailed description of the uniformization or randomization technique, which has been found to be useful in calculating various performability measures.

4.3 Calculating Performability Measures using Uniformization

We now turn to the discussion of another method that can be used to calculate performability measures, namely, the uniformization or randomization technique. This approach proceeds by replacing the continuous time Markov process that represents system behavior by an equivalent process, that of a discrete time Markov chain subordinated to a Poisson process. The technique can be used to calculate both transient and steady state solutions. In [23, 24] a methodology for calculating performability measures for repairable systems based on uniformization was presented. In this section we discuss that approach in detail.

We begin by reviewing the uniformization technique, which was introduced by Jensen [52] and is covered in many books on stochastic processes (see, for example, Çinlar [9], Heyman and Sobel [48], Keilson [54] and Ross [85]). Recall that the behavior of the system under study is modeled by a homogeneous continuous time Markov process $\mathcal{X} = \{X(t) : t \geq 0\}$ with generator $\mathbf{Q}$ defined on a finite state space $S = \{a_i : i = 1, \ldots, M\}$. The $M \times M$ generator matrix $\mathbf{Q}$ has the form

$$\mathbf{Q} = \begin{bmatrix} -q_1 & q_{12} & \cdots & q_{1M} \\ q_{21} & -q_2 & \cdots & q_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ q_{M1} & q_{M2} & \cdots & -q_M \end{bmatrix}.$$

For $i \neq j$, the (i,j) entry represents the (exponential) rate at which a transition occurs from state a_i to state a_j, while the negative of the ith diagonal entry satisfies $q_i = \sum_{j \neq i} q_{ij}$ and

represents the rate out of state a_i. Since the state space is finite, the rates q_i are uniformly bounded, and we let $\Lambda \geq \max\{q_i\}$. It is possible to think of X as a Markov process for which the rate out of each state is the same parameter Λ by using the following construction. For the ith state simply add fictitious transitions from a_i back to itself, so that with probability q_{ij}/Λ a transition to a_j occurs and with probability $1 - (q_i/\Lambda)$ a fictitious self-transition takes place. This construction preserves the basic structure of X, while the transition rates for all states are now identical. Thus the Markov process X can be thought of as a discrete time Markov chain subordinated to a Poisson process in the following sense. Consider the discrete time Markov chain $\mathcal{Z} = \{Z_n : n = 0, 1, \ldots\}$ with state space $\mathcal{S}$ and with transition matrix $\mathbf{P} = \mathbf{Q}/\Lambda + \mathbf{I}$, and let $\mathcal{N} = \{N(t) : t \geq 0\}$ be a Poisson process with rate Λ that is independent of $\mathcal{Z}$. Then the above discussion shows that we may interpret $X(t) = Z_{N(t)}$ for $t \geq 0$. That is, the transition times of X occur according to the Poisson process $\mathcal{N}$, while the transitions themselves are governed by the discrete time chain $\mathcal{Z}$ according to the transition matrix $\mathbf{P}$.

Using this form for the Markov process X, numerically stable algorithms for the calculation of transient (and steady state) distributions of various measures can be derived, an approach advocated by Grassmann in [38, 39]. For example, consider the calculation of the transient state probability distribution $P(t) = \langle P_1(t), \ldots, P_M(t) \rangle$, where $P_i(t)$ represents the probability that X is in state a_i at time t under the initial distribution $P(0)$, i.e. $P(t) = P[X(t) = a_i]$. Then, conditioning on the number of (Poisson) transitions n in a period of length t, it is easy to see that

$$P(t) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \pi(n), \tag{6}$$

where $\pi(n) = \langle \pi_1(n), \ldots, \pi_M(n) \rangle$, and $\pi_i(n)$ is the probability that the chain $\mathcal{Z}$ is in the ith state at time n, i.e. $\pi_i(n) = P[Z_n = a_i]$. The $\pi(n)$ satisfy the recursion $\pi(n+1) = \pi(n)\mathbf{P} = \pi(0)\mathbf{P}^n$, with initial distribution $\pi(0) = P(0)$. Note that, although the infinite series in the expression for $P(t)$ must be truncated during calculation, error bounds are readily computable from properties of the Poisson distribution, and in fact calculations may be performed within a prespecified error tolerance.

Uniformization can also be used to calculate performability measures of Markov reward models. We assume there are $K+1$ rewards $\rho_1 > \cdots > \rho_{K+1}$ which may be associated with states or transitions, and without loss of generality we suppose that $\rho_1 = 1$ and $\rho_{K+1} = 0$ (otherwise replace ρ_i by $(\rho_i - \rho_{K+1})/(\rho_1 - \rho_{K+1})$). Recall that $PPF(t)$, the point performability, is the expected instantaneous reward at time t. Thus it is simply

$$PPF(t) = \sum_{i=1}^{M} r_i P_i(t) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{i=1}^{M} r_i \pi_i(n) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \{\mathbf{r} \cdot \pi(n)\}, \tag{7}$$

where r_i is the reward associated with state a_i (and thus is one of the ρ_l, $l = 1, \ldots, K+1$). Both the point availability and the reliability at time t can be calculated as special cases

of the above formula. Using similar arguments, other simple performability measures have been easily calculated using uniformization. These include mean time to failure, expected availability and expected cumulative reward (see [40], [43], [92], [65], [72]).

As an example, the random variable $ACR(t) = CR(t)/t$ represents the total accumulated reward averaged over t. Its expected value can be expressed in terms of the integral of the point performability over $(0, t)$. That is,

$$E[ACR(t)] = \frac{1}{t}\int_0^t \left[\sum_{n=0}^{\infty} e^{-\Lambda s}\frac{(\Lambda s)^n}{n!}\{\mathbf{r}\cdot\pi(n)\}\right] ds.$$

Performing the integration yields

$$E[ACR(t)] = \sum_{n=0}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\left[\frac{\sum_{j=0}^{n}\{\mathbf{r}\cdot\pi(j)\}}{n+1}\right]. \tag{8}$$

Note that this equation can be written in the form

$$E[ACR(t)] = \sum_{n=0}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!} f(n)$$

where

$$f(n+1) = \frac{n+1}{n+2} f(n) + \frac{\mathbf{r}\cdot\pi(n+1)}{n+2},$$

and thus it can be easily evaluated in a recursive manner. Furthermore, $f(n) \leq \max\{r_i\} = 1$ for all n, so that the nth term of the infinite series in equation (8) is bounded by the corresponding term of the Poisson distribution. Similar recursions may be found for other performability measures considered below. An equivalent expression for the expectation is

$$E[ACR(t)] = \frac{1}{\Lambda t}\sum_{n=0}^{\infty} E_{n+1,\Lambda}(t)\{\mathbf{r}\cdot\pi(j)\} \tag{9}$$

where $E_{n+1,\Lambda}(t) = 1 - \sum_{j=0}^{n} e^{-\Lambda t}(\Lambda t)^j/j!$ is the $n+1$-stage Erlangian distribution.

In order to actually use (8) or (9) to calculate the expected cumulative reward averaged over the period $(0, t)$, an infinite series must be truncated after a finite number of steps, say N. The amount of error introduced due to this truncation when using (8), for example, is then

$$e_R(N) = \sum_{n=N+1}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\left[\frac{\sum_{j=0}^{n}\{\mathbf{r}\cdot\pi(j)\}}{n+1}\right] \leq \sum_{n=N+1}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}.$$

In [40] it is shown that the error from (9) is even smaller than $e_R(N)$, assuming the same truncation value N. Thus calculations can be done to within a predetermined error tolerance ϵ by using properties of the Poisson distribution [38]. This type of behavior is common for

the measures we calculate using uniformization, and computations can be done so that the total error can be easily bounded. Also note that the major computational effort in the calculation of $E[\mathcal{X}'R(t)]$ only involves the uniformized chain $\tilde{\mathcal{Z}}$. Thus several different observation periods $(0, s)$ can be considered, since only the Poisson terms change.

We now present a methodology for calculating transient performability measures using uniformization over a finite observation period $(0, t)$. Transitions of the uniformized chain split the time period of interest into intervals during which the process $\mathcal{X}$ remains in a particular state. The various performability measures considered depend on certain states and/or transitions, and the methodology we introduce is based on "coloring" intervals corresponding to these states or transitions. Performance measures that can be calculated include distribution functions (PDF), density functions (pdf), and probability mass functions (pmf). Simpler performability measures such as expectations can be calculated as a byproduct of the method of solution. This approach was first introduced in [23] for certain availability measures (rewards of 0 or 1) and then extended to more general reward functions in [24]. Besides numerical stability and the ability to specify error tolerances in advance as discussed above, other advantages of the methodology include the physical interpretation of the various random variables studied along with the simplicity of implementation of the numerical algorithms.

We now consider the problem of calculating various quantities in a performability environment, that is, under general rewards. Our interest is in measures such as the *distribution* of accumulated reward during a finite observation period. Transitions during $(0, t)$ split the period into intervals that have rewards or colors associated to them, perhaps based on the state of the system during the interval or based on the transition at the start of the interval. Intervals with identical rewards are assigned the same color, and uniformization is used to calculate the resulting measures which are defined in terms of the colors.

Recall that there are $K + 1$ different rewards $1 = \rho_1 > \cdots > \rho_{K+1} = 0$ associated with states or transitions (pairs of states) of the Markov process $\mathcal{X}$. Let $M(t)$ be the measure of interest that we wish to calculate. If n transitions of $\mathcal{X}$ occur during $(0, t)$, the observation period is divided into $n + 1$ intervals. Assume that there are k_l intervals of color l (with reward ρ_l) for $l = 1, \ldots, K + 1$. Define the vector $\mathbf{k} = (k_1, \ldots, k_{K+1})$ which indicates the number of intervals corresponding to each reward, and recall that $\|\mathbf{k}\| = k_1 + \cdots + k_{K+1}$. We will frequently refer to a "coloring" $\mathbf{k}$. Note that there are $\binom{K+n+1}{n+1}$ ways of assigning colors to the intervals (i.e. such that $\|\mathbf{k}\| = n + 1$). Conditioning on n and $\mathbf{k}$ gives

$$M(t) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{\|\mathbf{k}\|=n+1} \Omega[n, \mathbf{k}] M(t, n, \mathbf{k}) \tag{10}$$

where

$$\Omega[n, \mathbf{k}] = P[\text{coloring } \mathbf{k} | n \text{ transitions}] \tag{11}$$

and

$$M(t, n, \mathbf{k}) = M(t)|n \text{ transitions, coloring } \mathbf{k}. \tag{12}$$

This is the main equation that is used to calculate the various performability measures of interest.

In order to apply (12), both $\Omega[n, \mathbf{k}]$ and $M(t, n, \mathbf{k})$ have to be calculated. Note that the quantity $\Omega[n, \mathbf{k}]$ depends only on the underlying discrete Markov chain $\mathcal{Z}$. For the performability measures discussed below, recursions for $\Omega[n, \mathbf{k}]$ were found that are combinatorial in the number of colors and may be quite expensive for general models. However, the nature of highly dependable systems can be exploited to carefully organize the recursions in a way that drastically reduces the computation for the performability measures that we consider. Finding an expression for $M(t, n, \mathbf{k})$ is frequently challenging from a theoretical point of view, but formulas for certain measures can be found by using the following probabilistic reasoning. Consider the set $G_{\mathbf{k}} \subset \mathcal{S}^{n+1}$ of all possible sample paths of $\mathcal{Z}$ such that the first n transitions yield the coloring $\mathbf{k}$. Then for the performability measures considered in this paper, further conditioning on $\nu \in G_{\mathbf{k}}$ reveals that $M(t, n, \mathbf{k}) = M(t, n, \mathbf{k}, \nu)$. That is, the particular sample path influences the measures only through the number of colored intervals and not, for example, through the order in which such intervals occur. To show this result, we must appeal to properties such as exchangeability of the interval length random variables and independence of the Poisson process $\mathcal{N}$ and the discrete chain $\mathcal{Z}$. Using this relationship, expressions were found that allow one to easily calculate the conditional measure $M(t, n, \mathbf{k})$.

The availability case of only rewards 0 and 1 is of special interest. Here the intervals either are "marked" (reward 1) or are not marked (reward 0). Conditioning on n transitions and k marked intervals during $(0, t)$, the main formula becomes

$$M(t) = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{k=0}^{n+1} \Gamma[n, k] M(t, n, k) \tag{13}$$

where

$$\Gamma[n, k] = P[k \text{ marked intervals}|n \text{ transitions}] \tag{14}$$

and

$$M(t, n, k) = M(t)|n \text{ transitions}, k \text{ marked intervals}. \tag{15}$$

4.3.1 Performability Measures Based on Number of Colored Intervals

As a first example of this approach, we discuss the calculation of performability measures based on the number of events that occur during $(0, t)$. Examples of such events include system failure, failure of a particular component that caused the system to fail, repair of a component, and the failure of a component that caused other components to fail. The

rewards we consider correspond to transitions (i.e. failures and repairs). There is a user-defined $M \times M$ matrix $\mathbf{H} = [r_{ij}]$, the (i,j) entry of which is the reward associated with a transition from state a_i to state a_j. Each reward r_{ij} is one of the $K+1$ distinct values $1 = \rho_1 > \cdots > \rho_K > \rho_{K+1} = 0$ discussed above. Given $\mathbf{H}$, the reward matrix $\mathbf{R} = [r_{ij}P_{ij}]$ gives the rewards obtained from one step of the discrete time Markov chain $\mathcal{Z}$. The random variable of interest is $NR(t)$, which represents the total reward accumulated during the period $(0,t)$.

We first note that the expected value of $NR(t)$ can be calculated in a straightforward manner without resorting to the device of coloring intervals. Consider the Markov chain $\mathcal{Z}$, and let σ_k be the reward obtained from the kth transition. Then given n transitions of N in $(0,t)$, the total reward is $NR(t) = \sigma_1 + \cdots + \sigma_n$. The expected reward at the kth transition is given by $E[\sigma_k] = \|\pi(k-1)\mathbf{R}\|$, and so the total expected reward is simply

$$E[NR(t)] = \sum_{n=1}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\sum_{k=1}^{n}\|\pi(k-1)\mathbf{R}\|. \tag{16}$$

To find the distribution function of $NR(t)$ (equivalently its pmf) we use the coloring of intervals and equation (10). Here we must calculate $M(t,n,\mathbf{k}) = P[NR(t) \le r|n,\mathbf{k}]$ and $\Omega[n,\mathbf{k}] = \Omega_1[n,\mathbf{k}]$. Given n transitions and a coloring $\mathbf{k}$, the total reward is $NR(t) = \boldsymbol{\rho}\cdot\mathbf{k} = \sum_{l=1}^{K+1}\rho_l k_l$, since there are k_l transitions with reward ρ_l. Therefore, $P[NR(t) \le r|n,\mathbf{k}] = 1$ if $\boldsymbol{\rho}\cdot\mathbf{k} \le r$ and $P[NR(t) \le r|n,\mathbf{k}] = 0$ otherwise. Thus

$$P[NR(t) \le r] = \sum_{n=1}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\sum_{\|\mathbf{k}\|=n+1,\,\boldsymbol{\rho}\,\mathbf{k}\le r}\Omega_1[n,\mathbf{k}]. \tag{17}$$

The recursion used to calculate $\Omega_1[n,\mathbf{k}]$ is easy to describe. Define $\theta_l[n,\mathbf{k}]$ as the probability that after n transitions of $\mathcal{Z}$, there are k_l transitions with reward ρ_l ($l = 1,\ldots,K+1$) and $Z_n = a_i$. Next define the vector $\theta[n,\mathbf{k}] = \langle\theta_1[n,\mathbf{k}],\ldots,\theta_M[n,\mathbf{k}]\rangle$. The recursion for $\theta[n,\mathbf{k}]$ is

$$\theta_j[n,\mathbf{k}] = \sum_{l=1}^{K+1}\sum_{\{i:r_{ij}=\rho_l\}}\theta_i[n-1,\mathbf{k}-\mathbf{1}_l]P_{ij}, \tag{18}$$

where $\mathbf{1}_l$ is a vector of length $K+1$ with 1 in position l and 0 in every other entry. The above equation holds, since the counter for color l is incremented only if the reward associated to the nth transition is ρ_l. Then we calculate

$$\Omega_1[n,\mathbf{k}] = \|\theta[n,\mathbf{k}]\|. \tag{19}$$

The initial conditions for θ are given by the initial probability distribution of the chain $\mathcal{Z}$, namely,

$$\theta[0,\mathbf{0}] = \pi(0).$$

Note that for general n, not all possible colorings $\mathbf{k}$ need to be considered when computing (17). To see this, for a reward $\rho_l > 0$, any coloring with more than r/ρ_l intervals of color l will yield an accumulated reward during any observation period that is greater than the specified reward level r with probability one. Therefore, such cases need not be considered, which drastically reduces the number of colorings that appear in the recursion.

The special case of 0,1 rewards can be used to obtain measures involving the number of times a particular event occurs (e.g. system failure). In this case, an interval is marked if the transition leading into it is such an event. Let the random variable $\Upsilon(t)$ represent the number of events of interest during the observation period. We wish to calculate the distribution function (PDF) of $\Upsilon(t)$, that is, we set $M(t) = P[\Upsilon(t) \leq K]$ in equation (13). We also set $\Gamma[n,k] = \Gamma_1[n,k]$. With this notation, we have

$$P[\Upsilon(t) \leq K] = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{k=0}^{n+1} \Gamma_1[n,k] P[\Upsilon(t) \leq K | n,k].$$

Given n transitions, at most n intervals can be marked (the first interval is never marked since no transition leads into it), so that $\Gamma_1[n, n+1] = 0$. If k intervals are marked, then $\Upsilon(t) = k$ with probability 1. Therefore, $P[\Upsilon(t) \leq K | n,k] = 1$ for $k \leq K$ and is 0 otherwise. These observations yield the formula

$$P[\Upsilon(t) \leq K] = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{k=0}^{\min(n,K)} \Gamma_1[n,k]. \tag{20}$$

Other quantities of interest include the pmf of $\Upsilon(t)$ and its moments. Clearly we have

$$P[\Upsilon(t) = K] = \sum_{n=K}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \Gamma_1[n,K]. \tag{21}$$

Using (21), the mth moment is

$$E[\{\Upsilon(t)\}^m] = \sum_{n=1}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{k=1}^{n} k^m \Gamma_1[n,k]. \tag{22}$$

Note that when the marked events represent system failure, the reliability at time t is simply the probability of no such event or $P[\Upsilon(t) = 0]$, which is given in (21). The time until the first marked event occurs, $L(t)$, (the lifetime if the distinguished event represents system failure) is another random variable of interest. The distribution of $L(t)$ can be easily calculated by observing that $L(t) > s$ is equivalent to $\Upsilon(s) = 0$ (for $s < t$) and applying (21). Integrating $P[L(t) > s]$ gives the expected time until the first marked transition (the expected lifetime) as

$$E[L(t)] = \frac{1}{\Lambda} \sum_{n=0}^{\infty} E_{n+1,\Lambda}(t) \Gamma_1[n,0], \tag{23}$$

similar to equation (8). This is also equal to

$$E[L(t)] = t \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \left[\frac{\sum_{j=0}^{n} \Gamma_1[j,0]}{n+1} \right]. \tag{24}$$

Note that in this case, truncation at step N gives an error bound of

$$e_L(N) \leq t \sum_{n=N+1}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!}.$$

Letting $t \to \infty$ in (23) gives the mean time until a marked event (mean time to failure) as

$$\text{MTTE} = \frac{1}{\Lambda} \sum_{n=0}^{\infty} \Gamma_1[n,0]. \tag{25}$$

In order to use equation (20) (and the equations for the other measures), it is necessary to calculate $\Gamma_1[n,k]$. A recursion to calculate this quantity is a special case of (18). That is, let $\mathbf{F} = [F_{ij}]$ be a user-defined $M \times M$ matrix of zeros and ones, with $F_{ij} = 1$ if the transition $i \to j$ is an event of interest and $F_{ij} = 0$ otherwise. As an example, the matrix that has a 1 in the (i,j) entry if and only if a_i is an operational state and a_j is a failed state ($a_i \in S_O$, $a_j \in S_F$) gives $\Upsilon(t)$ the interpretation of the number of system failures during $(0,t)$. Similar matrices can be used to represent additional events of interest. Assume then that the matrix $\mathbf{F}$ is given, and (for $j = 1, \ldots, M$) let $\gamma_j[n,k]$ be the probability that n transitions have occurred in $\mathcal{Z}$, k of these transitions are marked and the state after the nth transition is a_j. The vector $\gamma[n,k] = (\gamma_1[n,k], \ldots, \gamma_M[n,k])$ may be calculated using the recursion

$$\gamma_j[n,k] = \sum_{\{i:F_{ij}=1\}} \gamma_i[n-1,k-1]P_{ij} + \sum_{\{i:F_{ij}=0\}} \gamma_i[n-1,k]P_{ij}, \tag{26}$$

where the first sum accounts for the nth transition being an event of interest and the second sum for it not being a marked event. Then we calculate

$$\Gamma_1[n,k] = \sum_{j=1}^{M} \gamma_j[n,k] = \|\gamma[n,k]\|. \tag{27}$$

The recursion (26) is illustrated in Figures 1 and 2 of [24]. As discussed there, the nature of dependability models can be exploited to ensure that such a calculation is tractable. For example, assume that the event of interest is a failure of some type. Since failures seldom occur in highly dependable systems, most of the probability mass of $\Upsilon(t)$ will be clustered around $k = 0$. Therefore, it is highly likely that only values for small k must be calculated to achieve a specified error tolerance. Practical experience confirms that substantial savings result in the computation of such measures for dependability models.

Before concluding our discussion on measures given by marking transitions, we briefly indicate another method of calculating several quantities involving $\Upsilon(t)$. The approach outlined above was developed for the calculation of the distribution (equivalently the pmf) of the number of a certain type of event. Thus the probability of the number of marked intervals in $(0,t)$ $\Gamma_1[n,k]$ had to be calculated. However, if only the mean number of such events is of interest, the following method can be used. The matrix $\mathbf{M}$ with entries $M_{ij} = P_{ij}F_{ij}$ gives the one step marking probabilities for the chain $\mathcal{Z}$, while $\mathbf{P} - \mathbf{M}$ gives the probabilities of not being marked. For $k = 1, \ldots, n$, the probability that the kth transition is marked is seen to be

$$P[k\text{th transition is marked}] = \|\pi(k-1)\mathbf{M}\| = \|\pi(0)\mathbf{P}^{k-1}\mathbf{M}\|.$$

Then the expected number of events of interest during $(0,t)$ can be calculated using the formula

$$E[\Upsilon(t)] = \sum_{n=1}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\sum_{k=1}^{n}\|\pi(k-1)\mathbf{M}\|, \tag{28}$$

which is a special case of equation (16). Determining the probability that no events happen during the observation period (e.g. the system reliability) can also be easily computed. Although this is one term of the pmf, it can be calculated as the probability that no transitions are marked. Therefore,

$$P[\Upsilon(t) = 0] = \sum_{n=1}^{\infty} e^{-\Lambda t}\frac{(\Lambda t)^n}{n!}\|\pi(0)(\mathbf{P} - \mathbf{M})^n\|. \tag{29}$$

Similarly, the distribution and expected value of the time until a marked transition occurs can be calculated using this approach.

4.3.2 Performability Measures Based on Length of Colored Intervals

We next turn to the calculation of performability measures based on the length of intervals of the observation period. In this case, rewards are based on the state of the process X, instead of on transitions. With each state a_i there is associated a reward r_i, which is one of the $K+1$ values $1 = \rho_1 > \cdots > \rho_K > \rho_{K+1} = 0$. Thus corresponding to the Markov process $X(t)$, the random variable $r_{X(t)}$ gives the instantaneous reward at a time instant t. A random variable of more significance is $ACR(t)$, the total accumulated reward during the period $(0,t)$ averaged over its length t. We now consider the calculation of the distribution of $ACR(t)$ with the methodology introduced above based on coloring of intervals.

Given n transitions and a coloring $\mathbf{k}$ (where $\|\mathbf{k}\| = n+1$), let ζ_l be the sum of the lengths of all intervals of color l, for $l = 1, \ldots, K+1$. Then the conditional total reward averaged over t is

$$ACR(t)|n, \mathbf{k} = \frac{1}{t}\sum_{l=1}^{K+1}\rho_l\zeta_l. \tag{30}$$

In this case, we set $M(t) = P[ACR(t) \leq r]$ and $\Omega[n, \mathbf{k}] = \Omega_2[n, \mathbf{k}]$. The parameter r is assumed to satisfy $0 = \rho_{K+1} \leq r \leq \rho_1 = 1$ to avoid trivial cases.

We first discuss the evaluation of the conditional distribution $P[ACR(t) \leq r|n, \mathbf{k}]$. For notational convenience, we assume that every color appears at least once in the vector $\mathbf{k}$ (i.e. $k_l \neq 0$ for $l = 1, \ldots, K+1$). If this is not the case, certain terms are not present in the sum in equation (30) and an obvious relabeling can be done. We will first express the random variables ζ_l in terms of certain intervals of $(0, t)$. To that end, suppose that n transitions of the uniformized chain occur during the observation period at times $0 < \tau_1 < \cdots < \tau_n < t$. These transitions split the period $(0, t)$ into $n+1$ intervals with lengths $Y_1, \ldots, Y_{n+1}$. During the ith interval, the process $\mathcal{X}$ remains in a particular state Z_{i-1}, which depends only on the underlying discrete time Markov chain $\mathcal{Z}$. Furthermore, the transitions are governed by a Poisson process of rate Λ, so that the transition times τ_i are distributed as the order statistics of n independent and identically distributed random variables uniform on $(0, t)$. Let $U_1, \ldots, U_n$ be iid uniform on $(0, 1)$, and let $U_{(1)}, \ldots, U_{(n)}$ be their order statistics. Then we can identify τ_i with $tU_{(i)}$, so that $Y_1 = tU_{(1)}$, $Y_i = t(U_{(i)} - U_{(i-1)})$ for $i = 1, \ldots, n$ and $Y_{n+1} = t(1 - U_{(n)})$. Although it is clear that the Y_i are dependent random variables (their sum is the fixed number t), it is well-known that they are exchangeable [85]. That is, the joint distribution of the Y_i is invariant under any permutation of $1, \ldots, n+1$.

Exchangeability of the Y_i and the independence of $\mathcal{N}$ and $\mathcal{X}$ now allow us to assume that the first k_1 intervals are of color 1, the next k_2 intervals are of color 2, and so on. Defining $n_j = \sum_{l=1}^{j} k_l$ for $j = 1, \ldots, K$, we may assume that

$$\begin{aligned} \zeta_1 &= tU_{(n_1)} \\ \zeta_j &= t(U_{(n_j)} - U_{(n_{j-1})}) \qquad \text{for } j = 2, \ldots, K \\ \zeta_{K+1} &= t(1 - U_{(n_K)}) \end{aligned}$$

Therefore, we may write (recall that $\rho_{K+1} = 0$)

$$ACR(t)|n, \mathbf{k} = \sum_{j=1}^{K} (\rho_j - \rho_{j+1}) U_{(n_j)}, \tag{31}$$

and so the conditional distribution is

$$P[ACR(t) \leq r|n, \mathbf{k}] = P\left[\sum_{j=1}^{K} (\rho_j - \rho_{j+1}) U_{(n_j)}) \leq r\right]. \tag{32}$$

Determining $M(t)|n, \mathbf{k}$ has now been reduced to finding the distribution of a linear combination of uniform order statistics on $(0, 1)$. An expression for this quantity has been obtained by Weisberg [96] (see also [21]), and we now describe his result. Define m to be the largest index i such that $r \leq \rho_i$. Weisberg's result is that

$$P[ACR(t) \leq r|n, \mathbf{k}] = 1 - \sum_{i=1}^{m} \frac{g_i^{(k_i - 1)}(\rho_i)}{(k_i - 1)!}, \tag{33}$$

where $g_i^{(k)}$ is the kth derivative of the function

$$g_i(x) = \frac{(x-r)^n}{\prod_{l=1,l\neq i}^{K+1}(x-\rho_l)^{k_l}}. \tag{34}$$

The derivatives of the functions $g_i(x)$ may be evaluated recursively in a straightf rward manner [96] (see also [24]).

We next develop a recursion for the evaluation of $\Omega_2[n,\mathbf{k}]$, for which the coloring of intervals is based on the state of the process X (equivalently the state of the chain $\mathcal{Z}$). For each state a_j, we define $c(j)$ to be the color associated with it, that is, $r_j = \rho_{c(j)}$. Let $\omega_i[n,\mathbf{k}]$ be the probability that there are k_l intervals with reward ρ_l ($l = 1,\ldots,K+1$) when there are n transitions of $\mathcal{Z}$, and $Z_n = a_i$, and define the vector $\omega[n,\mathbf{k}] = \langle\omega_1[n,\mathbf{k}],\ldots,\omega_M[n,\mathbf{k}]\rangle$. Then the recursion for $\omega[n,\mathbf{k}]$ is given by

$$\omega_j[n,\mathbf{k}] = \sum_{i=1}^{M}\omega_i[n-1,\mathbf{k}-\mathbf{1}_{c(j)}]P_{ij}, \tag{35}$$

where $\mathbf{1}_{c(j)}$ is a vector of length $K+1$ with 1 in position $c(j)$ and 0 in every other entry. That is, when the number of transitions increases from $n-1$ to n and $Z_n = a_j$, the counter for the number of intervals that have color $c(j)$ is incremented by 1 while all other counters remain the same. Then we calculate

$$\Omega_2[n,\mathbf{k}] = \|\omega[n,\mathbf{k}]\|. \tag{36}$$

The remarks concerning the calculation of $\Omega_1[n,\mathbf{k}]$ at the end of the previous section apply equally well to to the above recursion. That is, although the number of vectors $\omega[n,\mathbf{k}]$ grows combinatorially with the number of rewards, the nature of highly dependable systems can be exploited when calculating performability measures for such models. In the present situation, usually higher rewards will be assigned to states that represent a system with a large number of working components (e.g. throughput degrades as more components fail). Furthermore, it is reasonable to assume that the system will spend most of its time in states with few failed components, i.e. the set of states with i failed components is more likely than the set of states with j failed components for $i < j$. Thus, the recursion can be organized in a manner that reduces the computation drastically (see [24] for more details).

We now briefly discuss the calculation of availability measures based on the length of marked intervals (0,1 rewards). One example is the distribution of total up time during $(0,t)$ (the cumulative operational time distribution). Recall that the random variable $O(t)$ represents the amount of time during $(0,t)$ that the system (or a particular set of components) is working. We wish to calculate $M(t) = P[O(t) \leq s]$ (for $s < t$). Equivalently, the distribution of the (interval) availability $A(t) = O(t)/t$ is to be calculated. An interval is

marked in this case if the state corresponding to it belongs to the set of operational states S_O, and we set $\Gamma[n,k] = \Gamma_2[n,k]$ for this marking. From equation (13), recall that we must calculate $\Gamma_2[n,k]$ and $P[O(t) \leq s|n,k]$.

Using exchangeability, the k operational intervals can be assumed to be the first k, so that the conditional distribution is given by the distribution of $U_{(k)}$, that is

$$P[O(t) \leq s|n,k] = \sum_{i=k}^{n} \binom{n}{i} \left(\frac{s}{t}\right)^i \left(1 - \frac{s}{t}\right)^{n-i}.$$

As for $\Gamma_1[n,k]$, a simple recursion is obtained by specializing (35) to the present case. For $1 \leq j \leq M$ define $\eta_j[n,k]$ to be the probability that k of $Z_0, \ldots, Z_n$ of the Markov chain $\mathcal{Z}$ correspond to operational states and $Z_n = a_j$. Also define the vector $\eta[n,k] = (\eta_1[n,k], \ldots, \eta_M[n,k])$. The counter k is incremented only for operational states, so that clearly we have

$$\eta_j[n,k] = \begin{cases} \sum_{i=1}^{M} \eta_i[n-1,k-1]P_{ij} & a_j \in S_O \\ \sum_{i=1}^{M} \eta_i[n-1,k]P_{ij} & a_j \in S_F, \end{cases} \tag{37}$$

where $\mathbf{P}$ is the transition matrix of the uniformized chain $\mathcal{Z}$. Then we calculate

$$\Gamma_2[n,k] = \sum_{j=1}^{M} \eta_j[n,k] = \|\eta[n,k]\|. \tag{38}$$

The initial conditions are given by the initial probability distribution of the chain $\mathcal{Z}$, that is,

$$\eta_j[0,1] = \begin{cases} \pi_j(0) & a_j \in S_O \\ 0 & \text{otherwise.} \end{cases}$$

$$\eta_j[0,0] = \begin{cases} \pi_j(0) & a_j \in S_F \\ 0 & \text{otherwise.} \end{cases}$$

A matrix form for the recursion for $\Gamma_2[n,k]$ is given in [23]. Although the above recursion can in principal be used to calculate the distribution of time during $(0,t)$ in any specified set of states, by exploiting dependability models the cumulative operational time distribution can be calculated with a large computational savings.

Other quantities of interest can also be calculated as byproducts of the above computation. Since $P[A(t) \leq s/t] = P[O(t) \leq s]$, the availability distribution has already been determined. The density of availability can be found by differentiation, with the result that

$$f_{A(t)}(s/t) = \sum_{n=1}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{k=1}^{n} \Gamma_2[n,k] k \binom{n}{k} \left(\frac{s}{t}\right)^{k-1} \left(1 - \frac{s}{t}\right)^{n-k}. \tag{39}$$

The expected availability or the expected fraction of time during $(0,t)$ that the system is working can be calculated as follows. Since the interval lengths Y_i are exchangeable

random variables, it is easily seen that each has expected value $t/(n+1)$. Therefore, given n transitions in $(0,t)$, if k are operational then the expected total up time is $kt/(n+1)$, and so

$$E[A(t)] = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \sum_{k=1}^{n+1} \frac{k}{n+1} \Gamma_2[n,k]. \tag{40}$$

The reliability at time t is the probability that the system does not fail during the observation period, that is, the probability that the system is operational during the entire period $(0,t)$. Given n transitions, the system does not fail if every Y_i corresponds to an operational interval. Thus the reliability is

$$P[O(t) = t] = \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \Gamma_2[n, n+1]. \tag{41}$$

Similar to the discussion about marking of events, the distribution of the lifetime $L(t)$ can be determined by noting that $L(t) > s$ is equivalent to $O(s) = s$ and applying the above formula for reliability. The expected lifetime is obtained by integration as

$$E[L(t)] = \frac{1}{\Lambda} \sum_{n=0}^{\infty} E_{n+1,\Lambda}(t) \Gamma_2[n, n+1], \tag{42}$$

or equivalently

$$E[L(t)] = t \sum_{n=0}^{\infty} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!} \left[\frac{\sum_{j=0}^{n} \Gamma_2[j, j+1]}{n+1} \right]. \tag{43}$$

Finally, the mean time to failure is

$$\text{MTTF} = \frac{1}{\Lambda} \sum_{n=0}^{\infty} \Gamma_2[n, n+1]. \tag{44}$$

Note that $\Gamma_1[n,0] = \Gamma_2[n, n+1]$ when marked transitions represent system failure, so that the expressions for reliability, expected lifetime and mean time to failure are direct translations of previous formulas.

For the transient performability measures discussed in the previous two sections, terms of the Poisson distribution need to be calculated. A stable algorithm for such calculation is given below in section B of the appendix. The limiting measure mean time to failure (equivalently, mean time to event) does not contain such terms, and a straightforward procedure to compute it may converge slowly. This problem is addressed in [22], where a method to speed convergence is presented.

4.3.3 Other Measures

In the preceding two sections, the calculation of cumulative performability measures based on the uniformization method and coloring intervals was presented. However, as discussed in

section 2, other measures are also of interest to the analyst, one of these being the probability that a so-called near coincident fault occurs. In this section we review an approach to that problem using uniformization and the marking of intervals. The preceding sections have also focussed on the calculation of transient measures over a finite observation period, and in this section we also indicate how steady state measures can be obtained. A particular example of an analysis of several scheduled maintenance policies for a repairable system is briefly discussed.

We first introduce the near coincident fault problem and show how it can be addressed using uniformization. In a highly reliable system, the occurrence of two faults in a short period of time may have disastrous consequences. If a failure of a particular type (type 1) occurs within the observation period, there is a vulnerable length of time Δt during which the system identifies and isolates that failure. If a second failure (type 2) occurs during that vulnerable period, perhaps to the components that participate in the recovery process, the result may be catastrophic to the system. Therefore, it is of interest to compute the probability that two such failures occur within a time Δt. One approach to this problem has appeared in the papers [61] and [62], where the effect of such a near coincident fault is modeled explicitly. We now show that the probability of a near coincident fault can be calculated using the methodology presented above based on uniformization and the marking of intervals.

Consider the uniformized time homogeneous Markov process $\mathcal{X}$ that represents the behavior of the system. Given n transitions, recall that the observation period $(0, t)$ is divided into $n+1$ intervals of length $Y_1, \ldots, Y_{n+1}$. Our interest is in marking an interval if it has the possibility of corresponding to a near coincident fault, that is, if the transition leading into it represents a fault of type 1 and the transition leading out of it represents a fault of type 2. Let $CF(t)$ be the random variable that is the minimum of the lengths of the marked intervals, and we seek to calculate $P[CF(t) \leq \Delta t]$.

As for previous cases, we need to calculate $M(t) = P[CF(t) \leq \Delta t | n, k]$ and $\Gamma[n, k] = \Gamma_3[n, k]$ and then apply formula (13). First, using exchangeability, $CF(t)$ has distribution

$$\begin{aligned} P[CF(t) \leq \Delta t | n, k] &= 1 - P[Y_1 > \Delta t, \ldots, Y_k > \Delta t] \\ &= 1 - \{\max(0, 1 - k\Delta t/t)\}^n, \end{aligned}$$

where the last equality follows from known results about uniform order statistics [21].

Although the recursion for $\Gamma_3[n, k]$ is similar to that for $\Gamma_1[n, k]$, it involves the additional complication of a marking based on pairs of transitions. A matrix **F** of zeros and ones indicates the transitions that are classified as faults of type 1, while a similar matrix **G** keeps track of the type 2 faults. We also introduce a 0-1 variable δ, which represents the potential that an interval has of being marked. Define $\chi_i[n, k, 0]$ to be the probability that there are k marked intervals assuming n transitions of $\mathcal{Z}$, the last transition was not

of type 1 and $Z_n = a_i$. Similarly, define $\chi_j[n,k,1]$ in the natural way, and let $\chi[n,k] = \langle \chi_1[n,k,0], \chi_1[n,k,1], \ldots, \chi_M[n,k,0], \chi_M[n,k,1] \rangle$. Clearly $\Gamma_3[n,k] = \|\chi[n,k]\|$. We claim

$$\chi_j[n,k,\delta] = \sum_{i:F_{ij}=\delta} \chi_i[n-1,k,0]P_{ij} + \sum_{i:F_{ij}=\delta,G_{ij}=1} \chi_i[n-1,k-1,1]P_{ij} + \sum_{i:F_{ij}=\delta,G_{ij}=0} \chi_i[n-1,k,1]P_{ij} \quad (45)$$

for $\delta = 0, 1$. To see this, first note that the "new" value of δ on the lefthand side of the equation is determined solely by F_{ij}. Next observe that the first sum accounts for intervals without type 1 transitions leading into them, and so they cannot possibly be marked. The second and third sums account for intervals that have the potential to be marked, and whether or not they do become marked depends on the value of G_{ij} (i.e. whether the transition leading out of the interval is of type 2).

As we have seen, various transient measures can be calculated using the uniformization technique. However, steady state measures may also be of interest to the analyst, and uniformization can be used for their calculation. One example appears in [25], where several scheduled maintenance policies for repairable systems were analyzed. For each model that was considered, the calculation of steady state measures was reduced to the transient case, with the observation period of interest defined in terms of certain embedded points of the process that describes system behavior. Quantities corresponding to these embedded points were obtained using transient uniformization calculations, and results concerning Markov chains with rewards were then used to obtain the steady state results.

4.3.4 Additional Remarks

Uniformization is one of the best solution methods for transient analysis of Markov models. However, certain computational issues have to be addressed. An important issue is the calculation of the Poisson distribution term that appears in the formulas for the measures surveyed, and we include an algorithm for its calculation in section B of the appendix which avoids numerical problems. Another issue is related to *stiff* Markov chains, those which have states with output rates that differ by orders of magnitude. As a consequence, some state transitions occur at a fast rate, while others rarely occur when compared with these transitions. The uniformization technique has computational complexity that is proportional to the length t of the observation interval and to the largest output rate among all states of the model (Λ). Therefore, in order to capture the transient behavior of "slow" events, large values of t (as compared to $1/\Lambda$) have to be chosen, which degrades the performance of the method. Melamed and Yadin [65] address the problem and propose a so-called "select randomization" method. An approximation method based on decomposition of the state space into "fast" and "slow" states has been proposed by Bobbio and Trivedi [6], and the description of a software

tool for stiff Markov chains that combines this approximation technique with uniformization appears in [84].

We have surveyed several methods for calculating performability measures, the two main approaches being transform methods and the uniformization technique. Recently, Donatiello and Grassi [29] have developed a new solution method for calculating the distribution of performability which is based on both uniformization and the basic Laplace transform methodology outlined above. The main idea behind the approach is to first condition the performability distribution function (F_Y) on the number of transitions of the uniformized process, as described earlier. The authors then find a recursive expression for the conditional distribution function ($F_{Y|n}$), by following the main steps of the Laplace transform method used in [30] for acyclic chains. That is, $F_{Y|n}$ is further conditioned on the time τ of the first transition of the embedded Markov process yielding the function ($F_{Y|n,\tau}$). The distribution function of τ can be found by using properties of the Poisson process which governs the transitions of the uniformized chain. The recursive equation for $F_{Y|n,\tau}$ is then transformed in the two variables, a partial fraction expansion is performed and the result is inverted. The final expression for $F_{Y|n,\tau}$ is given in terms of coefficients which are calculated recursively. The approach is shown to be computationally more efficient than the method of [24].

4.4 The State Space Explosion Problem

Performability models may often contain millions or billions of states. Even if usual decomposition assumptions are satisfied, the number of states in a dependability model grows exponentially with the number of components that can fail. Similarly, the number of states in a performance model, such as a queueing network, grows exponentially with the number of resources and customers, and an exact solution is impractical unless the network possesses special properties. As an example, exact algorithms exist for product form networks to calculate efficiently the joint queue length probabilities from which steady state performability measures can be obtained (e.g. [26]). Large state space cardinality has a major impact, not only on numerical solution techniques, but also on model generation. While current tools may be able to handle tens (or even hundreds) of thousands of states, the solution of medium to large size models is an important problem. One way to deal with this problem is through state space reduction techniques, such as truncation of the state space or aggregation of states. The main issue that arises when these techniques are applied is bounding the final error.

In this section we briefly survey some of the approaches that have been proposed to alleviate the state space explosion problem. We present both steady state analysis methods and others that are applicable for transient analysis. Lumping and methods that are concerned specifically with the generation of the most probable states are also discussed.

4.4.1 Steady State Techniques

We first consider methods that have been developed for steady state analysis. As indicated in the introduction, decomposition [17] plays an important role in performability analysis. The basic aggregation/disaggregation technique [17] can also be used for the solution of performance models in order to obtain reward rates to associate with structural models. However, Markov chains corresponding to structural models usually do not have a nearly decomposable structure. Courtois and Semal [18, 19] computed bounds on the steady state probability of each state in a subset of states of a Markov model, conditioned on the system being in a state of the subset. Such results can be used to compute bounds on certain dependability measures, given a specific condition that determines the subset of interest. The results can also be used to bound individual state probabilities, which may lead to a bound on dependability measures. However, the cost of such a "direct" application of the method can be prohibitive for large models (e.g. transition rates between subsets of states must be generated).

Muntz *et al.* [75] extended the basic technique of Courtois and Semal to obtain bounds on steady state availability (or steady state performability). An important characteristic of the method is that only a small subset of states has to be generated and solved. The approach is based on the observation that real systems are designed to have a high level of availability. We therefore expect most of the probability mass to be concentrated on the relatively small subset of states that represent a system with most of its components operational, with the result that the system very rarely reaches other states. The idea is to maintain a detailed description of the system model for such "popular" states and to aggregate the remaining states. To illustrate the approach, consider a Markov model of a system and organize the states in subsets $\mathcal{F}_I$ with I components failed. In the transition rate matrix

$$\begin{bmatrix} Q_{00} & Q_{01} & \cdots & Q_{0,N-1} & Q_{0,N} \\ Q_{10} & Q_{11} & \cdots & Q_{1,N-1} & Q_{1,N} \\ 0 & Q_{21} & \cdots & Q_{2,N-1} & Q_{2,N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & Q_{N-1,N-1} & Q_{N-1,N} \\ 0 & 0 & \cdots & Q_{N,N-1} & Q_{N,N} \end{bmatrix}, \tag{46}$$

the submatrices Q_{IJ} represent the transitions between states in subsets $\mathcal{F}_I$ and $\mathcal{F}_J$, $I, J = 0, \ldots, N$. The submatrices Q_{IJ} for $I - J \geq 2$ are identically zero, since it is assumed that two or more components cannot finish repair at exactly the same time. However, this assumption does not preclude multiple repair facilities or operational dependency constructs (e.g. SAVE).

Now assume that we further partition the state space into two subsets $\mathcal{D} = \bigcup_{i=0}^{K-1} \mathcal{F}_i$ and $\mathcal{R} = \bigcup_{i=K}^{N} \mathcal{F}_i$. Clearly, the steady state availability A is given by

$$A = P(\mathcal{D})A_{\mathcal{D}} + P(\mathcal{R})A_{\mathcal{R}},$$

where $P(\mathcal{D})$ ($P(\mathcal{R})$) is the probability that the system is in a state of $\mathcal{D}$ ($\mathcal{R}$) and $A_{\mathcal{D}}$ ($A_{\mathcal{R}}$) is the conditional steady state availability given that the system is in $\mathcal{D}$ ($\mathcal{R}$). Since $0 \leq A_{\mathcal{D}} \leq 1$, $0 \leq A_{\mathcal{R}} \leq 1$ and $P(\mathcal{R}) = 1 - P(\mathcal{D})$, it is easy to see that

$$P(\mathcal{D})A_{\mathcal{D}} \leq A \leq P(\mathcal{D})A_{\mathcal{D}} + 1 - P(\mathcal{D}).$$

Therefore,

$$\{P(\mathcal{D})\}_{\text{lb}}\{A_{\mathcal{D}}\}_{\text{lb}} \leq A \leq 1 - \{P(\mathcal{D})\}_{\text{lb}}[1 - \{A_{\mathcal{D}}\}_{\text{ub}}],$$

where subscript "lb" ("ub") indicates a lower (upper) bound on the term. Bounds on $P(\mathcal{D})$ and $A_{\mathcal{D}}$ can be obtained by the results of [18], but the procedure may be too expensive for large models, and the availability bounds may not be tight [75]. Instead, construct the following matrix from (46)

$$\left[\begin{array}{cc|cccccccc}
Q_{00} & Q_{0B} & 0 & Q_{0K} & \cdots & Q_{0L} & Q_{0,L+1} & \cdots & Q_{0,N-1} & Q_{0,N} \\
Q_{B0} & Q_{BB} & 0 & Q_{BK} & \cdots & Q_{BL} & Q_{B,L+1} & \cdots & Q_{B,N-1} & Q_{BN} \\
\hline
Q_{B0} & 0 & Q_{BB} & Q_{BK} & \cdots & Q_{BL} & Q_{B,L+1} & \cdots & Q_{B,N-1} & Q_{BN} \\
0 & 0 & Q_{KB} & Q_{KK} & \cdots & Q_{KL} & Q_{K,L+1} & \cdots & Q_{K,N-1} & Q_{KN} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & Q_{LL} & Q_{L,L+1} & \cdots & Q_{L,N-1} & Q_{LN} \\
0 & 0 & 0 & 0 & \cdots & Q_{L+1,L} & Q_{L+1,L+1} & \cdots & Q_{L+1,N-1} & Q_{L+1,N} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & Q_{N-1,N-1} & Q_{N-1,N} \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & Q_{N,N-1} & Q_{N,N}
\end{array}\right], \tag{47}$$

where B represents the subset of states $\bigcup_{i=1}^{K-1} \mathcal{F}_i$, and the states in this subset have been replicated in a way that yields an equivalent model. Let $\mathcal{D}'$ ($\mathcal{R}'$) represent states corresponding to the upper left corner (lower right corner) of the matrix in (47). Note that system transitions occur for the most part between states in $\mathcal{D}'$, and every entry into $\mathcal{D}'$ from $\mathcal{R}'$ is through the single state 0 (no failed components). Clearly, $P(\mathcal{D}') \leq P(\mathcal{D})$. We now aggregate the states in each of the $N - K + 2$ subsets in $\mathcal{R}'$. Although an exact aggregation would produce exact results, it would require the solution of the whole model. However, for dependability models, the rates between aggregates can be easily bounded as

$$\left[\begin{array}{cc|cccccccc}
Q_{00} & Q_{0B} & 0 & Q_{0K} & \cdots & Q_{0L} & Q_{0,L+1} & \cdots & Q_{0,N-1} & Q_{0,N} \\
Q_{B0} & Q_{BB} & 0 & Q_{BK} & \cdots & Q_{BL} & Q_{B,L+1} & \cdots & Q_{B,N-1} & Q_{BN} \\
\hline
- & 0 & \bullet & + & \cdots & + & + & \cdots & + & + \\
0 & 0 & - & \bullet & \cdots & + & + & \cdots & + & + \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & \bullet & + & \cdots & + & + \\
0 & 0 & 0 & 0 & \cdots & - & \bullet & \cdots & + & + \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & \bullet & + \\
0 & 0 & 0 & 0 & \cdots & 0 & 0 & \cdots & - & \bullet
\end{array}\right], \tag{48}$$

where + represents an upper bound on failure rates (e.g. the sum of failure rates), − represents a lower bound on repair rates (e.g. the minimum of repair rates) and • represents an aggregate state. Using (48) decreases the residence time in $\mathcal{D}'$ whenever this subset is reentered. If we then solve for the steady state probabilities of (48) and assign reward 0 (1) to aggregate states in R', lower (upper) bounds on availability are obtained. Additional details can be found in [75]. Extensions that provide the capability of increasing the accuracy of the bounds by incrementally generating more states of the transition rate matrix are presented in [76].

4.4.2 Transient Techniques

We now consider methods developed for transient analysis. A simple but not necessarily good bound can be obtained by truncating the state space, i.e. only certain states in the Markovian model are generated. The remaining states are aggregated into a single absorbing state. Now assume that the rewards are bounded, $r_{\rm lb} \leq r_i \leq r_{\rm ub}$ for all i. It is easy to see that the distribution of cumulative reward in the original model is bounded by

$$I^{\rm lb}[CR(t) > y] \leq I[CR(t) > y] \leq I^{\rm ub}[CR(t) > y],$$

where the superscript lb (ub) indicates that the absorbing state in the truncated model is assigned reward $r_{\rm lb}$ ($r_{\rm ub}$). Note that the quality of the bounds depends on the probability of reaching a nongenerated state during the observation period. An important issue is how to generate states so as to minimize such a probability, and this will be discussed later in the section.

In [20], Courtois and Semal obtain bounds on the mean and second moment of the time until absorption (e.g. the time until failure or, more generally, the time until a critical state is reached). Instead of using a chain with a single absorbing state, they work with a transformed process which is ergodic. Consider a chain with one absorbing state and stochastic matrix

$$\begin{bmatrix} P_{00} & P_{01} & \cdots & P_{0,N-1} & P_{0N} \\ P_{10} & P_{11} & \cdots & P_{1,N-1} & P_{1N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \tag{49}$$

where the submatrices P_{IJ}, $I, J = 0, \ldots, N-1$ correspond to particular subsets of states. Now construct the matrix

$$\mathbf{P} = \begin{bmatrix} P_{00} & P_{01} & \cdots & P_{0,N-1} & P_{0N} \\ P_{10} & P_{11} & \cdots & P_{1,N-1} & P_{1N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ d_0 & d_1 & \cdots & d_{N-1} & 0 \end{bmatrix} \tag{50}$$

where $\mathbf{d} = \langle d_0, \ldots, d_{N-1} \rangle$ is the initial probability vector. Note that each time the Markov chain of (50) reaches state N (the absorbing state in (49)), the chain restarts under the initial distribution. Let the vector π be the solution of $\pi = \pi \mathbf{P}$ (normalized so that its entries sum to 1), and consider its last entry π_N. Now $1/\pi_N$ is the mean time between two visits of state N, and thus intuitively the mean time to absorption (MTTA) of (49) is

$$\text{MTTA} = \frac{1}{\pi_N} - 1. \tag{51}$$

This suggests that a bound on MTTA can be obtained in the following manner. First, determine bounds on π_I, the conditional steady state distribution of states in subset I, for all I. These bounds can be obtained using results in [18] by solving a matrix with states in subset I only. Second, from the above bounds and additional results in [18], obtain bounds on the steady state distribution of each subset I using a matrix formed by aggregating (approximately) states in subset I, $I = 0, \ldots, N-1$. Finally, from these latter bounds, use equation (51) to bound MTTA. Further results concerning the tightness of these bounds in addition to bounds on the second moment of the time until absorption can be found in [20].

Bounding the absolute value of the difference between the expected accumulated reward of a Markov model and that of a perturbed model for which the rewards and/or transition probabilities are "slightly" different was considered by van Dijk [94]. The truncation of a Markov chain model is such an example, and one may bound the results for the original model by solving the truncated model. Other examples include the transformation of a model into one with special structure that can be efficiently solved (e.g. a product form queueing network). The approach can be applied to steady state as well as transient measures [95]. Further applications of this approach can be found in [44].

4.4.3 Lumping

Lumping is a method commonly used to reduce the state space of a Markov model, while preserving sufficient detail in the lumped model so that the measures of interest can be evaluated. In order to illustrate the general idea, consider the example of Figure 1 and assume that both processors can fail, that they have identical failure and repair rates and that they are repaired independently. The structural model of the system has four states, $(0,0)$ representing both processors operating, $(1,1)$ representing both processors failed and $(0,1)$ and $(1,0)$ representing a single operational processor. If the reward rates associated with states $(0,1)$ and $(1,0)$ are identical, it is not hard to see that these two states can be lumped (combined) and performability measures can be easily calculated from the solution of the lumped chain. In [55] conditions on the transition probabilities of a Markov model are given for lumping states into disjoint sets. Briefly, the lumping of states into the disjoint sets $S_1, \ldots, S_N$ may be possible when the sum of the transition rates from a state $a_i \in S_I$

to states in S_J, $J \neq I$, are identical for all states in S_I, $I = 1, \ldots, N$. However, lumping also depends on the measure to be calculated as well as the reward rates assigned to the states. For instance, in the example above, lumping is possible even when the reward rates assigned to states $(0,1)$ and $(1,0)$ are different, if the measure of interest is the expected reward averaged over t for $t \rightarrow \infty$. However, this is not the case if one wants to calculate the distribution of accumulated reward over a finite interval. Lumping has been applied to availability modeling [35]. In [78] a discussion of lumping applied to Markov reward models is presented.

An important issue for performability modeling is how to identify the states that can be lumped from a high level model description. In a recent paper, Sanders and Meyer [89] proposed a method of contructing models for which the state space generated is significantly reduced when compared to the state space obtained from the application of "usual" model construction and state space generation techniques. Their approach is based on recognizing identical submodels in a SAN model and using results on lumping. Roughly, submodels with identical behavior are combined and joined with different submodels to construct the final SAN reward model. The construction process is such that the lumpability conditions are satisfied whenever possible. Both the structure of the model as well as the performability measures to be calculated are taken into account in the selection of the appropriate state variables of the reduced state space.

4.4.4 Generating the Most Probable States

The bounding methodology of [75] assumes that a natural partitioning of the state space exists, such that the majority of the probability mass is concentrated in the states of the initial partitions (i.e. partitions $\mathcal{F}_I$ with smaller values of I). Although this works well for most availability models, in general it may not be an easy task to find the most probable states. So-called "dynamic state exploration" techniques address the issue of generating the most probable states automatically.

Grassmann [41, 42] implements the uniformization technique in a dynamic fashion. At a step n of the uniformization procedure, designate a set of active states as those having a nonzero probability. This set is dynamically adjusted at each step, and the transition rate matrix is generated incrementally according to the current set of active states. This procedure avoids the generation of the entire transition rate matrix before the uniformization algorithm starts.

Yang and Kubat [97] developed an algorithm to generate the most probable states, one by one, until a stopping criterion is reached for a specific model of Li and Silvester [58]. The model is composed of N components, and component i operates in a given mode j with

probability P_{ij}. The procedure works because of the characteristics of the model, but it does not apply to general models.

Dimitrijevic and Chen [27] proposed a dynamic state exploration technique for general Markov models. A rough outline of the algorithm is as follows. The generated states are divided into two subsets, the explored states (S_E) and the unexplored states (S_U). At each step of the algorithm, the number of visits V_i to each state a_i between visits to the initial state is calculated. The state in S_U with the largest such value (say a_k) is removed from S_U and placed in S_E. Finally, S_U is updated by including the states that are reachable from a_k but are not yet in either S_E or S_U. The algorithm terminates when a stopping criterion is met, for example when the mean residence time in S_E is greater than a given tolerance. As part of the procedure, an efficient recursive computation of the visit ratios was developed. In [28], the algorithm was improved to reduce the computational complexity to $O(N^3)$ (the storage requirements remain $O(N^2)$), where N is the number of generated states.

Recently Mejia Ochoa and de Souza e Silva [63] showed that computational and storage improvements can be obtained for the above method by using iterative techniques in the calculation of the visit ratios. Furthermore, they proposed a new algorithm that chooses a new state at each step so that the mean residence time is maximized when the state is included in S_E. This method of selecting states has advantages over previous approaches as discussed in [63]. Other measures that can be used in choosing a state are also investigated.

5 Summary

The interest in performability related problems has grown noticeably in the past ten years, and the accomplishments to date are many. We have tried to indicate some of the important issues involving the specification and solution of performability models and to emphasize the relationship between these two problems.

Concerning model specification, there is a clear need for high level languages tailored to the way the user thinks about his system. Among the important issues that arise in the development of a tool we mention the flexibility of adding new features to the language and the method by which performability measures are specified from a high level description of the system and then mapped to the mathematical model. These issues have been addressed and some of the proposed approaches have been discussed.

As for the solution of the mathematical model, we have surveyed various methods proposed in the literature, notably Laplace transform methods, decomposition methods and methods based on the uniformization technique. This last technique provides a unified approach to calculating many performability measures, and it was considered in detail. A

Markov performability model of real systems may have millions or billions of states, and thus the need to deal with the state space explosion problem is evident. Decomposition techniques are promising in this respect, since their basic approach is to obtain bounds without resorting to the solution of the whole model. Other promising techniques include those based on identifying symmetry on a high level model specification in order to obtain a reduced (lumped) state space. Note also that generation of the complete transition matrix can be avoided using the uniformization technique, as mentioned in section 4.

Solution techniques are closely tied to model specification and generation. Not only does the specification of a measure to be calculated have an impact on the choice of solution method, but solution techniques may influence model generation as discussed in section 4. As an example, the fact that different measures can be calculated in parallel using uniformization may be taken into account by the analyst. Another example of such influence is the way in which a reward structure is obtained for use in a structural model. We wish to emphasize the importance of considering jointly model specification and solution for performability evaluation.

Appendix

A Specification Examples Using the Object Oriented Paradigm

In this section of the appendix, we illustrate the modeling paradigm proposed in [5] by presenting the object definition for two types of objects. We also show how the objects can be instantiated in order to construct the model of a system.

For the first example, we choose to describe an object type that has all of the features of a SAVE component [34]. We call this object type "SAVE_COMPONENT" (recall that the behavior of a generic SAVE component is summarized in Figure 5). Although Prolog is used to implement the paradigm, the description below can be understood without previous knowledge of this language.

```
TYPE : SAVE_COMPONENT;
NAME : Object_Name;
STATE : (Mode,Numb_Failures);
EVENT fail: (Mode,Numb_Fails1) → (Mode,Numb_Fails2);
        READ :      comp(Object_Name,Numb_Comp),
                    repair(Object_Name, Mode, Component_Repair).
```

```
                    affected_components(Object_Name, Mode,List_of_Components_Affected);
        EVALUATION : calc_total_fails(Object_Name, Total_Fails),
                    calc_rate(Object_Name,Mode,T);
        CONDITION :  Total_Fails < Numb_Comp,
                    T > 0;
        ACTION :     Numb_Fails2 := Numb_Fails1 + 1,
                    send_message(msg("fail",Object_Name,Mode,1), Component_Repair),
                    affect_components("fail", List_of_Components_Affected);
        RATE :       T;
MESSAGE ("repaired", Mode, Numb): (Mode,Numb_Fails1) → (Mode,Numb_Fails2);
        ACTION : Numb_Fails2 := Numb_Fails1 - Numb;
MESSAGE ("fail", Mode,Numb_Affected,Affect_Prob,Prob_Mode) :
        (Mode,Numb_Fails1) → (Mode,Numb_Fails2);
        READ :       comp(Object_Name,Numb_Comp);
        EVALUATION : calc_total_fails(Object_Name,Total_Fails);
        CONDITION :  Total_Fails < Numb_Comp;
        ACTION :     Numb_Fails2 = Numb_Fails1;
        PROBABILITY : 1 - (Affect_Prob * Prob_Mode);

        READ :       comp(Object_Name,Numb_Comp),
                    repair(Object_Name, Mode, Component_Repair);
        EVALUATION : total_fails(Object_Name,Total_Fails);
        CONDITION :  Numb_Affected ≤ Numb_Comp - Total_Fails;
        ACTION :     Numb_Fails2 = Numb_Fails1 + Numb_Affected,
                    send_message(msg("fail", Object_Name,Mode,Numb_Affected),
                       Component_Repair);
        PROBABILITY : Affect_Prob * Prob_Mode;

        READ :       comp(Object_Name,Numb_Comp),
                    repair(Object_Name, Mode, Component_Repair);
        EVALUATION : total_fails(Object_Name,Total_Fails);
        CONDITION :  Numb_Affected > Numb_Comp - Total_Fails;
        ACTION :     Numb_Fails2 = Numb_Comp,
                    send_message(msg("fail",Object_Name,Mode,
                       Numb_Comp - Total_Fails), Component_Repair);
        PROBABILITY : Affect_Prob * Prob_Mode;
```

The object type definition begins with the name of this type of object: SAVE_COMPONENT. The NAME statement indicates the name of the variable to be replaced by the name of the object when instantiated. STATE includes the parameters that define the state of the object: the failure mode and the number of units that are failed in that failure mode. Note that the state can represent units failed in as many different failure modes as defined by the user.

From the description above we note that the object type SAVE_COMPONENT has one event called *fail* and can receive two messages: a *repaired* message and a *fail* message. The EVENT and MESSAGE statements indicate the initial and final state of the object when the actions related to each statement are performed ((*initial state*) → (*final state*)). The

event *fail* has a rate T which is calculated by a (Prolog) function (not shown) of the state of the object and other parameters supplied by the user. The clauses under READ and EVALUATION are used to assign parameter values (given by the user) relevant to this object and to evaluate the value of certain variables. The *comp(...)* clause simply obtains the number of units for the object. The clause *repair* obtains the name of the object which is responsible for repairing a unit that is failed in mode "Mode". The *affected_components* clause obtains the list of components that are possibly affected when a unit fails in mode "Mode". The clause *calc_total_fails(...)* is a function which calculates the total number of units of this component that are currently failed, and *calc_rate(...)* is a function which calculates the value of T. Following the clauses under READ and EVALUATION, there is one set of actions and their preconditions. If the total number of failed units is less than the total number of units of the object and $T > 0$, then the following actions are taken: (a) the number of failed units is incremented; (b) a message *fail* is sent to object type "Component_Repair", which will repair the unit. This message is represented by the function *send_message(msg(X), destination)* where X is the message body (data to be sent) and *destination* is the name of the object that will receive the message and; (c) the function *affect_components* results in a message to be sent to each component in the list of affected components obtained by the *affected_components* clause. This function can be defined in Prolog as:

```
affect_components("fail",[ ]).
affect_components("fail", (Name,Mode,Num_Affected,affect_Prob,Prob_Mode).Tail) <-
  send_message(msg("fail", Mode,Numb_Affected,Affect_Prob,Prob_Mode), Name),
  affect_components("fail",Tail).
```

As indicated above, each element in this list of affected components gives the name of an object that may be affected ("Name"), the mode of failure if affected ("Mode"), the number of units affected ("Num_Affected"), the probability of affecting the object ("affect_Prob") and the probability that the affected units fail in that mode ("Prob_Mode"). If the list of affected components is empty ("[ ]"), no action is taken. Otherwise, a message *fail* is sent to the first object in the list, and the function *affect_components* is called recursively with the tail of the list. Other functions mentioned above can also be easily defined.

The first of the two messages that can be received indicates that a repair was performed and a simple action is taken: the number of units failed in mode "Mode" is decremented by the number of repaired units ("Numb"). In the second message, the object is informed that another component has failed and that this object may be affected. The message also informs the number of units to be affected, the probability that this object is affected and the probability of being affected in the indicated mode ("Mode"). Note that each action is associated with a probability that the action is performed. The actions performed can be inferred from the above explanation.

It is interesting to observe that (as in SAVE) when an object fails it may affect other components, but an affected component cannot affect any other component. This behavior can be easily altered if we include the clauses *affected_components* and *affect_components* in the definition of the *fail* message. In this way, it would be possible to model components that are affected by other affected components in the system.

Once this object type is defined, it can be instantiated to create a system model. For instance, to define a database system with two processors, a front-end and a database as described in [34], the following clauses have to be included by the user:

```
TYPE(processor, SAVE_COMPONENT).
TYPE(front_end, SAVE_COMPONENT).
TYPE(database, SAVE_COMPONENT).
TYPE(repair_server, PRIORITY_SERVER).
```

These clauses instantiate three objects of type SAVE_COMPONENT and one object of type PRIORITY_SERVER (to be defined below) to model the repair queue. Other clauses have to be included to give the number of units for each object, the list of objects affected by a failure, the failure and repair rates, etc. Below we give several examples of these clauses.

```
COMP(processor,2).
COMP(front_end,1).
COMP(database,1).

AFFECTED_COMPONENTS(processor,1,[ ]).
AFFECTED_COMPONENTS(processor,2,[database,1,1,1-c,1]).
AFFECTED_COMPONENTS(front_end,1,[ ]).
AFFECTED_COMPONENTS(database,1,[ ]).
```

The first *affected_components* clause indicates that, if a processor fails in mode 1, it does not affect any component. The second *affected_components* clause indicates that, if a processor fails in mode 2, the database is affected with probability $1 - c$.

As a second example to further illustrate the object oriented paradigm, we next define an object type priority server. Queues and queueing networks are the main modeling paradigm used for performance evaluation. They are also used in availability modeling to represent contention for the resources that perform repairs. A queue may be considered as a collection of tasks that contend for one or more servers. Each task brings one or more units of work to be executed by the servers. The servers choose the next task to be executed according to a given policy. We organize the tasks into classes which define the route to be followed when they finish service (the next object to be visited) and the amount of work to be executed during service. For a priority server, classes are also associated with a service priority, and

the mapping is defined by the user. Tasks may change classes as they move from object to object.

Below is the definition of an object of type PRIORITY_SERVER and the (Prolog) function *put_queue*, which includes a received task in the proper place of the queue according to its priority. In our notation, priority number 1 is associated with the highest priority tasks.

```
TYPE : PRIORITY_SERVER;
NAME : Object_Name;
STATE : Queue;
EVENT serve: Queue1 → Queue2;
        READ :          rate(Object_Name, Initial_Class, T),
                        capacity(Object_Name, Capacity),
                        route(Object_Name, Initial_Class, Prob, Destination, Final_Class);
        CONDITION : Queue1 = (Initial_Class, Prior).Tail;
        ACTION :    Queue2 = Tail,
                    send_message(msg("task", Final_Class), Destination);
        RATE :      Prob * (Capacity / T);
MESSAGE ("task", Class) : Queue1 → Queue2;
          READ :    mapping(Class, Prior),
          ACTION : put_queue((Class, Prior),Queue1,Queue2);
          PROBABILITY : 1;

put_queue((Class, Prior),[ ],(Class, Prior)).
put_queue((Class, Prior), (Class2, Prior2).Tail2, (Class, Prior).(Class2, Prior2).Tail2) <-
   Prior < Prior2.
put_queue((Class, Prior), (Class2, Prior2).Tail2, (Class2, Prior2).New_Tail) <-
   Prior ≥ Prior2,
   put_queue((Class, Prior), Tail2, New_Tail).
```

Note that there is only one event *serve* which indicates that a task is being served. A message *task* indicates that a new task has arrived. The clauses *rate*, *capacity* and *route* are specified by the user when the model is created and indicate, respectively, the amount of work (in units of work/task) associated to a task of a particular class, the capacity (in units of work/unit of time) of the server, the route to be followed by the task and its next class when the task finishes service.

B Calculation of the Poisson Distribution

In this section of the appendix, we present a numerically stable recursion for computing the terms of the Poisson distribution, which always appear in the transient expressions that

are calculated using the uniformization technique. Evaluating the performance measures presented in this paper involves the calculation of

$$\alpha_N = \sum_{n=0}^{N} e^{-\Lambda t} \frac{(\Lambda t)^n}{n!}, \tag{52}$$

where Λ is the Poisson rate of the uniformized process and N is such that the desired error tolerance is achieved (that is, $\alpha_N \geq 1 - \epsilon$). If α_N is evaluated in a straightforward way, it is very likely that underflow/overflow conditions will arise, even for moderate values of Λt. In what follows, we describe a simple algorithm which avoids underflow/overflow problems.

Let $\beta(n) = e^{-\Lambda t}(\Lambda t)^n/n!$, i.e. $\beta(n)$ is the $(n+1)$st term in α_N. It is well-known that the Poisson distribution can be approximated by a normal distribution for large values of Λt. Therefore, we can choose $N_{\min}$ such that the lower tail of the distribution has a value $\ll \epsilon$, but still is greater than the underflow limit. For example, using

$$N_{\min} = \max(0, \Lambda t - 10\sqrt{\Lambda t})$$

gives a value for the lower tail of the distribution around 10^{-30}, which is negligible in comparison to a reasonable value of ϵ (see also [38]). But $\beta(N_{\min})$ is greater than the underflow value in FORTRAN77 (approximately 10^{-70}).

Note that $N_{\min} = 0$ for small values of Λt, and then $\beta(N_{\min}) = e^{-\Lambda t}$. For $N_{\min} > 0$ we use the following algorithm. Define $E = \Lambda t e^{-\Lambda t/(N_{\min}+1)}$. Since $N_{\min} \simeq \Lambda t$ for large values of Λt, we have $E \simeq \Lambda t/e$. Now note that

$$\beta(N_{\min}) = e^{-\Lambda t/(N_{\min}+1)} \prod_{i=1}^{N_{\min}} \frac{E}{i}.$$

The idea of the algorithm is to calculate the above product so that the intermediate result stays as close to 1 as possible. To that end, we choose i such that $E/i \simeq 1$. Multiplying by E/j, $j < i$ ($j > i$) will tend to increase (decrease) the intermediate result. We use this observation to choose the proper value j such that E/j is the next term by which the intermediate result is multiplied. Finally, since $\beta(N_{\min})$ is greater than the underflow value, the final result is guaranteed to be calculable.

The remaining terms $\beta(n)$, for $N_{\min} < n \leq N$, are calculated by the recursion

$$\beta(n) = \beta(n-1)\frac{\Lambda t}{n},$$

and N is calculated such that $\sum_{N_{\min}}^{N} \beta(n) \geq 1 - \epsilon$.

Acknowledgments

We wish to thank B.R. Haverkort, J.C.S. Lui and R.R. Muntz for useful comments and suggestions.

References

[1] M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, 1985.

[2] M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. on Computer Systems*, 2:93–122, 1984.

[3] S.J. Bavuso. A user's view of CARE III. In *Proceedings of the 1984 Annual Reliability and Maintainability Symposium*, pages 382–389, January 1984.

[4] M.D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Trans. on Computers*, C-27(6):540–547, 1978.

[5] S. Berson, E. de Souza e Silva, and R.R. Muntz. An object oriented methodology for the specification of Markov models. In *The First International Conference on the Numerical Solution of Markov Chains*, pages 2–29, 1990.

[6] A. Bobbio and K.S. Trivedi. An aggregation technique for the transient analysis of stiff Markov chains. *IEEE Trans. on Computers*, C-35(9):803–814, 1986.

[7] B.R. Borgerson and R.F. Freitas. A reliability model for gracefully degrading and standby-sparing systems. *IEEE Trans. on Computers*, C-24(5):517–525, 1975.

[8] J.A. Carrasco and J. Figueras. METFAC: design and implementation of a software tool for modeling and evaluation of complex fault-tolerant computing systems. In *Proceedings of FTCS-16*, pages 424–429, 1986.

[9] E. Çinlar. *Introduction to Stochastic Processes*. Prentice-Hall, 1975.

[10] G. Chiola. A software package for the analysis of generalized stochastic Petri net models. In *Proceedings of the International Workshop on Timed Petri-nets*, pages 136–143, 1985.

[11] G. Ciardo, R.A. Marie, B. Sericola, and K.S. Trivedi. Performability analysis using semi-Markov reward processes. *IEEE Trans. on Computers*, C-39(10):1251–1264, 1990.

[12] G. Ciardo, J. Muppala, and K.S. Trivedi. SPNP: stochastic Petri net package. In *Proceedings of the Third International Workshop on Petri-nets and Performance Models*, pages 142–151, 1989.

[13] B. Ciciani and V. Grassi. Performability evaluation of fault-tolerant satellite systems. *IEEE Trans. on Communications*, COM-35(4):403–409, 1987.

[14] R. Cléroux and D.A. Kadi. A summary of periodic replacement policies with minimal repair. *Investigación Operativa*, 1(1):43–54, 1988.

[15] A. Costes, J.E. Doucet, C. Landrault, and J.C. Laprie. SURF: a program for dependability evaluation of complex fault-tolerant computing systems. In *Proceedings of FTCS-11*, pages 72–78, 1981.

[16] P.J. Courtois. Error analysis in nearly completely decomposable stochastic systems. *Econometrica*, 43:691–709, 1975.

[17] P.J. Courtois. *Decomposability: Queueing and Computer System Applications*. Academic Press, 1977.

[18] P.J. Courtois and P. Semal. Bounds for the positive eigenvectors of nonnegative matrices and for their approximations. *Journal of the ACM*, 31:804–825, 1984.

[19] P.J. Courtois and P. Semal. Computable bounds for conditional steady-state probabilities in large Markov chains and queueing models. *IEEE Journal on Selected Areas in Communications*, SAC-4(6):926–937, 1986.

[20] P.J. Courtois and P. Semal. Bounds for transient characteristics of large or infinite Markov chains. In *The First International Conference on the Numerical Solution of Markov Chains*, pages 446–468, 1990.

[21] H.A. David. *Order Statistics, 2nd Ed.* John Wiley & Sons, 1981.

[22] E. de Souza e Silva and H.R. Gail. Calculating availability and performability measures of repairable computer systems using randomization. Technical report, IBM Research Report RC 12386, Yorktown Heights, N. Y., 1986.

[23] E. de Souza e Silva and H.R. Gail. Calculating cumulative operational time distributions of repairable computer systems. *IEEE Trans. on Computers*, C-35(4):322–332, 1986.

[24] E. de Souza e Silva and H.R. Gail. Calculating availability and performability measures of repairable computer systems using randomization. *Journal of the ACM*, 36(1):171–193, 1989.

[25] E. de Souza e Silva and H.R. Gail. Analyzing scheduled maintenance policies for repairable computer systems. *IEEE Trans. on Computers*, 39(11):1309–1324, 1990.

[26] E. de Souza e Silva and S.S. Lavenberg. Calculating joint queue length distributions in product form queueing networks. *Journal of the ACM*, 36(1):194–207, 1989

[27] D.D. Dimitrijevic and M.-S. Chen. An integrated algorithm for probabilistic protocol verification and evaluation. In *Proceedings of INFOCOM'89*, 1989.

[28] D.D. Dimitrijevic and M.-S. Chen. Dynamic state exploration in quantitative protocol analysis. In *Protocol Specification, Testing, and Verification, IX*, pages 327–338. North-Holland, 1990.

[29] L. Donatiello and V. Grassi. On evaluating the cumulative performance distribution of fault-tolerant computer systems. Technical report, University of Pisa, 1991.

[30] L. Donatiello and B.R. Iyer. Analysis of a composite performance reliability measure for fault-tolerant systems. *Journal of the ACM*, 34(1):179-199, 1987.

[31] J.B. Dugan, K.S. Trivedi, R.M. Geist, and V.F. Nicola. Extended stochastic Petri nets: Applications and analysis. In *Performance 84*, pages 507-519, 1984.

[32] D.G. Furchtgott and Meyer. Closed-form solutions of performability. *IEEE Trans. on Computers*, C-33(6):550-554, 1984.

[33] R.M. Geist and K.S. Trivedi. Ultra-high reliability prediction for fault-tolerant computer systems. *IEEE Trans. on Computers*, C-32(12):1118-1127, 1983.

[34] A. Goyal, W.C. Carter, E. de Souza e Silva, S.S. Lavenberg, and K.S. Trivedi. The system availability estimator. In *Proceedings of FTCS-16*, pages 84-89, 1986.

[35] A. Goyal, S.S. Lavenberg, and K.S. Trivedi. Probabilistic modeling of computer system availability. *Annals of Operations Research*, 8:285-306, 1987.

[36] A. Goyal and A.N. Tantawi. Evaluation of performability for degradable computer systems. *IEEE Trans. on Computers*, C-36(6):738–744, 1987.

[37] V. Grassi, L. Donatiello, and G. Iazeolla. Performability evaluation of multicomponent fault-tolerant systems. *IEEE Trans. on Reliability*, 37(2):216–222, 1988.

[38] W.K. Grassmann. Transient solutions in Markovian queueing systems. *Comput. & Ops. Res.*, 4:47–53, 1977.

[39] W.K. Grassmann. Transient solutions in Markovian queues. *European Journal of Operational Research*, 1:396–402, 1977.

[40] W.K. Grassmann. Means and variances of time averages in Markovian environments. *European Journal of Operational Research*, 31:132–139, 1987.

[41] W.K. Grassmann. Numerical solutions for Markovian event systems. In Kall *et al.*, editor, *Quantitative Methoden in den Wirtschaftswissenschaften*, pages 73–87. Springer, 1989.

[42] W.K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In *The First International Conference on the Numerical Solution of Markov Chains*, pages 375–395, 1990.

[43] D. Gross and D.R. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov processes. *Operations Research*, 32(2):343–361, 1984.

[44] B.R. Haverkort. *Performability Modeling Tools, Evaluation Techniques and Applications.* PhD thesis, University of Twente, the Netherlands, 1990.

[45] B.R. Haverkort and I.G. Niemegeers. Performability modelling using dynamic queueing networks. *Performance Evaluation Review*, 17:225, 1989.

[46] B.R. Haverkort and I.G. Niemegeers. On the mutual performance-dependability influence in dynamic queueing networks. In *Proceedings of the First International Workshop on Performability Modelling of Computer and Communication Systems*, pages 33–40, 1991.

[47] B.R. Haverkort, I.G. Niemegeers, and P. Veldhuyzen van Zanten. DyQNtool - a performability modelling tool based on the dynamic queueing network concept. In *Proceedings of the Fifth International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 1991.

[48] D.P. Heyman and M.J. Sobel. *Stochastic Models in Operations Research. Volume I.* McGraw-Hall, 1982.

[49] ISO/TC97/SC21/WG1/DIS9074. *Estelle - A Formal Description Technique Based on an Extended State Transition Model.* ISO, 1987.

[50] B.R. Iyer. Recent results in performability analysis. In Y. Yemini, editor, *Current Advances in Distributed Computing and Communications*, pages 50–64. Computer Science Press, 1987.

[51] B.R. Iyer, L. Donatiello, and P. Heidelberger. Analysis of performability for stochastic models of fault-tolerant systems. *IEEE Trans. on Computers*, C-35(10):902–907, 1986.

[52] A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skandinavsk Aktuarietidskrift*, 36:87–91, 1953.

[53] A.M. Johnson Jr. and M. Malek. Survey of software tools for evaluating reliability, availability and serviceability. *ACM Computing Surveys*, 20:227–271, 1988.

[54] J. Keilson. *Markov Chain Models - Rarity and Exponentiality*. Springer-Verlag, 1979.

[55] J.G. Kemeny and J.L. Snell. *Finite Markov Chains, 2nd Edition*. Springer-Verlag, 1987.

[56] 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. In *Proceedings of FTCS-16*, pages 252–257, 1986.

[57] V.G. Kulkarni, V.F. Nicola, and K.S. Trivedi. On modelling the performance and reliability of multimode computer systems. *Journal of Systems and Software*, 6(1,2):175–182, 1986.

[58] V.O.K. Li and J.A. Silvester. Performance analysis of networks with unreliable components. *IEEE Trans. on Communications*, COM-32(10):1105–1110, 1984.

[59] S.V. Makam and A. Avizienis. ARIES 81: a reliability and life-cycle evaluation tool. In *Proceedings of FTCS-12*, pages 276–274, 1982.

[60] R.A. Marie, A.L. Reibman, and K.S. Trivedi. Transient analysis of acyclic Markov chains. *Performance Evaluation*, 7(3):175–194, 1987.

[61] J. McGough. Effects of near-coincident faults in multiprocessor systems. In *Proceedings of the 5th Annual AIAA/IEEE Digital Avionics Systems Conference*, pages 16.6.1–16.6.7, 1983.

[62] J. McGough, M. Smotherman, and K.S. Trivedi. The conservativeness of reliability estimates based on instantaneous coverage. *IEEE Trans. on Computers*, C-34(7):602–609, 1985.

[63] P. Mejia Ochoa and E. de Souza e Silva. Dynamic state exploration in Markovian models. Technical report, Federal University of Rio de Janeiro, NCE, 1991.

[64] P. Mejia Ochoa and E. de Souza e Silva. Performance evaluation of distributed systems specified in Estelle. Technical report, Federal University of Rio de Janeiro, NCE, 1991.

[65] B. Melamed and M. Yadin. Randomization procedures in the computation of cumulative-time distributions over discrete state Markov processes. *Operations Research*, 32(4):926–944, 1984.

[66] J.F. Meyer. Computation-based reliability. *IEEE Trans. on Computers*, C-25(6):578–584, 1976.

[67] J.F. Meyer. On evaluating the performability of degradable computing systems. In *Proceedings of FTCS-8*, pages 44–49, 1978.

[68] J.F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Trans. on Computers*, C-29(8):720–731, 1980.

[69] J.F. Meyer. Closed-form solutions of performability. *IEEE Trans. on Computers*, C-31(7):648–657, 1982.

[70] J.F. Meyer. A unified approach for specifying measures of performance, dependability, and performability. In J. Laprie, editor, *Vol. 4: Dependable Computing and Fault-Tolerant Systems*. Springer-Verlag, 1990.

[71] J.F. Meyer, A. Movaghar, and W.H. Sanders. Stochastic activity networks: Structure, behavior, and application. In *Proceedings of the International Workshop on Timed Petri-nets*, pages 106–115, 1985.

[72] D.R. Miller. Reliability calculation using randomization for Markovian fault-tolerant computing systems. In *Proceedings of FTCS-13*, pages 284–289, 1983.

[73] M. Molloy. Performance analysis using stochastic Petri nets. *IEEE Trans. on Computers*, C-31(9):913–917, 1982.

[74] A. Movaghar and J.F. Meyer. Performability modeling with stochastic activity networks. In *Proceedings of the 1984 Real-Time Systems Symp.*, pages 215–224, 1984.

[75] R.R. Muntz, E. de Souza e Silva, and A. Goyal. Bounding availability of repairable computer systems. *IEEE Trans. on Computers*, 38(12):1714–1723, 1989.

[76] R.R. Muntz and J.C.S. Lui. Evaluating bounds on steady state availability from Markov models of repairable systems. In *The First International Conference on the Numerical Solution of Markov Chains*, pages 469–489, 1990.

[77] S. Natkin. *Réseaux de Petri Stochastiques*. PhD thesis, CNAM-Paris, Paris, 1980.

[78] V.F. Nicola. Lumping in Markov reward processes. Technical report, IBM Research Report RC 14719, Yorktown Heights, N. Y., 1989.

[79] V.F. Nicola, A. Bobbio, and K.S. Trivedi. A unified performance reliability analysis of a system with a cumulative down time constraint. Technical report, IBM Research Report RC 15279, Yorktown Heights, N. Y., 1989.

[80] T.W. Page Jr., S.E. Berson, W.C. Cheng, and R.R. Muntz. An object-oriented modeling environment. *ACM Sigplan Notices (Proceedings OOPSLA'89)*, 24(10):287–296, 1989.

[81] K.R. Pattipati and S.A. Shah. On the computational aspects of performability models of fault-tolerant computer systems. *IEEE Trans. on Computers*, 39(6):832–836, 1990.

[82] J.L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.

[83] P.S. Puri. A method for studying the integral functionals of stochastic processes with applications: I. Markov chain case. *Journal of Applied Probability*, 8(10):331–343, 1971.

[84] A. Reibman, K. Trivedi, S. Kumar, and G. Ciardo. Analysis of stiff Markov chains. *ORSA Journal on Computing*, 1(2):126-133, 1989.

[85] S.M. Ross. *Stochastic Processes*. John Wiley & Sons, 1983.

[86] R.A. Sahner and K.S. Trivedi. Performance and reliability analysis using directed acyclic graphs. *IEEE Trans. on Software Engineering*, SE-13(10):1105-1114, 1987.

[87] R.A. Sahner and K.S. Trivedi. Reliability modeling using SHARPE. *IEEE Trans. on Reliability*, R-36(2):186-193, 1987.

[88] W.H. Sanders and J.F. Meyer. METASAN: a performability evaluation tool based on stochastic activity networks. In *Proceedings of the 1986 Fall Joint Computer Conference*, pages 807-816, 1986.

[89] W.H. Sanders and J.F. Meyer. Reduced base model construction methods for stochastic activity networks. *IEEE Journal on Selected Areas in Communications*, 9(1):25-36, 1991.

[90] H.A. Simon and A. Ando. Aggregation of variables in dynamic systems. *Econometrica*, 29:111-138, 1961.

[91] R.M. Smith, K.S. Trivedi, and A.V. Ramesh. Performability analysis: Measures, an algorithm and a case study. *IEEE Trans. on Computers*, 37(4):406-417, 1988.

[92] U. Sumita, J.G. Shanthikumar, and Y. Masuda. Analysis of fault tolerant computer systems. *Microelectron. Reliab.*, 27(1):65-78, 1987.

[93] K.S. Trivedi, J.B. Dugan, R.M. Geist, and M.K. Smotherman. Hybrid reliability modeling of fault-tolerant computer systems. *Comput. Elec. Eng.*, 11:87-108, 1984.

[94] N. van Dijk. The importance of bias-terms for error bounds and comparison results. In *The First International Conference on the Numerical Solution of Markov Chains*, pages 640-663, 1990.

[95] N. van Dijk. Transient error bound analysis for continuous time Markov reward structures. *Performance Evaluation*, 1991 (to appear).

[96] H. Weisberg. The distribution of linear combinations of order statistics from the uniform distribution. *Annals Math. Stat.*, 42:704-709, 1971.

[97] C.-L. Yang and P. Kubat. Efficient computation of most probable states for communication networks with multimode components. *IEEE Trans. on Communications*, 37(5):535-538, 1989.

Copies may be requested from:

IBM Thomas J. Watson Research Center
Distribution Services F-11 Stormytown
Post Office Box 218
Yorktown Heights, New York 10598