From *Proceedings of the 26th Annual International Symposium on Fault-Tolerant Computing*, Sendai, Japan, June 25-27, 1996, pp. 116-125

A NEW METHODOLOGY FOR CALCULATING DISTRIBUTIONS OF REWARD ACCUMULATED DURING A FINITE INTERVAL

M. Akber Qureshi and William H. Sanders

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

{qureshi, whs}@crhc.uiuc.edu

Abstract

Markov reward models are an important formalism by which to obtain dependability and performability measures of computer systems and networks. In this context, it is particularly important to determine the probability distribution function of the reward accumulated during a finite interval. The interval may correspond to the mission period in a mission-critical system, the time between scheduled maintenances, or a warranty period. In such models, changes in state correspond to changes in system structure (due to faults and repairs), and the reward structure depends on the measure of interest. For example, the reward rates may represent a productivity rate while in that state, if performability is considered, or the binary values zero and one, if interval availability is of interest. This paper presents a new methodology to calculate the distribution of reward accumulated over a finite interval. In particular, we derive recursive expressions for the distribution of reward accumulated given that a particular sequence of state changes occurs during the interval, and we explore paths one at a time. The expressions for conditional accumulated reward are new and are numerically stable. In addition, by exploring paths individually, we avoid the memory growth problems experienced when applying previous approaches to large models. The utility of the methodology is illustrated via application to a realistic fault-tolerant multiprocessor model with over half a million states.
Keywords: Markov Reward Models, Performability, Interval Availability.

1 Introduction

Performability evaluation is an important approach for calculating the performance of a dependable computing system or network, taking into account changes in performance due to faults. Many methods have been proposed to evaluate system performability, but one of the most popular has been through the use of reward models [1, 2]. In this context, it is important to determine the probability distribution function of reward accumulated during a finite interval. The interval may correspond to the mission period in a mission-critical system, the time between scheduled maintenances, or a warranty period.

Most early work in this regard has been limited to acyclic Markov reward models (see, for example, [2, 3, 4]). Determining the distribution of reward accumulated over a finite interval for general (possibly cyclic) Markov reward models is more difficult due to the possible presence of an infinite number of paths. The first attempt to solve such models was made by Kulkarni et al. [5]. They numerically inverted an expression obtained in the transform domain to obtain a solution. Later Smith et al. [6] presented an improved version of the algorithm in [5], with a computational complexity of $O(n^3)$, where n is the number of states. In [7], Pattipati et al. applied partial differential equation techniques to compute system performability.

Also notable is the work of de Souza e Silva and Gail, who were the first to present a performability solution based on uniformization [8]. They formulated the probability distribution of reward accumulated over a finite interval by first conditioning on the number of transitions n of a uniformized process and then further conditioning on vectors corresponding to the number of visits to states with different rewards, given n. This algorithm was limited in applicability since the storage and computational complexity was combinatorial with the number of distinct rewards. Two attempts have been made to deal with this complexity. In particular, Donatiello and Grassi [9] presented an algorithm with a polynomial complexity in number of distinct rewards by combining uniformization and Laplace transform methods. In addition, in [10], de Souza e Silva and Gail presented a significant improvement on their previous algorithm, albeit limited to rate-based reward models. The storage complexity of their new algorithm is independent of the number of distinct rewards assigned to the states of the Markov process.

In spite of these significant advances, the developed algorithms have been limited to solving small models, and for short time intervals. The reasons are twofold: 1) the storage complexity is still polynomial with number of states and transitions of the uniformized process, and 2) in addition to the stated storage complexity, the algorithm requires the storage of the state transition matrix of the entire subordinated Markov process. When state spaces are large (as is often the case) or the interval is reasonably long (again, as is typical), these storage requirements make the computation of distribution of accumulated reward over a finite interval very difficult.

In this paper, we present a path-based approach (used previously to solve dependability models [11, 12]) to compute the distribution of time-averaged reward accumulated over a finite interval for general (possibly cyclic) reward models. In doing so, we trade storage complexity for time complexity. We show that the trade is advantageous if the probability mass is concentrated on a reasonable number of paths (up to tens of millions). Moreover, to use the path-based approach effectively, we present a numerically stable and computationally efficient algorithm to compute the conditional distribution of accumulated reward given a path. This formulation is new and not based on the often used Weisberg result [13]. Finally, we illustrate the usefulness of the path-based approach by computing the distribution of time-averaged accumulated reward over a finite interval of a highly redundant fault-tolerant multiprocessor system. The new approach presented in this paper is significant in that it can be applied to much larger systems than previously possible, and it is numerically stable.

2 Background

Uniformization (also known as Jenson's method and randomization) is a well-known method for computing the state-occupancy probabilities of a Markov process at specific time t. Its formulation involves construction of a discrete-time Markov process and Poisson process from an original continuous-time process. In particular, consider a continuous-time time-homogeneous Markov process $\{X_t : t \geq 0\}$ with generator matrix A and initial state distribution vector π_0. Let λ be greater than or equal to the maximum departure rate from any state in the process, and $\{Z_n : n \in N\}$ be a discrete-time time-homogeneous Markov chain defined on the same state space as $\{X_t : t \geq 0\}$ with single step transition matrix $P = A/\lambda + I$. Furthermore, let $\{N_t : t \geq 0\}$ be a Poisson process with rate λ. Then π_t, the row vector of state occupancy probabilities at time t, can be expressed as

$$\pi_t = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \pi_0 P^n.$$

de Souza e Silva and Gail [8, 14] formulated the probability distribution function (PDF) of the accumulated reward averaged over a finite interval of time using uniformization and several additional properties about a Markov chain subordinated to a Poisson process. Specifically, consider that the Poisson process has arrivals at instances $T_1 < T_2 < \ldots < T_n$ in an interval $(0, t)$. These are the instances when the subordinated Markov chain makes transitions. Furthermore, consider n independent random variables U_1, $U_2, \ldots U_n$ uniformly distributed on the interval $(0, 1)$. Let $U_{(1)}$, $U_{(2)}$, $\ldots$, $U_{(n)}$ be their order statistics. It is well-known that the joint distribution of the transition times of subordinated Markov chain, given n transitions of the Poisson process in an interval $(0, t)$, is identical to the joint distribution of the order statistics of n uniformly distributed random variables over the interval $(0, t)$. Moreover, it can easily be shown that tU_i, the product of the interval t and the random variable U_i, is uniformly distributed over an interval $(0, t)$. Therefore

$$\begin{aligned} P\{T_1 \leq t_1, T_2 \leq t_2, \ldots, T_n \leq t_n\} &= \\ P\{tU_{(1)} \leq t_1, tU_{(2)} \leq t_2, \ldots, tU_{(n)} \leq t_n\}, \end{aligned} \tag{1}$$

for $t_1 < t_2 < \ldots < t_n$.

Given n transitions of the Poisson process, each path of the subordinated Markov chain consists of $n + 1$ sojourn times. Using the result from Equation (1), these $n + 1$ sojourn times Y_i can be represented as $Y_1 = tU_{(1)}$, $Y_2 = t(U_{(2)} - U_{(1)})$, $\ldots$, $Y_{n+1} = t(1 - U_{(n)})$. It is also known that the random variables Y_i, $i = 1, 2, \ldots, n+1$, are exchangeable. In particular, exchangeability says that

$$\begin{aligned} P\{Y_1 \leq t_1, Y_2 \leq t_2, \ldots, Y_{n+1} \leq t\} &= \\ P\{Y_{j_1} \leq t_1, Y_{j_2} \leq t_2, \ldots, Y_{j_{n+1}} \leq t\}, \end{aligned}$$

for all permutations of j_i's from $1, 2, \ldots, n+1$. Therefore, by using order statistics, the sequence of sojourn times in a state trajectory can be altered. In particular, suppose all paths with n transitions of the uniformized process are divided into sets such that each path in a set has an equal number of visits to states with identical rate rewards. Then exchangeability enables us to describe all paths within each set by a vector $\mathbf{k} = (k_1, k_2, \ldots, k_{K+1})$, where $K + 1$ is the number of distinct rate rewards and k_i, $i = 1, 2, \ldots, K + 1$, is equal to the number of visits to states with rate reward i. (Note $|\mathbf{k}| = n + 1$.)

Using these ideas, de Souza e Silva and Gail formulated the PDF of reward accumulated over a finite interval by further conditioning on the vectors $\mathbf{k}$ possible for each n. Accordingly, $P\{AR(t) \leq r\}$, the distribution of the time-averaged reward accumulated over a finite interval, was expressed as

$$\begin{aligned} P\{AR(t) \leq r\} = \sum_{n=0}^{\infty} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k}} P\{\mathbf{k} \mid n\} \times \\ P\{AR(t) \leq r \mid n, \mathbf{k}\}, \end{aligned} \tag{2}$$

where $P\{\mathbf{k} \mid n\}$ is the probability of vector $\mathbf{k}$ given n transitions.

The exact solution for $P\{AR(t) \leq r\}$ is not possible in this formulation, since the first summation in (2) is over an infinite set. However, as shown in [8, 14], a solution with error bound can be computed by truncating the first summation to some finite number N.

In particular, to a certain error bound, the probability distribution of time-averaged reward accumulated over a finite interval can be formulated as

$$\begin{aligned} P\{AR(T) \leq r\} \;=\; & \sum_{n=0}^{N} \frac{e^{-\lambda t}(\lambda t)^n}{n!} \sum_{\forall \mathbf{k}} P\{\mathbf{k} \mid n\} \times \\ & P\{AR(T) \leq r \mid n, \mathbf{k}\}. \end{aligned} \tag{3}$$

For Markov processes with large state spaces, second summation is challenging to compute due, primarily, to the computation of $P\{\mathbf{k} \mid n\}$. This computation requires knowledge of all $\mathbf{k}$ vectors that are possible given $n-1$ transitions of the uniformized process and the probability of occurrence of paths within these sets of vectors. Since there can be $\begin{pmatrix} K+n+1 \\ n+1 \end{pmatrix}$ $\mathbf{k}$ vectors for n transitions and the number of paths which can generate a vector $\mathbf{k}$ can be equal to the size of the state space, the task of computing PDF for large state spaces becomes immensely difficult. Moreover, this approach also requires the complete generation of the state space prior to starting the PDF computation. Generation of large state spaces in itself is an intensively memory complex problem.

In [10], de Souza e Silva and Gail improved on their earlier algorithm by finding a recursion which, for a given number of transitions, depends on sets of vectors $\mathbf{k}$. Instead of computing path probabilities for each vector $\mathbf{k}$, they directly computed the conditional distribution given n for sets of $\mathbf{k}$ vectors. Their new algorithm significantly improved the memory complexity since individual $\mathbf{k}$ vectors no longer need to be computed. However, due to the recursion on sets of $\mathbf{k}$ vectors, their algorithm needs a separate computation for every point on the probability distribution curve. When both rate and impulse rewards are used, their algorithm has $O(dMN^2\kappa(N,r))$ storage requirement [15], where d is the average number of non-zero entries in the transition matrix, M is the state-space size, N is the truncation point of the infinite series, and $\kappa(N,r)$ is the number of distinct values obtained by adding any combination of N impulse rewards that are less than r. Furthermore, their algorithm operates on the state transition matrix of the subordinated Markov process, which also must be stored. While the new algorithm takes significantly less memory than the original algorithm, it still requires prohibitively large amounts of memory for realistically sized models, which may have hundreds of thousands of states. The next section presents a formulation that avoids this problem, albeit at the cost of additional time, and avoids numerical difficulties in the formulation presented in [8].

3 Path-Based Algorithm

In this new formulation, we compute the probability distribution function of the time-averaged reward accumulated over a finite interval of time by conditioning on possible paths, rather than $\mathbf{k}$ vectors, that can occur. These paths are generated in a depth-first search manner, from a higher-level description, thus avoiding the memory complexity problems associated with previous approaches. However, this gain is not free, since the approach results in an increased time complexity. In the worst case, the number of paths of the uniformized process that must be generated grows exponentially with an increase in the truncation point. However, if the uniformized process has a sparse state-transition probability matrix or is such that the probability mass is concentrated on a reasonable number of the many paths, then the depth-first search approach becomes a reasonable choice.

In this case, we can compute the solution with a reasonable error bound by only considering the set of probable paths. Moreover, if we know the highest rate of the underlying Markov process (as would be the case if it were generated from a higher-level formalism), then the depth-first search approach does not even require generation of the state space prior to the solution. In this case, the paths of the uniformized process can be explored without generating the complete state space, and the solution for the time-averaged accumulated reward can be obtained on the fly.

The depth-first search approach for computing the PDF of the time-averaged reward accumulated over a finite interval is based on exploring paths of the uniformized process and computing conditional distribution given the path. To understand this approach, we first look at a path in the uniformized process. A path in the uniformized process corresponds to a sequence of states through the state space. Let $< s_0, s_1, \ldots, s_n >$ be a sequence in the uniformized process that occurs in time $(0,t)$. The probability of its corresponding path in time $(0,t)$ can be obtained by computing $P\{s_0, s_1, \ldots, s_n \mid n\} = p(s_0,s_1) \times p(s_1,s_2) \times \ldots p(s_{n-1},s_n)$, and $P\{n\}$,

$$\begin{aligned} P\{path\} &= P\{s_0, s_1, \ldots, s_n \;\wedge\; n\} \\ &= P\{n\} P\{s_0, s_1, \ldots, s_n \mid n\} \end{aligned}$$

where $p(s_i, s_j)$ is the transition probability from state s_i to s_j in the discrete time embedded process.

For the path-based approach, the distribution of the time-averaged reward accumulated over a finite interval can be expressed as

$$P\{AR(t) \leq r\} = \sum_{path \in P} P\{path\} P\{AR(t) \leq r | path\}, \tag{4}$$

where P is the (possibly infinite) set of possible paths in the process.

As with the formulation in (2), an exact solution of (4) is not possible because there are an infinite number of paths in a uniformized process. Since our intent is to only consider paths that are significant relative to the solution, we limit the number of paths considered by discarding those whose path probabilities are smaller than a specified value, defined as w. Let P_w denote the set of paths that have path probabilities greater than or equal to w. Accordingly, PDF can be expressed as

$$P\{AR(t) \leq r\} = \sum_{path \in P_w} P\{path\} P\{AR(t) \leq r | path\}. \quad (5)$$

As with the previous approach, considering a finite number of paths results in an error in the solution which can be bounded. In particular, let $E(w)$ be the error induced by discarding paths for which $P\{path\} < w$ and $0 \leq w \leq 1$. In order to bound $E(w)$, we first compute $E(path)$, the error produced by discarding a particular path. Let this path consist of the sequence $< s_0, s_1, \ldots, s_n >$. Note that by discarding a path, we are also discarding all longer paths which have states $s_0, s_1, \ldots, s_n$ as their first n states. We first look at the error induced by discarding the path of length n. In particular, the error will be

$$\begin{aligned} E(path) \; = \; & \frac{e^{-\lambda t}(\lambda t)^n}{n!} P\{s_0, s_1, \ldots, s_n \mid n\} \times \\ & P\{AR(t) \leq r | path\}. \end{aligned}$$

Since the conditional distribution $0 \leq P\{AR(t) \leq y | path\} \leq 1 \; \forall y$, we can bound the error by assuming it is equal to 1. This implies that

$$E(path) \; \leq \; \frac{e^{-\lambda t}(\lambda t)^n}{n!} P\{s_0, s_1, \ldots, s_n \mid n\}.$$

Now, since the entries in a row of a transition matrix sum to 1, the sum of the probabilities of all paths of length $n+1$ discarded due to the discarding of the path with state sequence $< s_0, s_1, \ldots, s_n >$ can be bounded by

$$\begin{aligned} & \sum_{\forall s_{n+1}} P\{s_0, s_1, \ldots, s_{n+1} \mid n+1\} \times \frac{e^{-\lambda t}(\lambda t)^{n+1}}{(n+1)!} \; = \\ & \quad P\{s_0, s_1, \ldots, s_n \mid n\} \frac{e^{-\lambda t}(\lambda t)^{n+1}}{(n+1)!}. \end{aligned}$$

Using the above argument repeatedly, the total error (denoted E^*) induced by discarding all paths of length n or longer with starting states $s_0, s_1, \ldots s_n$ can be bounded by

$$\begin{aligned} E^*(path) \; \leq \; & P\{s_0, s_1, \ldots, s_n \mid n\} \sum_{i=n}^{\infty} \frac{e^{-\lambda t}(\lambda t)^i}{i!} \\ = \; & P\{s_0, s_1, \ldots, s_n \mid n\} \times \\ & \left(1 - \sum_{i=0}^{n-1} \frac{e^{-\lambda t}(\lambda t)^i}{i!}\right). \end{aligned}$$

Hence a bound on the total error induced by discarding paths can be computed as

$$E(w) \; \leq \sum_{path \in P^D(w)} E^*(path),$$

where $P^D(w)$ is the set of paths discarded, during exploration, since they do not meet the choosen weight criteria.

4 Calculation of Conditional Distribution

Given the approach of the previous section, we need a way to compute the distribution of time-averaged reward accumulated over an interval of time conditioned on a particular path. Finding an efficient and numerically stable way to compute this distribution is the key to the development of a methodology that can be applied to realistically sized systems. Since our process is uniformized with paths that are exchangeable (see Section 2), the conditional accumulated reward will be the same for all paths that have the same number of visits to states with particular rate rewards. In other words, for all paths with $\mathbf{k}$ vector $\mathbf{k} = k_1, k_2, \ldots, k_{K+1}$ such that $|\mathbf{k}| = n+1$,

$$P\{AR(t) \leq r \mid path\} \; = \; P\{AR(t) \leq r \mid n, \mathbf{k}\}.$$

This suggests that, in principle, we could use the result from Weisberg [13], which gives the probability distribution function of a linear combination of selected order statistics of i.i.d. uniform random variables. Unfortunately, while this is correct, it is numerically unstable for practical problems, since for large n, it requires the subtraction of extremely large numbers (generated from factorials of large numbers) with nearly identical magnitude at multiple points in the algorithm. Multiple precision arithmetic could potentially be used to avoid this problem, but it is extremely difficult to know what precision should be used, because of the multiple subtractions.

To avoid these problems, we have developed a new algorithm that does not make use of the Weisberg result. It is based on an alternative formulation given in [16] and makes use of three new lemmas (presented in the following) that reduce both the amount of computation and the magnitude of numbers that must be stored as intermediate results. Furthermore, we are able to formulate the expression in a form that requires very few subtractions of numbers with large but nearly identical magnitude and, hence, are able to determine exactly the number of digits required to achieve a desired precision in the result.

In particular, the new formulation is based on expressing the problem as the linear combination of order statistics, rewriting these as a linear combination of Drichlet random variables, and then computing the distribution of the combination. (The Drichlet distribution is widely used in statistical mathematics; see [17] for example.) Specifically, consider, as was done previously, that $K+1$ different rate rewards are assigned to the states of the uniformized process. Furthermore, suppose that rate rewards are ordered such that

$$r_1 > r_2 > \ldots > r_{K+1} \geq 0.$$

Then define l_i to be the sum of the lengths of sojourn times with rate reward r_i. After doing this, the conditional averaged accumulated reward, $AR(t)$, given n and $\mathbf{k}$, can be expressed as

$$AR(t \mid n, \mathbf{k}) \; = \; \frac{1}{t} \sum_{i=1}^{K+1} r_i \times l_i.$$

Our problem is to compute the probability distribution of the conditional averaged accumulated reward variable, i.e.

$$P\{\frac{1}{t}\sum_{i=1}^{K+1} r_i \times l_i \leq r\}$$

which can also be expressed as

$$P\{\frac{1}{t}\sum_{i=1}^{K+1} b_i \times l_i \leq b\},$$

where $b_i = r_i - r_{K+1}$ and $b = r - r_{K+1}$. Using few results from [16] and some advanced calculus, the complementary distribution function of $AR(t)$, given $\mathbf{k}$ and n, denoted $(\overline{F}_{AR(t)}(b \mid n, \mathbf{k}))$, can be formulated as (for details see [18])

$$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) = \prod_{a=1}^{K} b_a^{-k_a} \sum_{l=1}^{x}\sum_{m=1}^{k_l} C_{l,m} \times b_l^m \times \left(1 - \left(\frac{b}{b_l}\right)^m \frac{H[m, m-n, m+1, (b/b_l)]}{m\ B(m, n-m+1)}\right) \quad (6)$$

where,

- x is the number of shifted rate rewards greater than b, i.e., $b_1 > b_2 > \cdots > b_x > b$,
- $B(i,j) = \frac{\Gamma(i)\,\Gamma(j)}{\Gamma(i+j)}$,
- $H[m, m-n, m+1, (b/b_l)]$ is a hyper-geometric function defined by the following series,

$$1 + \sum_{u=1}^{\infty} \frac{m(m+1)\ldots(m+u-1)\ (m-n)}{u!\ (m+1)} \times \frac{(m-n+1)\ldots(m-n+u-1)}{(m+2)\ldots(m+u)} \left(\frac{b}{b_l}\right)^n, \text{ and} \quad (7)$$

- $C_{l,m}$ are constant coefficients for the partial fraction of

$$G(s) = \frac{1}{(s+\beta_1)^{k_1}(s+\beta_2)^{k_2}\ldots(s+\beta_k)^{k_K}},$$

  where $\beta_i = (1/b_i)$, $i = 1, 2, \ldots, K$, are zeros of $1/G(s)$ and k_i, $i = 1, 2, \ldots, K$, denote their order.

By looking at (6), one can easily realize that its computation by straightforward means, like the Weisberg result, is susceptible to numerical errors. The main reason for these errors is the computation of factorials in computing $H[m, m-n, m+1, (b/b_l)]$, $B(m, n-m+1)$, and $C_{l,m}$. These expressions result in extremely large numbers which when subtracted using normal floating point arithmetic may introduce significant loss of precision. However, with appropriate manipulation, and selective use of extended precision math, these problems can be avoided. In the remainder of this section, we derive three lemmas, which show how to derive a computationally efficient and numerically stable means to compute $\overline{F}_{AR(t)}(b \mid n, \mathbf{k})$.

For simplicity, in the remaining discussion, let

$$F_l(n,m) = \left(1 - \left(\frac{b}{b_l}\right)^m \frac{H[m, m-n, m+1, (b/b_l)]}{m\ B(m, n-m+1)}\right).$$

Accordingly,

$$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) = \prod_{a=1}^{K} b_a^{-k_a} \sum_{l=1}^{x}\sum_{m=1}^{k_l} C_{l,m} \times b_l^m \times F_l(n,m),$$

which after algebraic manipulation can be written as

$$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) = \sum_{l=1}^{x} \prod_{\substack{a=1 \\ a \neq l}}^{K} b_a^{-k_a} \sum_{m=1}^{k_l} \frac{1}{b_l^{(k_l-m)}} \times C_{l,m} \times F_l(n,m).$$

4.1 Computation of $C_{l,m}$

To compute $C_{l,m}$, the constant coefficients of the partial fraction expansion of $G(s)$, we need to compute higher derivatives of the function $G_l(s)$ (the ith derivative of G_l is denoted $G_l^{(i)}$), where

$$G_l(s) \equiv (s+\beta_l)^{k_l} G(s) = \prod_{\substack{a=1 \\ a \neq l}}^{K} \frac{1}{(s+\beta_a)^{k_a}},$$

since

$$\begin{aligned} C_{l,m} &= \lim_{s \to (-\beta_l)} \frac{1}{(k_l-m)!}\ G_l^{(k_l-m)}(s) \\ &\equiv \frac{1}{(k_l-m)!}\ G_l^{(k_l-m)}(-\beta_l). \end{aligned} \quad (8)$$

To compute $C_{l,m}$, we use Bell polynomials [19], the higher derivatives of a composition of two functions. In particular, let $h = f \circ g$ be the composition of f with g, that is, $h(t) = f(g(t))$. Denote n-fold application of an operator d/dt by $d^{(n)}/dt^{(n)}$. Furthermore, denote $h_n = d^{(n)}h/dt^{(n)}$, $f_n = d^{(n)}f(g)/dg^{(n)}$, and $g_n = d^{(n)}g/dt^{(n)}$. Assume that the functions f, g, and h have derivatives of all the orders. Then the Bell polynomials can be expressed as follows:

$$\begin{aligned} h_1 &= f_1 g_1 \\ h_2 &= f_2 g_1^2 + f_1 g_2 \\ h_3 &= f_3 g_1^3 + f_2(3 g_1 g_2) + f_1 g_3 \\ &\vdots \end{aligned}$$

In general, $h_q = \sum_{p=1}^{q} f_p \alpha_{p,q}$, where α_{pq}'s are polynomials in g_i's that do not depend upon the choice of f. The computation of $\alpha_{p,q}$ for large values of q makes the computation of higher derivatives of a composition of two functions impractical. However, if the function $h = f(g)$ is defined such that all its derivatives with respect to g are identical, then an efficient recursive expression can be written to compute the higher derivatives. This fact is expressed formally in the following lemma, which due to lack of space is stated without proof.

Lemma 1 *Let $h = f(g)$ be a function such that $h = f_1 = f_2 = \ldots$. Then h_q, for some $q \geq 1$, can be computed as*

$$h_q = \sum_{i=1}^{q-1} \binom{q-1}{q-i} g_i h_{q-i} + g_q h \ .$$

Proof: See [18].

We can use Lemma 1 to efficiently compute the higher derivatives of $G_l(-\beta_l)$. To do this, we define $f(s) = \exp(s)$, $h(s) = f(g(s))$, and

$$g(s) \ = \ - \sum_{\substack{a=1 \\ a \neq l}}^{K} k_a \ln(s + \beta_a).$$

Note that $h(s) = \exp(g(s)) = G_l(s)$. Furthermore, observe that

$$\frac{d^{(p)}}{dg} G_l(g(s)) = \ \exp(g(s)), \text{ and}$$
$$G_l(-\beta_l) = \prod_{\substack{a=1 \\ a \neq l}}^{K} (\beta_a - \beta_l)^{-k_a}, \qquad (9)$$

for $p \geq 1$. Using Lemma 1, $G^{(k_l-m)}(-\beta_l) = h_{k_l-m}$ can then be expressed as

$$G_l^{(k_l-m)}(-\beta_l) \ = \ \sum_{i=1}^{k_l-m-1} \binom{k_l-m-1}{k_l-m-i} g_i G_l^{(k_l-m-i)}(-\beta_l) + g_{k_l-m} G_l(-\beta_l), \qquad (10)$$

where

$$g_p \ = \ (-1)^p (p-1)! \sum_{\substack{a=1 \\ a \neq l}}^{K} \frac{k_a}{(\beta_a - \beta_l)^p} \quad \text{for } p \geq 1.$$

By looking at Equation 10 and using an inductive argument, it can be realized that

$$G_l^{(k_l-m)}(-\beta_l) \ = \ G_l(-\beta_l) \times \tilde{G}_l^{(k_l-m)}(-\beta_l), \text{ where} \qquad (11)$$

$$\tilde{G}_l^{(k_l-m)}(-\beta_l) \ = \ \sum_{i=1}^{k_l-m-1} \binom{k_l-m-1}{k_l-m-i} g_i \tilde{G}_l^{(k_l-m-i)}(-\beta_l) + g_{k_l-m}. \qquad (12)$$

Now, using Equations 8 and 11, $C_{l,m}$ can be expressed as

$$C_{l,m} \ = \ \frac{G_l(-\beta_l) \times \tilde{G}_l^{(k_l-m)}(-\beta_l)}{(k_l - m)!}$$

Accordingly,

$$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) \ = \sum_{l=1}^{x} \prod_{\substack{a=1 \\ a \neq l}}^{K} b_a^{-k_a} \times G_l(-\beta_l) \times \sum_{m=1}^{k_l} \frac{1}{b_l^{(k_l-m)}} \times \frac{\tilde{G}_l^{(k_l-m)}(-\beta_l)}{(k_l-m)!} \times F_l(n, m).$$

Since $\beta_i = (1/b_i)$, using Equation 9, we can write

$$G_l(-\beta_l) \ = \ \prod_{\substack{a=1 \\ a \neq l}}^{K} \left(\frac{b_a b_l}{b_l - b_a} \right)^{k_a}.$$

Finally,

$$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) \ = \sum_{l=1}^{x} \prod_{\substack{a=1 \\ a \neq l}}^{K} \left(\frac{b_l}{b_l - b_a} \right)^{k_a} \times \sum_{m=1}^{k_l} \frac{1}{b_l^{(k_l-m)}} \times \frac{\tilde{G}_l^{(k_l-m)}(-\beta_l)}{(k_l-m)!} \times F_l(n, m).$$

In the remainder of this section, we present two more lemmas, one which gives a recursive expression to compute

$$H_l(m) = \frac{1}{b_l^{(k_l-m)}} \times \frac{\tilde{G}_l^{(k_l-m)}(-\beta_l)}{(k_l-m)!}$$

and another which gives a recursive expression for computing $F_l(n, m)$. Based on these results, we can then present an algorithm to compute $\overline{F}_{AR(t)}(b \mid n, \mathbf{k})$. Proofs of the lemmas are omitted due to space limitations.

4.2 Computation of $H_l(m)$

Lemma 2 *For $1 \leq m \leq k_l - 1$,*

$$H_l(m) \ = \ \frac{1}{k_l - m} \left[\sum_{i=1}^{k_l-m-1} g_i^{''} H_l(m+i) + g_{k_l-m}^{''} \right],$$

where

$$g_i^{''} \ = \ (-1)^i \sum_{\substack{a=1 \\ a \neq i}}^{K} k_a \left(\frac{b_a}{b_l - b_a} \right)^i,$$

and for $m = k_l$, $H_l(m) = 1$.

Proof: See [18].

4.3 Computation of $F_l(n,m)$

Lemma 3 *For $m = 1$,*

$$F_l(n,m) = \left(1 - \frac{b}{b_l}\right)^n,$$

and for $2 \le m \le n$,

$$F_l(n,m) = \binom{n}{m-1}\left(\frac{b}{b_l}\right)^{m-1}\left(1 - \frac{b}{b_l}\right)^{n-(m-1)} + F_l(n,m-1).$$

Proof: See [18].

5 Algorithm to Compute $\overline{F}_{AR(t)}(b \mid n, \mathbf{k})$

Recall that we can express the complementary conditional distribution in terms of $H_l(m)$ and $F_l(n,m)$ as

$$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) = \sum_{l=1}^{x} \prod_{\substack{a=1 \\ a \neq l}}^{K} \left(\frac{b_l}{b_l - b_a}\right)^{k_a} \times \sum_{m=1}^{k_l} H_l(m) \times F_l(n,m).$$

While the recursions developed in the previous section give us an efficient method to calculate $H_l(m)$ and $F_l(n,m)$, $H_l(m)$ will take on extremely large positive and negative values for certain $\mathbf{k}$ and large n. These values can cause loss of precision in a practical implementation of the algorithm. This loss of precision can be avoided, to some extent, by writing a recursion for $H_l(m) \times F_l(n,m)$ directly, which we denote by $X_l(n,m)$. In this notation,

$$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) = \sum_{l=1}^{x} \prod_{\substack{a=1 \\ a \neq l}}^{K} \left(\frac{b_l}{b_l - b_a}\right)^{k_a} \sum_{m=1}^{k_l} X_l(n,m). \tag{13}$$

Then, using Lemma (2), we can write a recursive expression to compute $X_l(n,m)$. Since $\frac{X_l(n,m)}{F_l(n,m)} = H_l(m)$, Lemma 2 says that

$$\frac{X_l(n,m)}{F_l(n,m)} = \frac{1}{(k_l - m)}\left[\sum_{i=1}^{k_l-m-1} g_i^{''} \frac{X_l(n,m+i)}{F_l(n,m+i)} + g_{k_l-m}^{''}\right],$$

for $1 \le m \le k_l - 1$. This implies that

$$X_l(n,m) = \frac{1}{(k_l - m)}\left[\sum_{i=1}^{k_l-m-1} g_i^{''} \times X_l(n,m+i) \times \frac{F_l(n,m)}{F_l(n,m+i)} + g_{k_l-m}^{''} \times F_l(n,m)\right]. \tag{14}$$

Note for $m = k_l$, $X_l(n,m) = F_l(n,m)$, since $H_l(k_l) = 1$. By writing the recursion this way, we avoid the problem with large $H_l(m)$, since the $F_l(n,m)$ are probabilities.

The formulation presented in (13) and (14) diminishes the numerical difficulty in computing $\overline{F}_{AR(t)}(b \mid n, \mathbf{k})$, but still requires careful implementation to avoid loss of precision in the computation. In particular, the formulation calls for the addition of a large number of positive and negative numbers, which will be large in magnitude for certain $\mathbf{k}$ and large n. These numbers may be very close to one another, since when added together they should equal a potentially very small probability. The main loss of precision in such cases is caused by the subtraction of two numbers which are close in value. In this case, a large shift to the left will be made after the operation to normalize the result, and many digits of precision will be lost.

Keeping this in mind, to minimize the number of subtractions, we split $X_l(n,m)$ into positive $X_l^{pos}(n,m) > 0$ and negative $X_l^{neg}(n,m) < 0$ parts such that

$$X_l(n,m) = X_l^{pos}(n,m) + X_l^{neg}(n,m).$$

Similar to the recursion in Equation 14, for $1 \le m \le k_b - 1$, recursions for $X_l^{pos}(n,m)$ and $X_l^{neg}(n,m)$ can be expressed as

$$X_l^{pos}(n,m) = \frac{1}{(k_l - m)}\left[\sum_{i=1}^{k_l-m-1} g_i^{''} X_l^{pos}(n,m+i) \times \frac{F_l(n,m)}{F_l(n,m+i)} I(g_i^{''} > 0) + \sum_{i=1}^{k_l-m-1} g_i^{''} X_l^{neg}(n,m+i) \times \frac{F_l(n,m)}{F_l(n,m+i)} I(g_i^{''} < 0) + I(g_{k_l-m}^{''} > 0) g_{k_l-m}^{''}\right] \tag{15}$$

and

$$X_l^{neg}(n,m) = \frac{1}{(k_l - m)}\left[\sum_{i=1}^{k_l-m-1} g_i^{''} X_l^{neg}(n,m+i) \times \frac{F_l(n,m)}{F_l(n,m+i)} I(g_i^{''} > 0) + \sum_{i=1}^{k_l-m-1} g_i^{''} X_l^{pos}(n,m+i) \times \frac{F_l(n,m)}{F_l(n,m+i)} I(g_i^{''} < 0) + I(g_{k_l-m}^{''} < 0) g_{k_l-m}^{''}\right], \tag{16}$$

where $I(x)$ is an indicator function (returns 1 if x is true and else 0) and for $m = k_l$, $X_l^{pos}(n,m) = F_l(n,m)$ and $X_l^{neg}(n,m) = 0$.

Given that all the additions are performed in ascending order of absolute magnitudes, then, for each shifted reward $b_l > b$, Equations 15 and 16 yield an accurate (to the precision of the computation) positive value and negative value. Both these values are then multiplied by

$$\prod_{\substack{a=1 \\ a \neq l}}^{K} \left(\frac{b_l}{b_l - b_a}\right)^{k_a},$$

which is denoted z in the algorithm below. Then, these two resulting numbers with powers greater than 0 are subtracted to yield a probability. Since the subtraction will result in a probability, the order of magnitude of the two numbers will always be the same (called *magnitude* in the following algorithm). This implies that to obtain an answer accurate to some specified precision, we need a decimal-digit precision in the computation equal to the sum of *magnitude* and the desired precision. If normal floating point math (as per the IEEE standard) does not provide the required precision, the algorithm obtains the required precision by invoking multi-precision routines, rolling back, and recomputing the required terms and sums. Thus, by the use of selective multi-precision, the algorithm always achieves the desired precision and, hence, is numerically stable. Since there is only a single subtraction needed for each shifted rate reward, each summation will at most roll back once, leading to an efficient algorithm.

More precisely, the algorithm to compute $\overline{F}_{AR(t)}$ $(b \mid n, \mathbf{k})$ is as follows. (Note that the given algorithm does not consider impulse rewards. However, an extension of the algorithm to include impulse rewards is straightforward and can be done in a manner similar to one presented by Qureshi and Sanders [20].)

Algorithm 1 *(Compute the complementary conditional distribution, $\overline{F}_{AR(t)}(b \mid n, \mathbf{k})$, using (13) and (14).)*

```
$\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) = 0.$

For $l = 1$ to $x$

    For $m = 1$ to $k_l$
        Compute $F_l(n, m)$ by using recursion given
        in Lemma (3).
    End(For).

    For $m = k_l$ to 1
        If ($m = k_l$)
            $X_l^{pos}(n, k_l) = F_l(n, k_l)$, and
            $X_l^{neg}(n, k_l) = 0.$
        Else
            Use Equation 15 to compute $X_l^{pos}(n, m)$.
            Use Equation 16 to compute $X_l^{neg}(n, m)$.
    End(For).

    Let $X_{negative}$ be the sum of all $X_l^{neg}(n, m)$, added
    in ascending order of their absolute magnitudes.

    Let $X_{positive}$ be the sum of all $X_l^{pos}(n, m)$, added
    in ascending order of their magnitudes.

    $X_{positive} = z \times X_{positive}.$
    $X_{negative} = z \times X_{negative}.$

    Let magnitude be the logarithm (base 10) of
    $X_{positive}$.

    Let needed digits = magnitude + desired precision.

    If (IEEE floating point precision > needed digits)
        $X_{total} = X_{positive} + X_{negative}$
    Else
        Recompute $X_{total}$ by recomputing $l_{th}$ iteration
        of for loop with needed digits precision.

    $\overline{F}_{AR(t)}(b \mid n, \mathbf{k}) = \overline{F}_{AR(t)}(b \mid n, \mathbf{k}) + X_{total}.$

End(For)
```

This algorithm, together with the path-based approach described in Section 3, provide the basis for stable and efficient calculation of the distribution of time-averaged reward accumulated over a finite interval. The overall algorithm to compute the distribution of accumulated reward starts by first generating possible paths and computing their respective probabilities. The probabilities of paths which correspond to identical $\mathbf{k}$ vectors are added together, while the probabilities of paths corresponding to distinct $\mathbf{k}$ vectors are stored separately. This exploration of paths and computation of path probabilities continues until either all paths with probabilities greater than or equal to the discarding weight are explored or the memory used reaches a preset machine-dependent limit. After reaching one of these conditions, the conditional distribution of accumulated reward is computed for each $\mathbf{k}$ vector generated, and after being multiplied by the probability of the explored paths corresponding to the the $\mathbf{k}$ vector, is added together with the results from other paths to form the unconditional distribution. If all paths with probability greater than the threshold have not yet been explored, the process is repeated, exploring more paths and then computing the required conditional distributions.

To test the algorithm on a non-trivial example, we have implemented it in C++, using the multi-precision library LiDIA [21]. The implementation has been used for several large examples and works well when the number of paths required to compute a desired accuracy is not more than tens of millions. One such example, with over one-half million states is given in the next section.

6 Example Study

The applicability of our approach is illustrated by considering the performability analysis of a highly redundant fault-tolerant multiprocessor system from Lee et al. [22]. The system, at the highest level, consists of three non-repairable computers. (Lee et al. considered a variable number of computers.) (Note that our algorithm applies to repairable, as well as non-repairable systems.) As shown in Figure 1, each computer is composed of three memory modules, of

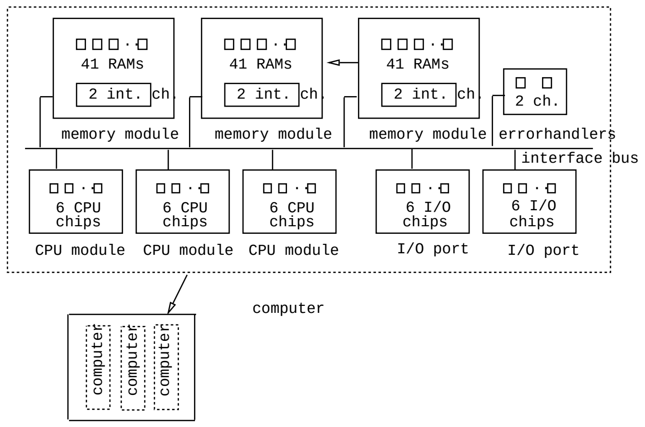

Figure 1: Fault-tolerant parallel computing system

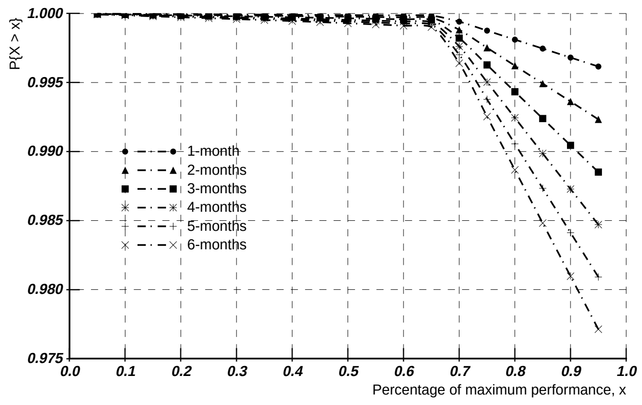

Figure 2: Performability Distribution

which one is a spare; three CPU units, of which one is spare; two I/O ports, of which one is spare; and two non-redundant error handling chips. A computer is classified as operational if at least two memory modules, two CPU units, one I/O port, and the two error-handling chips are working.

Internally, each memory module consists of 41 RAMs, out of which two are spare chips and two are interface chips. Each CPU unit and each I/O port consists of six non-redundant chips. A memory module is classified as operational if at least 39 of its 41 RAM chips and its two interface chips are working. Each component failure has coverage probabilities associated with it. The coverage probabilities are state dependent, and there are multiple levels of coverage (module, computer, multiprocessor). We used the same coverage probabilities and the failure rate for the chips as used in the original model by Lee et al. [22].

The performability measure considered is the complementary PDF of the fraction of jobs completed in the interval, considering faults, relative to the number of jobs that would complete in a fault-free system. More specifically, we consider a system where 300 jobs complete per second if 3 processors are working, 200 jobs complete per second if 2 processors are working, and 100 jobs complete per second if 1 processor is working. The measure, then, is the PDF of the number of jobs that complete divided by $300{*}T$, where T is the length of the interval under consideration.

The model was represented as a stochastic activity network, as was done in [23]. Space does not permit presentation of the SAN model here. The Markov level representation of the model is very large, consisting of 463,268 states even if symmetries are detected in the process. To the best of our knowledge, this is by far the largest and most complicated model that has ever been solved for the distribution of accumulated reward, and it illustrates the practicality of our approach.

We present results regarding both the performability of the considered system and the efficiency of the algorithm in the context of the example. In particular, Figure 2 depicts the complementary distribution of reward accumulated during time intervals of one to six months. Each line on the graph corresponds to a specific time interval, and it gives the probability that the system performs at a level higher than the value given on the x-axis. A discarding threshold value of $w = 10^{-11}$ was used for all time intervals, resulting in an accuracy of more than six digits for each result.

Figure 3 illustrates the effect of changing w on the number of paths considered and the bound on the error obtained due to discarding paths whose probabilities are less than the value w. For this example, we consider a utilization period of 1 year. Since the space required by previous approaches depends critically on

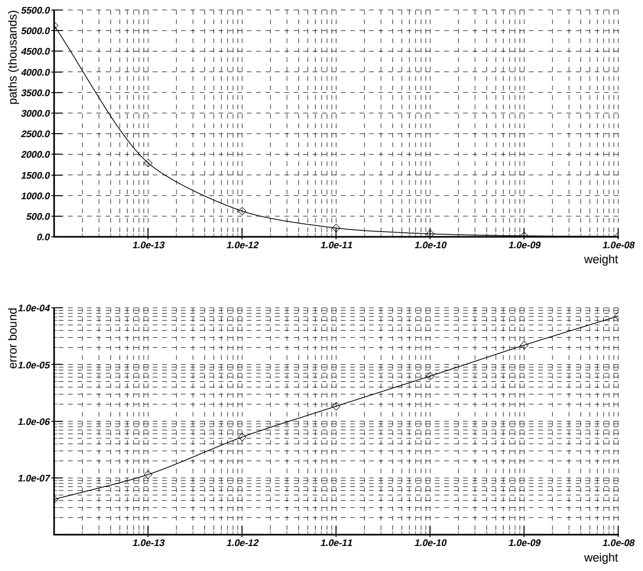

Figure 3: Effect of Discarding of Paths (t = one year)

the length of the interval, this (reasonable) time point would result in an unreasonably large memory requirement for current non-path-based approaches. Note that the number of paths that need to be considered decreases dramatically with an increasing weight, while the error bound achieved remains reasonable. This illustrates the effectiveness of our approach on systems with up to millions of probable paths.

7 Conclusion

In this paper, we have presented a new method for computing the distribution of time-averaged reward accumulated over a finite interval. The method is based on generating paths of the uniformized process which are probable, and computing the probability distribution of accumulated reward conditioned on each path. To make this practical, we have developed a new algorithm for computing the conditional distribution of accumulated reward which is computationally efficient and numerically stable. In addition, we have presented experimental results to show that the path-based approach is useful when solving Markov reward models with large state spaces and up to tens of millions of probable paths. We also have shown that, for the example considered, the number of paths required to obtain a result decreases dramatically as the path discarding threshold is increased while still obtaining reasonable accuracy.

These results bode well for the practical use of reward models in performability modeling.

Acknowledgment

We thank Luai Malhis, who built the SAN representation of the multiprocessor used as our example. Moreover, we thank the anonymous reviewers for their valuable comments.

References

[1] R. A. Howard, *Dynamic Probabilistic Systems (vol. II)*. John Wiley & Sons, 1971.

[2] J. F. Meyer, "Closed-form solutions of performability," *IEEE Transactions on Computers*, vol. 31, no. 7, pp. 648–657, 1982.

[3] L. Donatiello and B. R. Iyer, "Analysis of a composite performance reliability measure for fault-tolerant systems," *Journal of the ACM*, vol. 34, no. 1, pp. 179–199, 1987.

[4] A. Goyal and A. N. Tantawi, "Evaluation of performability for degradable computer systems," *IEEE Transactions on Computers*, vol. 36, no. 6, pp. 738–744, 1987.

[5] 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 16th Int. Symposium on fault-tolerant computing*, (Vienna, Austria), pp. 252–257, July 1985.

[6] R. M. Smith, K. S. Trivedi, and A. V. Ramesh, "Performability analysis: Measures, an algorithm, and a case study," *IEEE Transactions on Computers*, vol. 37, no. 2, pp. 406–417, 1987.

[7] K. R. Pattipati, Y. Li, and H. A. P. Blom, "A unified framework for the performability evaluation of fault-tolerant computer systems," *IEEE Transactions on Computers*, vol. 42, no. 3, pp. 312–326, 1993.

[8] E. de Souza e Silva and H. R. Gail, "Calculating availability and performability measures of repairable computer systems," *Journal of the ACM*, vol. 36, pp. 171–193, January 1989.

[9] L. Donatiello and V. Grassi, "On evaluating the cumulative performance distribution of fault-tolerant computer systems," *IEEE Transactions on Computers*, vol. 40, no. 11, pp. 1301–1307, 1991.

[10] E. de Souza e Silva and R. Gail, "Calculating transient distributions of cumulative reward," in *Proceedings of Sigmetrics/Performance-95*, (Ottawa, Canada), pp. 231–240, May 1995.

[11] R. Marie, A. L. Reibman, and K. S. Trivedi, "Transient analysis of acyclic Markov chains," *Performance Evaluation*, vol. 7, pp. 175–194, 1987.

[12] E. de Souza e Silva and P. M. Ochoa, "State space exploration in Markov models," *Performance Evaluation*, vol. 20, pp. 155–166, 1992.

[13] H. Weisberg, "The distribution of linear combinations of order statistics from the uniform distribution," *The Annals of Mathematical Statistics*, vol. 42, no. 2, pp. 704–709, 1995.

[14] E. de Souza e Silva and H. R. Gail, "Calculating cumulative operational time distributions of repairable computer systems," *IEEE Transactions on Computers*, vol. 35, pp. 322–332, April 1986.

[15] E. de Souza e Silva and R. H. Gail, "An algorithm to calculate transient distributions of cumulative reward," Tech. Rep. CSD-940021, University of California, Los Angeles, May 1994.

[16] T. Matsunawa, "The exact and approximate distributions of linear combinations of selected order statistics from a uniform distribution," *The Annals of the Institute of Statistical Mathematics*, vol. 37, pp. 1–16, 1985.

[17] S. M. Wilks, *Mathematical Statistics*. John Wiley & Sons, Inc., 1963.

[18] M. A. Qureshi, *Construction and Solution of Markov Reward Models*. University of Arizona, 1996.

[19] G. M. Constantine, *Combinatorial Theory and Statistical Design*. John Wiley & Sons, Inc., 1987.

[20] M. A. Qureshi and W. H. Sanders, "Reward model solution methods with impulse and rate rewards: An algorithm and numerical results," *Performance Evaluation*, vol. 20, pp. 413–436, 1994.

[21] V. . LiDIA Manual, "A library for computational number theory," tech. rep., Universitat des Saarlandes, August 1995.

[22] D. Lee, J. Abraham, D. Rennels, and G. Conte, "A numerical technique for the evaluation of large, closed fault-tolerant systems," in *Dependable Computing for Critical Applications*, pp. 95–114, Springer-Verlag, Wien, 1992.

[23] W. H. Sanders and L. M. Malhis, "Dependability evaluation using composed SAN-based reward models," *Journal of Parallel and Distributed Computing*, vol. 15, no. 3, pp. 238–254, 1992.