Rare Event Simulation for Highly Dependable Systems with Fast Repairs

Paper version accepted for the 7th International Conference on Quantitative Evaluation of SysTems (QEST) 2010.

Daniël Reijsbergen[1], Pieter-Tjerk de Boer[1], Werner Scheinhardt[1], and Boudewijn Haverkort[1,2]

[1]Center for Telematics & Information Technology, University of Twente, Enschede, The Netherlands
[2]Embedded Systems Institute, Eindhoven, The Netherlands
{d.p.reijsbergen, p.t.deboer, w.r.w.scheinhardt, b.r.h.m.haverkort}@utwente.nl

***Abstract*—Stochastic model checking has been used recently to assess, among others, dependability measures for a variety of systems. However, the employed numerical methods, as, e.g., supported by model checking tools such as PRISM and MRMC, suffer from the state-space explosion problem. The main alternative is statistical model checking, which uses standard simulation, but this performs poorly when small probabilities need to be estimated. Therefore, we propose a method based on importance sampling to speed up the simulation process in cases where the failure probabilities are small due to the high speed of the system's repair units. This setting arises naturally in Markovian models of highly dependable systems. We show that our method compares favourably to standard simulation, to existing importance sampling techniques and to the numerical techniques of PRISM.**

I. Introduction

The goal of probabilistic model checking is to quantitatively evaluate the validity of performance and dependability properties of stochastic systems. After the system has been modeled as a Markov chain, or specified in terms of a higher-level language such as AADL [19], properties of interest are specified using the logic pCTL [11] or CSL [1]. Then, a model checker is invoked to determine in which states of the Markov chain these properties are satisfied.

Two main approaches to probabilistic model checking have emerged in recent years. In the first (numerical) approach one generates the state space of the Markov model beforehand and then numerically determines in which states the specified pCTL or CSL formula holds [1], [2]. In the second (statistical) approach, the behaviour of the system over time is repeatedly simulated in order to draw a conclusion about whether the property is satisfied in a certain state at a given level of confidence [21], [20].

Both of these approaches can experience problems when the probabilities of interest become small. For estimating probabilities using simulation it is a well-known rule of thumb that for a rare event probability p, $100/p$ simulation runs are needed to obtain a reasonable estimate [5]. In modern dependable (embedded) computer and communications systems, interesting probabilities of the order of magnitude of 10^{-8} are not uncommon, and methods to speed up the simulation process receive an increasing amount of attention.

Importance sampling is a sophisticated form of simulation that uses information about the system model to speed up the simulation process. If done correctly, this can lead to large increases in the efficiency. The price that we pay for such better estimates is the loss of generality. Any stochastic system can be simulated naively, but an importance sampling approach that works in one setting will typically fail to perform well in other settings. For example, a system consisting of components that are prone to failure can be highly dependable because the individual component failure rates are low or because the repair rates are high, yet the technique from [18] was proven to work well only in the former setting.

As a consequence, we need to restrict ourselves to certain types of models and rare events in this paper. The models considered here describe systems consisting of parallel component types as will be explained in detail in Section II. We are mainly interested in the case where the repair rates are high, as this is a common situation in practical model checking problems for which existing importance sampling approaches have no fully satisfactory answer, but we will also consider the case where the failure rates of components are low. We do not need to impose that the component failure and repair rates remain constant when one or several components have already failed.

The properties of interest are expressed using CSL, as we are interested in the continuous time behaviour of these systems. We will consider both the (transient) unreliability and the (steady-state) unavailability, also to be described in Section II. Admittedly, this is still considerably different from being able to evaluate whether an *arbitrary* CSL-formula holds. However, we view our current method as a first step towards more general stochastic model checking procedures.

The rest of this paper is structured as follows. In Section II we introduce the distributed database system that we will use as a case study, specify probabilities of interest and explain how to estimate those probabilities using simulation. In Section III we describe importance sampling in the general setting. In Section IV we introduce our approach and analyse its theoretical strengths and limitations. In Section V we evaluate our technique empirically and compare it to standard statistical model checking, to another, very general importance sampling scheme and to the numerical techniques of the model checking tool PRISM. Section VI concludes the paper.

II. Model & Preliminaries

As said in the introduction, importance sampling methods use information about the way rare events occur in the model to speed up the simulation. Because this information depends heavily on the model, we must first specify what type of models we will consider. In this section we will first describe our case study and use it to specify what kind of modes our method can handle. We will then specify what probabilities we need to estimate and how to do this using standard simulation, as we will need the ideas behind it for our discussion of importance sampling in Section III.

A. Distributed Database System

1) Model Description: The distributed database system is a benchmark problem in the field of dependability evaluation [17]. It was recently studied in [4], and a variant was studied in [8]. It can be seen as part of a more general class of systems consisting of parallel component types. We assume that the system as a whole is fault-tolerant, and that the probability of system failure is low either because of the component failure rates being low or because of the repair rates being high.

Specifically, the distributed database system consists of 24 disks that are grouped together in 6 clusters of 4 disks, 4 disk controller units divided into two sets that each access three disk clusters and a processor that accesses the disk controllers. The processor has a spare that takes over in case of failure. There is one repair facility for each of the six disk clusters, one for each of the two sets of disk controllers and one for the processor and its spare. The system is depicted in Figure 1.

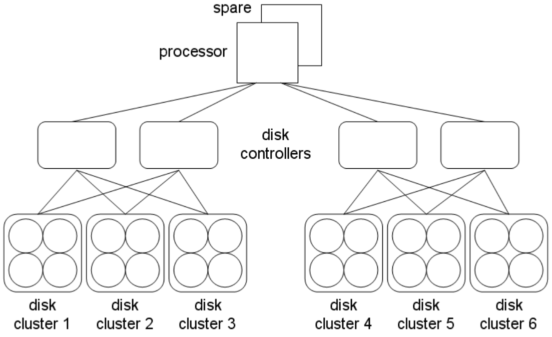

Figure 1. A distributed database system

We can distinguish 9 component *types*. Types $i = 1, \ldots, 6$ represent the disks in cluster i, types 7 and 8 represent the disk controllers in sets 1 and 2 respectively and type 9 represents the processors. The interfailure times and repair durations are assumed to be exponentially distributed. Let $\vec{x}$ be a vector in $\mathbb{N}^9$ in which each element x_i denotes how many components of type i have failed. We call this vector the *state* of the process. The system will be assumed to start in an initial state $\vec{x}_0$ at time $t_0 = 0$.

Let $\mathcal{D} = \{1, \ldots, 9\}$ be the set of component types. The failure and repair rates of components of these types may depend on the current state, so let the failure rates be some nonnegative function $\lambda_i(\vec{x})$ for each component type $i \in \mathcal{D}$ and $\vec{x} \in \mathbb{N}^9$. Let the repair rates similarly be given by nonnegative functions $\mu_i(\vec{x})$. The failures and repairs are called *transitions*. The repair rate of component type i can only be positive when there is at least one failed component of type i, and the failure rate can only be positive if there are components of type i left that are operational.

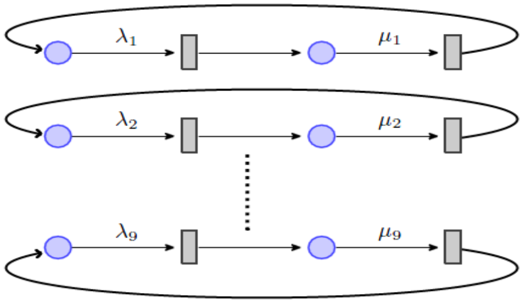

Figure 2. The distributed database system modeled as a stochastic Petri net.

We also note that the system can be modelled as a stochastic Petri net (SPN; see, e.g., [3]). In Figure 2 the system of the case study is depicted as an SPN. Each state is then a *marking*, the component types are then *places* and the number of *tokens* in place i then represents the number of failed components of type i.

2) Operation and Failure: The system is said to be operational if a processor can access all the data in the disks — this condition is satisfied if each of the following subconditions holds: (1) at least one processor is up, (2) at least one disk controller in each of the controller clusters is up, and (3) at least three disks are up in each of the six disk clusters. In general, the method that we introduce in this paper works when system failures occur if for at least one component type i, a specified number n_i of components has failed. System failure in the benchmark setting falls into this category.

Using this definition of system failure we can formalise what kind of measures we will estimate. The *unreliability* is the probability that the system stops being operational at some point before a specified time bound τ. The *unavailability* is the steady-state probability that at some time point t in steady-state the system is not operational. Unreliability properties can be expressed using CSL as $\mathcal{P}_{\bowtie p}(\Diamond^{<\tau}\mathit{fail})$ and steady-state unavailability properties as $\mathcal{S}_{\bowtie p}(\mathit{fail})$, where *fail* is an atomic property that is assigned to all states which represent system failure as defined earlier.

3) The Benchmark Case and Generalisations: The failure and repair rates of the individual components are given by:

unit	failure rate	repair rate
disks	λ	μ
disk controllers	3λ	μ
processors	3λ	μ

In the benchmark case (see [4]) we have $\lambda = 1/6000$ and $\mu = 1$. The rates in the literature are per hour, and the time bound τ for the unreliability is 5 weeks, so equal to 840 in this setting. The individual components all have the same failure distribution regardless of how many other components are up. E.g., the total failure rate of type 1 components (the first disk cluster) is 4λ when no components are down, 3λ when one component has failed, and so on. The component repair rate of each i is always μ if $x_i > 0$, because there is only repair facility per type.

One further generalisation is the number of spares. We introduce a new parameter n and assume there are n processors, n disk controllers per set and $2n$ disks per cluster. For $n = 2$ we are back in the benchmark case. Let failure in this more general setting be defined to occur when either (1) no processor is up, (2) in one disk controller set, no disk controller is up or (3) in one disk cluster at least n disks are down.

B. Discrete Event Simulation

Now that we have modeled the system and specified probabilities that we want to estimate, we will discuss *how* these probabilities can be estimated. The standard simulation-based approach is called *standard* discrete-event simulation or *Monte Carlo simulation*.

1) Path Generation: As mentioned in the introduction, we repeatedly simulate the behaviour of the system in order to come up with an estimate. The result of one simulation procedure is called a sample *run* or *path*. We define a *run* (timed path) to refer to a series of states and transition times that occur until we stop observing the system. By a (timeless) *path* we just refer to the series of states that we encountered. Let Ω be the set of all runs, then this is the sample space from which we randomly sample runs ω. We will not further delve into the measure theoretic background of Ω in this paper.

We generate samples from Ω as follows: we start the run at time $t_0 = 0$ in state x_0, which for the unreliability is given by the '*empty*' state $\vec{0}$. Then we consecutively determine which transition is taken and how long it takes until this transition is taken. This is done as follows: let, at step k, $\vec{x}_k$ be the state and let its *exit rate* $\eta(\vec{x}_k)$ be defined as

$$\eta(\vec{x}_k) = \sum_{j' \in \mathcal{D}} \lambda_{j'}(\vec{x}_k) + \mu_{j'}(\vec{x}_k). \tag{1}$$

We pick the transition j of type i as the next transition to be taken with probability

$$p_{\vec{x}_k}(j) = \begin{cases} \dfrac{\lambda_i(\vec{x}_k)}{\eta(\vec{x}_k)}, & \text{if } j \text{ is a failure transition,} \\[2ex] \dfrac{\mu_i(\vec{x}_k)}{\eta(\vec{x}_k)}, & \text{if } j \text{ is a repair transition.} \end{cases} \tag{2}$$

Also, let T_k be the time instance at which the k'th step is taken. Then we let the sojourn time $\Delta = T_{k+1} - T_k$ have probability density

$$f_{\vec{x}_k}(\delta) = \eta(\vec{x}_k)e^{-\eta(\vec{x}_k)\delta}. \tag{3}$$

We continue until we can terminate - a condition that depends on the property whose validity we seek to evaluate.

2) Estimating the unreliability: Let Φ^r be the event that the system hits a failure state before time τ, which we assume to be a model parameter given before the start of the simulation. Then the unreliability is given by $\pi = \mathbb{P}(\Phi^r) = \mathbb{E}(\mathbf{1}_{\Phi^r})$, where $\mathbf{1}_{\Phi^r}(\omega)$ denotes the indicator function which equals 1 if ω satisfies Φ^r and 0 otherwise. For each sample run we can evaluate whether Φ^r was satisfied on that run. So, after having sampled a series of runs $\{\omega_i, \ldots, \omega_N\}$ we can estimate $\mathbb{P}(\Phi^r)$ using

$$\hat{\pi} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\Phi^r}(\omega_i). \qquad (4)$$

Let $\hat{\sigma}$ be the sample standard deviation of our series of runs. The 95%-confidence interval for this estimate is then given by (see [14], page 254)

$$\left[\; \hat{\pi} - 1.96 \frac{\hat{\sigma}}{\sqrt{N}} \quad , \quad \hat{\pi} + 1.96 \frac{\hat{\sigma}}{\sqrt{N}} \; \right].$$

3) Estimating the unavailability: Estimating the steady-state unavailability using simulation is a little bit more tricky. To avoid having to '*warm-up*' the simulation before it reaches approximate equilibrium we apply a renewal argument. We partition the behaviour of the system as time progresses into disjoint *busy cycles*. A busy cycle starts and ends when we enter state $\vec{0}$. Let V be the steady-state unavailability, let Z be the amount of time during which the system is unavailable during a busy cycle and let D be the duration of a busy cycle. Then $\mathbb{E}(V) = \mathbb{E}(Z)/\mathbb{E}(D)$. The ratio estimator $\hat{v}$ is given by

$$\hat{v} = \frac{\hat{z}}{\hat{d}}, \qquad (5)$$

where $\hat{z}$ and $\hat{d}$ are the Monte Carlo estimates for $\mathbb{E}(Z)$ and $\mathbb{E}(D)$ respectively. This estimator is biased, but strongly consistent (i.e., $\hat{v} \to \mathbb{E}(V)$ as $N \to \infty$; see [14], page 533). We generate different runs for the estimates $\hat{z}$ and $\hat{d}$ to avoid dependence. This becomes even more necessary when we use importance sampling because techniques that focus on rare events would lead to a large variance of $\hat{d}$ (more details will be given in Section IV-D). The 95%-confidence interval (see [14], pages 532–533) is then given by

$$\left[\; \hat{v} - 1.96 \frac{\hat{\sigma}_v}{\hat{d}\sqrt{N}} \quad , \quad \hat{v} + 1.96 \frac{\hat{\sigma}_v}{\hat{d}\sqrt{N}} \; \right],$$

where $\hat{\sigma}_v^2 = \hat{\sigma}_z^2 + \hat{\sigma}_d^2 \hat{v}^2$ and $\hat{\sigma}_z^2$ and $\hat{\sigma}_d^2$ are the sample variances of sequences containing the V's and D's respectively.

Although the above estimation procedures work in many cases, the downside is that when the probability that we need to estimate is small the number of runs N that we need in (4) or (5) is enormous. Finding a solution to this problem will be the focus of the next two sections.

III. Principles of Importance Sampling

In this section, we will only describe importance sampling for estimating the probability of failure before time τ, but something similar can be done for the steady-state unavailability. The problem with small probabilities is that the fraction of runs in which a rare event happens is very small. When we apply importance sampling, we carry out a similar stepwise procedure as in the Section II-B1 but we use a different probability distribution in order to increase this fraction.

In this section, we will first describe importance sampling in Section III-B and then introduce the so-called zero-variance estimator on which we base our method in Section III-C. Before we start with the formal definitions of the aforementioned concepts, we first give an intuitive description.

A. Intuitive Description

Assume that we want to estimate some small probability w. Using standard simulation, we randomly draw zeros and ones such that the fraction of ones is expected to be w (see (4)). Suppose we now somehow make the probability of drawing a non-zero *twice as large*. Then, if we multiply the value $\mathbf{1}_{\Phi^r}(\omega_i)$ of the ith run in (4) by $\frac{1}{2}$, we obtain an estimator that is unbiased and which has a *lower variance* than the standard estimator. Now suppose we already know w and make drawing a non-zero exactly w^{-1} times as likely. Hence, we draw a non-zero with probability one and multiply each $\mathbf{1}_{\Phi^r}(\omega_i)$ by the precise probability that we wish to estimate, resulting in an estimator with *zero variance*.

Unfortunately, the systems we study are far too complex to '*just*' multiply the probability of drawing a non-zero by some number and multiply by a constant weighting factor. There are too many ways in which the event of interest can occur. We will need to tweak the individual transition probabilities and sojourn time densities, and in order to obtain an efficient new distribution we need to know enough about our system. The basic way to do this in complex stochastic systems will be discussed below.

B. Basic Setup of Importance Sampling

Recall that we consider systems consisting of parallel component types. Assume that we can at least divide the transitions into two classes: repair transitions and failure transitions. Also assume that failure transitions have low rates. One could ask how this makes the probability of system failure small. A first answer could be that the low component failure rates cause the failure transitions to rarely '*win the race*' against the repair transitions.

So, assume that at some step of the simulation process we are in a state where at least one repair transition is enabled. The idea is now to use a new probability distribution p^*, which we call the *simulation distribution* (also known as a *change of measure*), for the simulation such that the component failure probabilities are much higher than under the old distribution (2). We compensate for this overestimation by weighting the estimate with the ratio of p and p^* — like the factor $\frac{1}{2}$ of the example in III-A.

Every time a transition is sampled using the new density this weighting factor needs to be considered. The final weighting factor L of each run ω_i, called the *likelihood ratio*, is simply the *product* of the individual ratios $p_{\vec{x}_k}/p^*_{\vec{x}_k}$ in the run. Our new estimator then becomes

$$\hat{\pi} = \frac{1}{N} \sum_{i=1}^{N} L(\omega_i) \cdot \mathbf{1}_{\Phi^r}(\omega_i). \tag{6}$$

It is easy to prove that this estimator is unbiased for any new distribution that assigns positive probability to transitions that have positive probability under the old distribution.

We do not need to restrict ourselves to changing the *transition probabilities*. Note that the system failure probability can also be small because the time interval $[0, \tau]$ is too short for a sufficient number of component failures to occur. To remedy this, we can replace the *sojourn time density* f of (3) by a new density f^* with a higher transition rate. If we also account for the ratios f/f^* in the likelihood ratio L then our estimator remains unbiased and, if done correctly, has an even lower variance.

C. Zero Variance

Consider the following ideal situation: for every state $\vec{x}$ and time points $t \in [0, \tau]$ we already *know* the probability of system failure within $\tau - t$ time units. Call this probability $w(\vec{x}, t)$. Let $\chi(\vec{x}, j)$ be the new state that we obtain if transition j is chosen when we are in state $\vec{x}$, and $\mathcal{J}$ the set of all transition indices. Then we can introduce a new simultaneous density of the transition $j \in \mathcal{J}$ and sojourn time δ, i.e.,

$$p^*_{\vec{x}}(j, \delta) = \tag{7}$$
$$\frac{p_{\vec{x}}(j, \delta) \cdot w(\chi(\vec{x}, j), t - \delta)}{\int_0^T \sum_{j' \in \mathcal{J}} p_{\vec{x}}(j', \delta') \cdot w(\chi(\vec{x}, j'), t - \delta') d\delta'},$$

where $p_{\vec{x}}(j, \delta) = p_{\vec{x}}(j) \cdot f_{\vec{x}}(\delta)$. This new simulation density was proven to yield an estimator with zero-variance in [6]. Of course, we do not explicitly know the function w, or else we would not need to simulate. However, we might be able to come up with an approximation $\hat{w}$ for w. Then, we replace the function w in (7) by this approximation. If the simulation distribution associated with the approximation $\hat{w}$ is good enough then we have succeeded in overcoming the main problem facing standard Monte Carlo simulation of rare events.

IV. The New Simulation Distribution

The obvious next question is how to find a good way to find an approximation $\hat{w}$ that we can use to replace w in (7). In the following subsection, we will, as a first step, split the joint distribution of (7) into two distributions for the transitions and sojourn times respectively, and explain how to draw sojourn times in an efficient manner. In the remaining subsections we will find better approximations for w step-by-step.

A. Drawing Sojourn Times

If w were known explicitly we would use (7) by first drawing a transition and then selecting a sojourn time conditional on this transition. The latter step, drawing sojourn times δ from (7), can be computationally expensive. Typically, the distribution function of the sojourn time conditioned on a transition j is not invertible, which would force us to resort to accept-reject schemes (see [12], chapter 18).

To avoid this, we apply our first simplification: we use the old density function of δ conditional on transition j to occur before time $\tau - t$. This gives us the following density:

$$f^*_{\vec{x}}(\delta) = \frac{\eta(\vec{x}) e^{-\eta(\vec{x})\delta}}{1 - e^{-\eta(\vec{x})(\tau - t)}}.$$

The failure transitions are then drawn with probability

$$p^*_{\vec{x}}(j) = \frac{p_{\vec{x}}(j) \cdot \hat{w}(\chi(\vec{x}, j), t)}{\sum_{j' \in \mathcal{J}} p_{\vec{x}}(j') \cdot \hat{w}(\chi(\vec{x}, j'), t)} \tag{8}$$

for some approximation $\hat{w}$ yet to be determined (remember that $\mathcal{J}$ is the set of all transition indices). The technique of conditioning sojourn times on being smaller than $\tau - t$ is called *forcing* (see [15] or [16]). We do this for all transitions individually.

One could also draw a whole series of transitions and then condition on the sum of their sojourn times being smaller than $\tau - t$, but then we would have to deal with general sums of exponentially distributed random variables and that is something we want to avoid, as the evaluation of distribution functions of these sums can quickly become computationally expensive.

B. *Approximating w using Straight Paths*

A way of approximating rare event probabilities is to consider only the paths that lead to system failure of components of a certain type i (that is, $x_i \geq n_i$) *without cycles*. We define a cycle as a sequence of states in which the first and last state are the same. They necessarily consist of at least one failure and one repair transition.

Consider any state $\vec{x} \neq \vec{0}$ and a cycle starting in $\vec{x}$ of which the first transition is a component failure. For this cycle to occur, the failure transition must have occurred before a repair transition. When the component failure rates are made smaller or the repair rates are made higher, the occurrence of paths that contain these cycle becomes less likely. Accordingly, the straight paths — i.e., those without cycles — become a better approximation. This motivates the following, most basic, approximation for w.

Let a *straight path to failure* be a path that ends in a system failure state and which contains only failure transitions of a single component type. Let d be the number of types. From each state $\vec{x}$, we have d straight paths to failure, one for each type k. Let ν_k failures remain until the critical level n_k is reached, and let the vector of states that are seen along this path be denoted by $[\vec{x}_{k,0}, \ldots, \vec{x}_{k,\nu_k}]$, with $\vec{x}_{k,0}$ equal to $\vec{x}$. The probability of this path being taken equals

$$p([\vec{x}_{k,0}, \ldots, \vec{x}_{k,\nu_k}]) = \prod_{i=0}^{\nu_k - 1} \frac{\lambda_j(\vec{x}_{k,i})}{\eta(\vec{x}_{k,i})},$$

where $\eta(\vec{x})$ is defined as in (1). We can then use

$$\begin{aligned} \hat{w}^*(\chi(\vec{x}, j), t_i) = \quad & \hat{w}^*(\chi(\vec{x}, j)) \\ = \quad & \sum_{k=1}^{d} \prod_{i=0}^{\nu_k - 1} \frac{\lambda_j(\vec{x}_{k,i})}{\eta(\vec{x}_{k,i})} \end{aligned} \tag{9}$$

as a time-independent approximation of w. From now on, the $*$ in $\hat{w}^*$ indicates that we only use the straight paths as an approximation.

C. *Probability Contribution of Paths with Cycles*

Unfortunately, the approximation (9) is too crude. One shortcoming of $\hat{w}^*$ is that the most likely path from a state $\vec{x}'$ to system failure might not be one of the d straight paths. In many cases, the most likely path is the path in which the system first returns to state $\vec{0}$ and then takes one of the straight paths that determine $\hat{w}^*(\vec{0})$. This can be seen in Figure 3, which depicts the state space of a simplified model with only two component types. Starting from state $\vec{x}'$ in Figure 3, the dashed path is (when the rates are realistic) much less likely to occur than the solid line path because the failure transitions of the dashed path need to win the race against the repair transition of type 2.

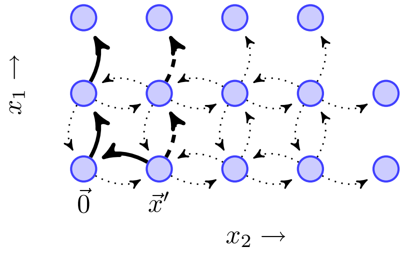

Figure 3. A model consisting of two types of components. One type has a many more spares than the other, but its components fail more quickly.

Accordingly, we also consider the straight paths from state $\vec{0}$ for our approximation $\hat{w}(\vec{x}')$. From state $\vec{x}'$, the system returns to state $\vec{0}$ with probability almost equal to one. Therefore, for each state $\vec{x}$ we can use the sum of $\hat{w}^*(\vec{x})$ and $\hat{w}^*(\vec{0})$ instead of just the former. As a consequence, the jump from state $\vec{0}$ to state $\vec{x}'$ will more often be taken under the new distribution. This is desirable — paths that contain cycles between state $\vec{0}$ and the states where one component has failed are almost equally likely as the straight paths from $\vec{0}$.

However, the contribution of $\hat{w}^*(\vec{0})$ needs to be time-dependent — cycles to the empty state are *only* likely when the empty state's exit rate $\eta(\vec{0})$ is high enough compared to the remaining time $\tau - t$. Otherwise, the extra jumps take too much time, which reduces the likelihood of these paths.

A crucial insight is that the time-independent function $\hat{w}^*(\vec{0})$ is still a good approximation for the probability of hitting a system failure state *during a busy cycle*. When the failure rates are low or the repair rates are high, the duration of the busy cycle is almost completely determined by the time spent in state $\vec{0}$. Therefore, the time it takes before we reach a system failure state is the sum of M busy cycle durations D_i. Here, the durations D_i are all independent and exponentially distributed approximately with the rates $\eta(\vec{0})$, whereas the number M follows a geometric distribution with approximate success parameter $\hat{w}^*(\vec{0})$.

From elementary probability theory we know that this sum follows an exponential distribution with rate $\eta(\vec{0}) \cdot \hat{w}^*(\vec{0})$, hence the probability that this is completed before $\tau - t$ time units has approximate probability $1 - \exp(-\eta(\vec{0}) \cdot \hat{w}^*(\vec{0})(\tau - t))$.

For small x, $1 - e^{-x}$ approximately equals x. Therefore, we do not use $\hat{w}^*(\vec{0})$ as our approximation for $w(\vec{0})$ but the time-dependent function $\eta(\vec{0}) \cdot \hat{w}^*(\vec{0})(\tau - t)$. This motivates our final approximation,

$$\hat{w}(\vec{x}, t) \;=\; \tag{10}$$
$$\begin{cases} \hat{w}^*(\vec{0}) \cdot (\tau - t) \cdot \eta(\vec{0}), & \text{if } \vec{x} = \vec{0}, \\ \hat{w}^*(\vec{x}) + \hat{w}^*(\vec{0}) \cdot (\tau - t) \cdot \eta(\vec{0}), & \text{otherwise .} \end{cases}$$

This new distribution (10) keeps the estimator efficient when the rarity of the event of interest is not caused by the low component failure rates but rather the high recovery rates. Our numerical results will show that this adaptation is crucial in practical situations.

D. Steady State Unavailability

So far, we have described how to estimate the unreliability Φ^r, but a similar approach can be used for the unavailability V. Consider the ratio estimator (5). The problem that we face is that for the vast majority of runs the time fraction $Z(\omega)$ will equal zero, regardless of whether the failure rates were low or the repair rates were high. So, we need to increase the probability of hitting a system failure during a busy cycle.

The procedure will be as follows: we start in the empty state and simulate using (10) substituted into (8) — since this is a steady-state performance measure we set $\tau - t \equiv 0$ in (10), thus effectively disabling returns to state $\vec{0}$. We stop when we reach system failure and from then on simulate using the old distribution until we reach state $\vec{0}$ [13]. Meanwhile, we record the amount of time during which the system was in a failed state.

If we would apply the same distribution as we used for (Z), the paths that immediately fall back to the empty state before reaching a system failure state are never sampled because $\hat{w}(\vec{0}, t) \equiv 0$. This has no effect on the unbiasedness of the estimator $\hat{z}$ because paths that immediately fall back to $\vec{0}$ contribute nothing to $\mathbb{E}(Z)$. However, they do contribute heavily to $\mathbb{E}(D)$. Therefore, to avoid bias and inconsistency in $\hat{d}$ we generate two series of runs, one for Z with importance sampling and one for D without importance sampling [10]. After we have computed the estimates, we substitute them into (5) and use the same confidence interval as the one described in Section II-B.

V. Results

In this section, we demonstrate that our method produces good results in practice. We compare our method to a few other well-known techniques. The first of these is the standard Monte Carlo method. A more efficient method is that of balanced failure biasing (BFB) combined with forcing (see Section IV-A or [15]). Under BFB, the total probability of a component failure is set to $\frac{1}{2}$, uniformly distributed over the individual component types (and similarly for the repairs — for more information, see [18]). The third simulation method found in the result tables of this section are the estimates produced using our new method, abbreviated as Path-IS. Finally, we will compare our method to the numerical methods of the model checking tool PRISM.

When we display the experimental results in a table, we first give the statistical estimates. These are either the standard Monte Carlo estimates as in (4) or importance sampling estimates as in (6), which will be clear from the context, and are given in the form of a 95%-confidence interval. To the right of the estimates, we state the number of simulation runs used to produce these estimates. The number of simulation runs for each method was picked such that the computation time was comparable to that of PRISM. In the last row(s), we display the numerical solutions and the number of states in the PRISM and Arcade models (the latter tool uses lumping/bisimulation minimisation to reduce the size of the state space). The exact computation and simulation times are specified in the text of the following subsections.

A. Experimental Setup

Next, we will describe how we will test the strength of our method. Of course, there is a fundamental difference between the numerical approach and the statistical approach in the sense that numerical methods (if they converge) give an almost perfect (depending on the stopping criterion) approximation after some fixed time interval. On the other hand, statistical methods produce confidence intervals that can be made as narrow as one would like, depending on how much time one is willing to spend. The best way to say something about the applicability of an approach for the user is to look at the wall-clock time.

We used a computer with a 2.8 GHz Intel® Core$^{\text{TM}}$ 2 Duo processor (32-bit) and 3 GB of RAM, running Windows XP. All simulations were run with a simple Java program that generated (pseudo-)random numbers using a fast Mersenne twister [1]. We used version 3.3.1 of PRISM.

B. Unavailability

Of the two measures discussed in this paper, the unavailability is the easiest to approximate. Because it considers the system when it is in equilibrium, no information about the transient behaviour of the system is needed. Numerical methods to analytically determine or iteratively approximate it are well-established.

First, we will show in Table I that our results are consistent with the other tools and the literature, namely [4]. The unavailability in [4] was only given in one significant digit, and the total run time was not specified. When we lower the component failure rate parameter λ from $1/6000$ to $\frac{1}{6} \cdot 10^{-6}$, we get similar results, with the exception of standard Monte Carlo. This is displayed in Table II. Increasing μ from 1 to $1,000$ gives us equivalent results, as depicted in Table III (note that the unavailability values for $\lambda = \frac{1}{6} \cdot 10^{-6}$ and $\mu = 1,000$ are exactly the same. This is not a coincidence, as the solution depends only on the transition rates through the ratio $\frac{\lambda}{\mu}$). In all these cases PRISM does better than the simulation approaches discussed so far — indeed, for models with small state spaces PRISM's steady-state techniques can be preferred to simulation, regardless of λ. However, if we increase the number of spare components n to 3, the size of the state space blows up from $421\ 875$ states to $7\ 529\ 536$, as can be seen in Table IV. This causes PRISM's computation time to increase, from about 3.4 seconds for Tables I-III to 113.1 seconds for Table V. When we increase n even further, we hit tougher boundaries on the applicability of numerical methods due to the state space explosion problem. For $n \geq 4$, the amount of memory that our system has available for "*creating* [a] *vector for diagonals*" is insufficient and PRISM terminates without giving a solution (even after adjusting the memory usage maxima in PRISM's settings). For $n = 6$, Path-IS still produces accurate estimates when we set the simulation time to a mere 60 seconds, as can be seen in Table VI. BFB underestimates the unavailability, a well-known phenomenon when the change of measure being used is not suitable for the problem [7].

	$\hat{v}$ (10^{-6})	# runs
MC	3.677 ± 0.778	388 196
BFB	3.647 ± 0.104	169 484
Path-IS	$\mathbf{3.511 \pm 0.035}$	**79 611**
	v (10^{-6})	# states
PRISM	3.498	421 875
Arcade	3	2 100

Table I
Unavailability ($\hat{v}$) results for the benchmark case. $\lambda = 1/6000, \mu = 1, n = 2$.

	$\hat{v}$ (10^{-12})	# runs
MC	5.847 ± 11.460	386 538
BFB	3.532 ± 0.105	165 943
Path-IS	$\mathbf{3.521 \pm 0.036}$	**78 179**
	v (10^{-12})	# states
PRISM	3.500	421 875

Table II
Unavailability ($\hat{v}$) results when $\lambda = \frac{1}{6} \cdot 10^{-6}; \mu = 1, n = 2$.

	$\hat{v}$ (10^{-12})	# runs
MC	0 ± 0	384 418
BFB	3.504 ± 0.102	165 115
Path-IS	$\mathbf{3.465 \pm 0.035}$	**76 923**
	v (10^{-12})	# states
PRISM	3.500	421 875

Table III
Unavailability ($\hat{v}$) results when $\mu = 1,000$; $\lambda = 1/6000, n = 2$.

n	# states	# non-zeros
2	421 875	5 737 500
3	7 529 536	111 329 568
4	66 430 125	1 027 452 600
5	382 657 176	6 087 727 800
6	1 655 595 487	26 853 394 932

Table IV
State space sizes and numbers of non-zero entries in the transition rate matrix of the models built by PRISM for different values of n.

	$\hat{v}$ (10^{-9})	# runs
MC	7.235 ± 6.135	12 977 468
BFB	5.656 ± 0.151	3 434 986
Path-IS	$\mathbf{5.580 \pm 0.015}$	**1 315 050**
	v (10^{-9})	# states
PRISM	5.578	7 529 536

Table V
Unavailability ($\hat{v}$) results when $n = 3$; $\mu = 1, \lambda = 1/6000$.

	$\hat{v}$ (10^{-16})	# runs
MC	0 ± 0	6 708 624
BFB	0.148 ± 0.225	803 752
Path-IS	$\mathbf{1.173 \pm 0.016}$	**205 654**
	v (10^{-16})	# states
PRISM	N.A.	1 655 595 487

Table VI
Unavailability ($\hat{v}$) results when $n = 6$; $\mu = 1, \lambda = 1/6000$.

C. Unreliability

The unreliability is (from a theoretical point of view) a more interesting case than the unavailability because, unlike the latter value, the former value is not known in closed form for the models that

[1] http://www.cs.gmu.edu/∼sean/research/

we consider [9] — hence, we simply have to use numerical and/or statistical methods. First, note that we have defined the unreliability to refer to the probability of system failure before some time point τ (in this case 840 hours), allowing the repair of components in this time interval. In [4] and [17], component repairs were *not* allowed to occur.

Because PRISM's numerical evaluation was very quick (0.235 seconds), we gave the statistical methods more time (60 seconds). After all, the purpose of Table VII is only to show that our results are consistent with the literature even when the repair transitions are disabled. Again, no run time was given for Arcade in [4]. Note that standard Monte Carlo and BFB give the best results in this setting because their simplicity allows them to sample many more runs within the (real) time constraint. When we allow repairs to occur the unreliability drops to approximately 0.0029. It takes PRISM little more than 30 seconds to compute this probability. This computation time does not depend on λ, as it took a comparable amount of time to generate the results of Table VIII, where we lowered λ to $\frac{1}{6} \cdot 10^{-6}$.

	$\hat{\pi}$	# runs
MC	0.5981 ± 0.0003	8 304 940
BFB	0.5976 ± 0.0003	5 116 887
Path-IS	$\mathbf{0.5977 \pm 0.0019}$	**93 526**
	π	# states
PRISM	0.5980	421 875
Arcade	0.5980	2 100

Table VII
Unreliability ($\hat{\pi}$) results without repair when $\mu = 0$; $n = 2$, $\lambda = 1/6000$.

However, when we increase μ, the time that PRISM needs to produce a solution increases along with it. The applied numerical methods require that the transition rate matrix be uniformised, and the uniformisation rate increases linearly in μ. PRISM's computation time in turn increases linearly in the product of the uniformisation rate and the mission time (see [12], chapter 15). Because the uniformisation rate is so much higher than the original exit rate of the empty state, many unnecessary self-loops are taken into account. This can heavily slow down the computation. On the other hand, the accuracy of the Path-IS estimate remains constant as μ increases since the jumps out of the empty state still occur with the same low rate. A few estimates together with PRISM computation times are given in Figure 4. Notice that when $\mu = 100$, PRISM takes over half an hour to produce an approximation, while our simulation method can produce a decent estimate in 10 seconds.

	$\hat{\pi}$ (10^{-9})	# runs
MC	0 ± 0	18 438 588
BFB	2.936 ± 0.024	1 042 866
Path-IS	$\mathbf{2.937 \pm 0.001}$	**992 231**
	π (10^{-9})	# states
PRISM	2.936	421 875

Table VIII
Unreliability ($\hat{\pi}$) results when $\mu = 1$; $n = 2$, $\lambda = \frac{1}{6} \cdot 10^{-6}$.

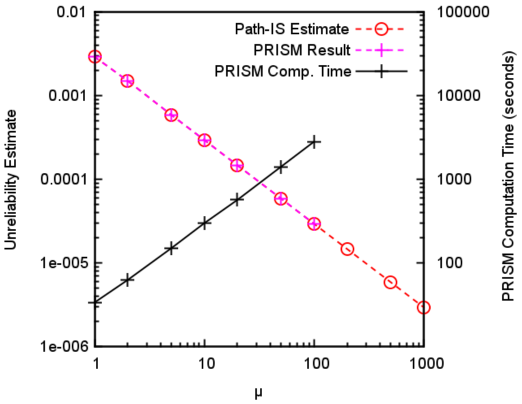

Figure 4. Estimates for the probability π (the unreliability) from PRISM (dashed, crosses) and Path-IS (dashed, circles) on left vertical axis; PRISM run time (solid, crosses) on right vertical axis. The Path-IS run time was only 10 seconds, but the bounds of the 95%-confidence interval were still not distinguishable from the estimate at this scale.

For high μ (and high τ), the confidence intervals of BFB are also noticeably wider than those of Path-IS. For $\mu = 1000$ (10 second run time), they were 2.943 ± 0.013 and 2.920 ± 0.358 (times 10^{-6}) respectively. Again, discussing why goes beyond the scope of this paper.

For higher n PRISM again starts to suffer from the state-space explosion problem. We omit results for this scenario as they are comparable to the results for the unavailability when n is high.

VI. Conclusions and Discussion

A. General Conclusion

In this paper we have introduced an efficient simulation technique that is able to estimate dependability measures in situations were system failure is a rare event due to high repair rates or low component failure rates. The approach that we used, based on (1) the zero-variance measure for transient failure probabilities in CTMCs, and (2) likely paths to failure, is something that we hope to generalise to other situations in the future.

We have demonstrated that our technique performs well even for large models as long as the component failure rates are much lower than the repair rates. Also, we have shown that our method performs well in comparison to other methods. Numerical techniques, as, e.g., implemented in PRISM, suffer from large state spaces and high uniformisation rates.

B. State Space Explosion vs. Extra Computations

The beauty of discrete-event simulation is that the state space does not need to be generated a priori. However, if no information about the states is stored then the values $\hat{w}^*(\vec{x})$ need to be computed again every time we encounter the same $\vec{x}$ throughout the simulation process. For the simulations, we used a caching approach in order to balance between these two extremes: we did not generate the state space beforehand, but each time a value $\hat{w}^*(\vec{x})$ was computed for a new $\vec{x}$ we stored it in a dynamic array in order to keep record only of the most interesting part of the state space.

However, when the list grows larger throughout the simulation process each step takes more time. A different approach could be to only store the values $\hat{w}^*$ for the states that are on or next to the straight paths, and recalculate all the other values at each step. An interesting topic of further research would be to investigate how these approaches influence the time needed to run simulations.

C. General System Failure Conditions

In this paper, we have only considered system failures caused by the number of failures of one component type i reaching a critical level n_i. More general failure conditions, e.g., system failure occurring when at some time point $t < \tau$ certain numbers $n_{k_1}, \ldots, n_{k_c}$ have failed for c component types $k_1 \ldots, k_c$, should not form a major obstacle. More paths to failure may need to be considered for a good approximation $\hat{w}$ — perhaps even a number of paths that increases exponentially in c — but many of them typically have equal probability which makes accounting for them easier.

Acknowledgements

This work is supported by the Netherlands Organisation for Scientific Research (NWO), project number 612.064.812.

References

[1] A. Aziz, K. Sanwal, V. Singhal, and R.K. Brayton. Verifying continuous-time Markov chains. *Lecture Notes in Computer Science*, 1102:269–276, 1996.

[2] C. Baier, B. Haverkort, H. Hermanns, and J.P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on software engineering*, 29(6):524–541, 2003.

[3] F. Bause and P.S. Kritzinger. *Stochastic Petri Nets*. Vieweg, 2002.

[4] H. Boudali, P. Crouzen, B.R. Haverkort, M. Kuntz, and M. Stoelinga. Arcade-A formal, extensible, model-based dependability evaluation framework. In *13th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS 08), Belfast*, volume 3, pages 243–248. Citeseer, 2008.

[5] J.A. Bucklew and R. Radeke. On the Monte Carlo Simulation of Digital Communication Systems in Gaussian Noise. *IEEE Trans. on Communications*, 51(2), 2003.

[6] P.T. de Boer, P. L'Ecuyer, G. Rubino, and B. Tuffin. Estimating the probability of a rare event over a finite time horizon. In *Proceedings of the 2007 Winter Simulation Conference*, pages 403–411, 2007.

[7] M. Devetsikiotis and J.K. Townsend. An algorithmic approach to the optimization of importance sampling parameters in digital communication system simulation. *IEEE Transactions on Communications*, 41(10):1464–1473, 1993.

[8] A. Goyal, W.C. Carter, E. de Souza e Silva, S.S. Lavenberg, and K.S. Trivedi. The system availability estimator. In *Proceedings of the 16th Int. Symp. on Fault-Tolerant Computing*, pages 84–89, 1986.

[9] A. Goyal, S.S. Lavenberg, and K.S. Trivedi. Probabilistic modeling of computer system availability. *Annals of Operations Research*, 8(1):285–306, 1987.

[10] A. Goyal, P. Shahabuddin, P. Heidelberger, V.F. Nicola, and P.W. Glynn. A unified framework for simulating Markovian models of highly dependable systems. *IEEE Transactions on Computers*, 41(1):36–51, 1992.

[11] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal aspects of computing*, 6(5):512–535, 1994.

[12] B.R. Haverkort. *Performance of computer communication systems: a model-based approach*. John Wiley & Sons, Inc. New York, NY, USA, 1998.

[13] P. Heidelberger. Fast simulation of rare events in queueing and reliability models. *Performance Evaluation of Computer and Communication Systems*, pages 165–202, 1993.

[14] A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill New York, 1991.

[15] M.K. Nakayama and P. Shahabuddin. Quick simulation methods for estimating the unreliability of regenerative models of large, highly reliable systems. *Probability in the Engineering and Informational Sciences*, 18(03):339–368, 2004.

[16] V.F. Nicola, M.K. Nakayama, P. Heidelberger, and A. Goyal. Fast simulation of highly dependable systems with general failure and repair processes. *IEEE Transactions on Computers*, 42(12):1440–1452, 1993.

[17] W.H. Sanders and L.M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of parallel and distributed computing*, 15(3):238–254, 1992.

[18] P. Shahabuddin. Importance sampling for the simulation of highly reliable Markovian systems. *Management Science*, pages 333–352, 1994.

[19] SAE AADL working group. *Architecture Analysis and Design Language (AADL)*. SAE standards AS5506, Nov 2004, http://www.sae.org/technical/standards/AS5506.

[20] H.L.S. Younes, M. Kwiatkowska, G. Norman, and D. Parker. Numerical vs. statistical probabilistic model checking. *International Journal on Software Tools for Technology Transfer (STTT)*, 8(3):216–228, 2006.

[21] H.L.S. Younes and R.G. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In *Proceedings of the 14th International Conference on Computer Aided Verification,*, pages 223–235, 2002.