
CSL
Coordinated Science Laboratory
The University of Illinois at Urbana-Champaign - College of Engineering

Technical Report

Dependency-Based Decomposition of Systems Involving Rare Events

Eric W. D. Rozier and William H. Sanders
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 West Main Street, Urbana, Illinois

{erozier2,whs}@illinois.edu

Keywords— dependence, decomposition, dependability, rare event, modeling

Abstract

Analysis of the dependability of large-scale systems presents challenges due to both the state space explosion problem and the increasing potential impact of rare events on dependability metrics. We propose a novel decomposition method that utilizes information on rare events of interest and system dependencies. We decompose models by building a graph that represents specified reward variables and the dependence relations implied in the model specification. Near-independent relationships that involve rare events and their consequences are then identified and used to decompose the model. The resulting submodels can then be assumed to be independent until the first rare event fires, at which point the resulting model state can be reanalyzed, modifying the decomposition to maintain the validity of the assumed independence. A simplified model based on a real data deduplication system is presented as evidence of one application of our approach.

1 Introduction

Large and complex systems present numerous problems for modelers. As systems become larger, they encounter the state-space explosion problem, presenting challenges for numerical solution. Though simulation can be used to estimate reward variables for even infinite state spaces, solution time grows with the number of events that must be processed. In cases where events in the model have rates that differ by many orders of magnitude, so-called *rare events*, the model presented becomes stiff, increasing the number of events that must be observed for the estimates of the chosen metrics to converge.

We examine these problems in the context of dependable systems that are composed of sets of nearly independent subsystems. This encompasses a rich domain with many practical applications. Scientific computing has historically driven work on large-scale computing resources, improving supercomputer performance by two orders of magnitude each decade, as well as increasing the performance gap between the processing ability of systems and that of individual nodes within a system [1]. The frequency of failure of these components and the propagation of these failures to other resources are increasingly important for understanding system dependability [2]. Large-scale storage systems have also witnessed similar trends, with rises in the capacity of storage systems outstripping trends in individual disk capacity. Combined with recent trends in rare silent disk failures [3] and the increased risk of error propagation posed by data deduplication [4, 5], there is a need to develop new ways to handle the complexity of these systems and the rare events that impact their dependability.

One of the most critical problems facing the modeling community is the "state-space explosion problem." Complex systems can feature large, or even infinite, state spaces, making it impossible to explore the space in its entirety. While there are many approaches to mitigating the computational challenges posed by large state spaces, two of the most common methods include largeness tolerance and largeness avoidance. Largeness tolerance utilizes algorithms and data structures to maximize both the number of states that can be represented within a given set of resources, and the speed at which these states can be accessed and modified. Examples include lumped matrix diagrams [6], specialized search trees [7], avoidance of explicit state-space representation [8], Kronecker products [9], and the algorithms presented by Kemper [10] and Buchholz [11].

State-space reduction methods typically focus on exploiting model or system characteristics that reduce the number of states that must be considered to compute a solution. Examples include methods for partial exploration of a state space [12], symmetry detection [13], and other methods that take advantage of structural properties, such as hierarchical modeling [14].

Rare events, despite the low probability of their occurrence, can have a large impact on the systems in which they occur. For safety-critical systems they may define unsafe situations that can cause a critical breakdown that results in loss of life. Outside of life-critical systems, they can represent faults that can result in catastrophic data loss [3] or system failure. As systems increase in size and complexity, rare event failures pose an increased risk when they occur on a per-component basis. While such failures may still occur with the same proportion to other events, large-scale systems, such as petascale computing resources, can often be expected to suffer a number of rare events within their normal lifetimes [15].

Two primary strategies exist for increasing the tractability of models that contain rare events: importance sampling and importance splitting. Importance sampling attempts to reduce the variance of estimates of a model's reward variables through mathematical biasing of the simulation, increasing the proportion of rare events witnessed. This is accomplished by biasing the distributions, yielding a biased estimator. The modeler must then come up with an appropriate way to unbias the estimators to correct for the biased distributions [16, 17]. While importance sampling can greatly speed up simulations of models that have rare events, choosing an appropriate set of biased distributions, and unbiasing functions for the estimators, is a bit of an art, and can prove difficult. Improper choices may have the effect of slowing down the simulation, or, worse, yielding incorrect estimators for the reward variables.

Importance splitting also attempts to bias the simulation to make rare events less rare, but does so

using a different approach. With importance splitting, the state space is partitioned into a number of subsets (or *levels*). A level contains those states that form critical points on "important paths" in the simulation, i.e., those that result in an increased probability of witnessing a rare event. Simulation paths that reach a level are split and re-sampled to increase the likelihood of witnessing a rare event. All trajectories resulting from a split are correlated, and generation of an unbiased estimator of the variance is not a straightforward process. Selection of appropriate points to split and levels is a model-specific problem that can impact the efficiency of solutions using this technique [18, 19].

Decomposition techniques also offer an approach to solving large, complex systems. They do so by dividing them into smaller submodels and finding solutions for the submodels separately. If the model cannot be broken into wholly independent submodels, the submodel interactions must also be characterized. If the submodels are weakly coupled, we may be able to consider the system as a composition of nearly independent submodels [20, 21].

In this paper we present a novel method for automatic decomposition of models that contain rare events. Our technique relies on models whose structures consist of a number of nearly independent submodels, made dependent by one or more rare events within the model. In Section 2 we present a model specification language, which is intended to generalize our techniques, since they do not rely on a specific formalism. In Section 3 we discuss a method for characterizing the dependence relationships in a model as a first step towards decomposition. We then present in Section 4 a definition of rare events and methods for identifying rare events in a model. In Section 5 we present an algorithm that uses our graph of model dependencies and identified rare events to decompose models. We discuss solution methods for the results of our decomposition in Section 6; in Section 7 we present an example of our methods and discuss the results. Finally, we conclude with Section 8.

2 Model Description

We present our method in the context of a generic model specification language based on the notation presented in [22]. This is intended as an alternative to presenting our results in a specific formalism, both to simplify the discussion of our techniques and to generalize our methods. While many different formalisms exist for describing discrete event systems, most can be mapped into our provided notation.

Definition 1. *A high-level model specification is a 5-tuple* $(S, E, \Phi, \Lambda, \Delta)$.

- *S is a finite set of state variables $\{s_1, s_2, \ldots, s_n\}$ that take values in $\mathbb{N}$.*
- *E is a finite set of events $\{e_1, e_2, \ldots, e_m\}$ that may occur in the model.*
- *$\Phi : E \times 2^{S \times \mathbb{N}} \rightarrow \{0, 1\}$ is the event-enabling function specification. For each event $e \in E$, and any set of state variables and their assignments q, event e is enabled and may occur for this set of state variable assignments iff $\Phi(e, q) = 1$.*
- *$\Lambda : E \times 2^{S \times \mathbb{N}} \rightarrow (0, \infty)$ is the transition rate function specification. For each event e, and set of state variables and their assignments q, event e occurs with rate $\Lambda(e, q)$ when the state variables of the model have the values given in q.*
- *$\Delta : E \times 2^{S \times \mathbb{N}} \rightarrow 2^{S \times \mathbb{N}}$ is the state variable transition function specification. For each event $e \in E$, and each set of state variables and their assignments $q \in 2^{S \times \mathbb{N}}$, $\Delta(e, q) \rightarrow q'$ defines the values assigned to all state variables of the model when e occurs.*

Definition 2. *The state of a model M is obtained from the mapping $\psi : S \rightarrow \mathbb{N}$, where for all $s \in S, \psi(s)$ is the value of the state variable s. $\Psi = \{\psi | \psi : S \rightarrow \mathbb{N}\}$ is the set of all such mappings.*

Given definition 2 we define a set of functions ϕ, λ, and δ, analogous to Φ, Λ, and Δ, which form the part of the model underlying the specification from definition 3.

Definition 3. *A model, M, is a 5-tuple $(S, E, \phi, \lambda, \delta)$ in which*

- *S is a set of state variables $\{s_1, s_2, \ldots, s_n\}$ that take values in $\mathbb{N}$.*
- *E is the set of events $\{e_1, e_2, \ldots, e_m\}$ that may occur in the model.*
- *$\phi : E \times \Psi \rightarrow \{0, 1\}$ is the event-enabling function. For each $e \in E$ and $\psi \in \Psi$, $\phi(e, \psi) = 1$ if event e_i may occur when the state of the model is ψ_i, and $\phi(e_i, \psi_i) = 0$ if the event e_i may not occur when the model is in state ψ_i.*
- *$\lambda : E \times \Psi \rightarrow (0, \infty)$ is the transition rate function. For each event $e \in E$, and each state ψ such that $\phi(e, \psi) = 1$, the event e occurs with rate $\lambda(e, \psi)$ while in state ψ.*
- *$\delta : E \times \Psi \rightarrow \Psi$ is the state transition function. For each event $e \in E$, and each state $\psi \in \Psi$, the transition function can be used to compute the new state resulting from the occurrence of e while the model is in ψ as $\delta(e, \psi) \rightarrow \psi'$.*

A trajectory, or behavior of a model, is described as a finite sequence of states and events. The model is assumed to be in some initial state, with events occurring with a rate determined by λ. When an event fires, the model transitions in accordance with the state transition function δ. The probability of transitioning from some arbitrary state, ψ_i, to a particular next state, ψ_j, is the probability that an event e is the next event such that $\delta(\psi_i, e) = \psi_j$. We calculate this probability as:

$$P(\psi_i \rightarrow \psi_j) = \frac{\sum_{e \in E | \delta(e,\psi_i)=m_j} \lambda(e, \psi_i)}{\sum_{e \in E | \phi(e,\psi_i)=1} \lambda(e, \psi_i)}. \quad (1)$$

In addition to specifying a model of a system, one must specify the performability, availability, or dependability measures for a model. For the formalism given in definition 1, these measures are specified in terms of reward variables [23]. Reward variables are specified as a reward structure [24] and a variable type.

Definition 4. *Given a high-level model specification $M = (S, E, \Phi, \Lambda, \Delta)$, we define two reward structures: rate rewards and impulse rewards.*

- *A rate reward is defined as a function $\mathcal{R} : 2^{S \times \mathbb{N}} \rightarrow \mathbb{R}$, where for $q \in 2^{S \times \mathbb{N}}$, $\mathcal{R}(q)$ is the reward accumulated when for all $(s, n) \in q$ the marking of s is n.*
- *An impulse reward is defined as a function $\mathcal{I} : E \rightarrow \mathbb{R}$, where for $e \in E, \mathcal{I}(e)$ is the reward earned upon completion of e.*

Definition 5. *Let $\Theta_M = \{\theta_0, \theta_1, \ldots\}$ be a set of reward variables, each with reward structure $\mathcal{R}$ or $\mathcal{I}$ associated with a model M.*

The type of a reward variable determines how the reward structure is evaluated, and can be defined over an interval of time, an instant of time, or in steady state, as shown in [25, 23].

3 Analyzing Model Dependence

In order to decompose a given model M, we first analyze the dependence relationships present in the specification. Our goal is to identify and exploit structural properties as they relate to rare events that form dependence relationships with otherwise independent submodels.

In the ideal case, we would be able to identify fully independent submodels, which we could trivially decompose. Most systems, however, are more complex and feature some level of dependence between

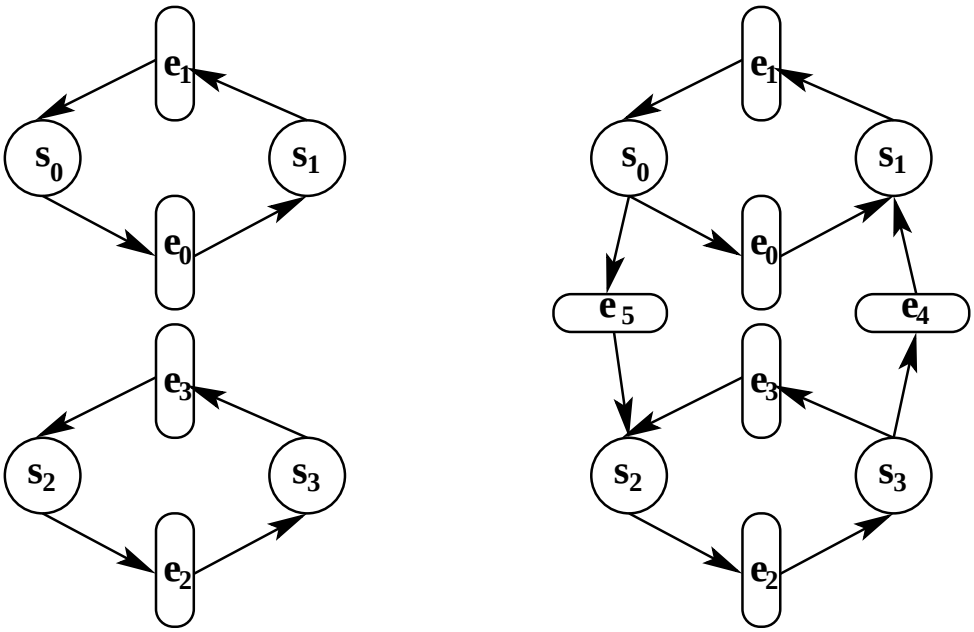


Figure 1: Two examples of near-independent submodels. The initial values of all state variables in both models are given by $s_0 = s_2 = 1$, and $s_1 = s_3 = 0$.

potential submodels. In these cases, it is sometimes possible to utilize the idea of *near-independence* [26]. Characterizing the dependence of two portions of a model is complex, and involves developing a measure of how far the model is from an ideal set of truly independent models. For that reason, using the terminology discussed in [26], we will simply present a qualitative discussion of structures that can arise in models that feature near-independence.

3.1 Nearly Decomposable Models

When rare events form the point of common connection between two near-independent submodels, we wish to exploit the model structure to decompose it into a set of smaller, more tractable submodels, $\Xi = \{\xi_0, \xi_1, \ldots\}$. Given that the rate of firing of rare events is much lower than that of other events in our model, these near-independent relationships can be assumed during much of a trajectory of a model to be equivalent to independence. In between subsequent rare events, submodels cannot interact, potentially simplifying the solution process.

We present a Petri net, in Figure 1a, to aid in our discussion. A Petri net representation is used simply because it may be familiar to the reader and because the simple nature of the example model

permits it. We do not mean to imply that all models that can be specified by definition 1 can be written in this form. In Figure 1a we have a model with state variables s_0, s_1, s_2, and s_3 and events e_0, e_1, e_2, and e_3. An arc connecting a state variable to an event indicates that the event is enabled only when the indicated state variable has a value greater than zero, and that upon firing of the event, the state variable is decremented. An arc connecting an event to a state variable indicates that the state variable should be incremented when the event is fired. While the state of the submodel constructed of state variables s_0 and s_1 and events e_0 and e_1 does not depend directly on events fired by the second submodel that is constructed of state variables s_0 and s_1 and events e_0 and e_1, it can still depend on the state of the other submodel in two key ways.

- *Rate dependence*: The two submodels can be said to have *rate dependence* if the transition rate function specification Λ of an event in one submodel is defined in terms of the state variables in the other submodel.

- *External dependence*: When an event in one submodel has an event-enabling function specification, Φ, defined in terms of the state or state variables of another submodel, we say the submodels feature *external dependence*.

A third type of structure, synchronization dependence, is discussed in [26]. It corresponds to simultaneous changes in the values of two or more state variables in two or more submodels. We expand upon this notion in light of our concern with rare events to describe a new structural feature, illustrated in Figure 1b.

- Δ-*dependence*: When the firing of an event changes the value of state variables in two or more otherwise independent submodels, we say that they feature Δ-*dependence*.

While the submodel shown in Figure 1b would not normally be considered near-independent, if e_4 and e_5 represent rare events, the states of the two submodels depend on each other only rarely, when those events fire.

For our technique, we propose to exploit these dependence relationships when the event that is the root cause of the dependence is rare.

3.2 Model Dependency Graph

To exploit the structural properties described in the previous subsection, we must first analyze the model in terms of our defined dependence relationships. To do so, we construct a model dependency graph.

Definition 6. *The model dependency graph of a model M is defined as an undirected labeled graph, $G_M = (V, A, L)$, where V is a set of vertices composed of two subsets $V = V_S \cup V_E$, A is a set of arcs connecting two vertices such that one vertex is always an element of the subset V_S and one vertex is always an element of the subset V_E, and L is a set of labels applied to elements of A from the set $\{\Phi, \Lambda, \Delta, R\}$. Let V_S denote the subset of vertices representing the state variables $S \in M$, and V_E denote the subset of vertices representing the events $E \in M$.*

Algorithm 1 Compute the model dependency graph, G_M

$V_S \leftarrow \emptyset$
$V_E \leftarrow \emptyset$
$\forall s_i \in S, V_S \leftarrow V_S \cup v_{s_i}$
$\forall e_i \in E, V_E \leftarrow V_E \cup v_{e_i}$
$V \leftarrow V_S \cup V_E$
$A \leftarrow \emptyset$
$L \leftarrow \{\Phi, \Lambda, \Delta, R\}$
for all $\Phi(e_i, q), e_i \in E, q \in 2^{S \times \mathbb{N}}$ **do**
  for all $s_j \in q$ such that s_j is not defined in q for all possible assignments **do**
    $A \leftarrow v_{s_j} v_{e_i}$ with label Φ
  end for
end for
for all $\Lambda(e_i, q), e_i \in E, q \in 2^{S \times \mathbb{N}}$ **do**
  for all $s_j \in q$ such that s_j is not defined in q for all possible assignments **do**
    $A \leftarrow v_{s_j} v_{e_i}$ with label Λ
  end for
end for
for all $\Delta(e_i, q), e_i \in E, q \in 2^{S \times \mathbb{N}}$ **do**
  for all $s_j \in q$ such that s_j is not defined in q for all possible assignments **do**
    $A \leftarrow v_{s_j} v_{e_i}$ with label Δ
  end for
end for
return $G_M \leftarrow (V, A, L)$

By performing algorithm 1, we construct G_M using the model specification from definition 1. This results in G_M having a node for every state variable in S and event in E, and arcs connecting an arbitrary state variable s_i to an arbitrary event e_j, iff

- The enabling condition of e_j depends on the value of s_i. This indicates an *external dependence* and is marked with the label Φ.

- The rate of the event e_j depends on the value of s_i. This represents a *rate dependence* and is marked with the label Λ.
- The firing of e_j changes the value of s_i. This represents a Δ *dependence* and is marked with the label Δ.

Proposition 1. *For a given model M, the graph G_M constructed by algorithm 1 represents all possible dependencies between all events and state variables in a model.*

Proof. Proof by contradiction. Suppose there exist some state s_i and some event e_j that are directly dependent and not captured by G_M. All direct dependencies due to Φ, Λ, and Δ are encoded in G_M as labeled edges by algorithm 1; thus, the dependence must be one outside of the definition of Φ, Λ, and Δ. Since M is defined using definition 1, no such direct dependencies can exist. Thus our graph represents all possible direct dependencies.

Suppose there exist two elements $\alpha, \beta \in S \cup E$ that are indirectly dependent and not captured by the graph G_M. They are indirectly dependent if they are both state variables and the value of α can affect the value of β or vice versa. If α is a state variable and β is an event, they are indirectly dependent if the firing of β can affect the value of α, or if the value of α can affect the value of $\Phi(\beta, q), \Lambda(\beta, q)$, or $\Delta(\beta, q)$ for $q \in 2^{S \times \mathbb{N}}$. If α and β are both events, they are indirectly dependent if the firing of α can affect the value of $\Phi(\beta, q), \Lambda(\beta, q)$, or $\Delta(\beta, q)$ for $q \in 2^{S \times \mathbb{N}}$ and vice versa.

In the model this indirect dependency will take the form of a series of event firings and state variable changes, each of which is either enabled by, or has its rates set by, a state variable upon which it depends, and which changes the value of subsequent state variables upon which future events depend. For such a sequence $\{s_i, e_j, s_k, e_l, \ldots\}$ to exist, every consecutive pair in the sequence $(s_i, e_j), (e_j, s_k), (s_k, e_l), \ldots$ must be directly dependent. If this is true, then from algorithm 1, there must exist a path defined by a series of vertices in V and arcs in A from the vertex representing the starting state or event in the sequence, to the vertex representing the final state or event in the sequence, such that path visits each vertex that corresponds to intermediate states and events in the sequence. Therefore the indirect dependence of α and β must be represented by the path $v_\alpha, v_\alpha v_i, v_i, \ldots, v_j, v_j v_\beta, v_\beta$. □

In addition to analyzing the dependencies in M, we also wish to study the dependencies implied by the pair (M, Θ_M). Recall from definition 4 that a reward variable may have one of two reward structures, rate reward or impulse reward. In the case of rate rewards, the reward structures are defined in terms

of a mapping between $2^{S\times\mathbb{N}}$ and $\mathbb{R}$; for impulse rewards, the reward structures are defined in terms of a mapping between E and $\mathbb{R}$. These relations imply a new category of dependence.

- *Reward dependence*: When a reward variable $\theta_i \in \Theta_M$ exists such that its reward structure is defined in terms of the state variables of two submodels, or in terms of the events of two submodels, we say they feature reward dependence.

Unlike the other forms of dependence we have defined, reward dependence is more likely to inhibit decomposition of our model into submodels, for reasons that will become apparent in Section 5. We add these dependencies to G_M using algorithm 2.

Algorithm 2 Add reward dependencies to the model dependency graph

```
Given G_M as generated from algorithm 1
V_Θ ← ∅
∀θ_i ∈ Θ, V_Θ ∪ v_θi
for all θ_i ∈ Θ do
  if θ_i is a rate reward defined over q ∈ 2^(S×ℕ) then
    for all s_j ∈ q such that s_j is not defined in q for all possible assignments do
      A ← v_sj v_θi with label R
    end for
  end if
  if θ_i is a rate reward defined over E' ⊂ E then
    for all e_j ∈ E' do
      A ← v_ej v_θi with label R
    end for
  end if
end for
return G_M ← (V, A, L)
```

We add reward dependencies to the model dependency graph by first creating a new subset of vertices, V_Θ, which represent our reward variables. Arcs in A can now additionally connect a vertex $v_i \in V_\Theta$ to a vertex $v_j \in V_S$ if v_i represents a rate reward and can connect a vertex $v_i \in V_\Theta$ to a vertex $v_j \in V_E$ if v_i represents an impulse reward. Our previously unused arc label R is used to label all arcs in A that contain at least one vertex in V_Θ, indicating that they represent reward dependencies.

4 Identifying Rare Events

Having constructed our model dependency graph G_M, we wish to attempt decomposition of our model using the encoded dependence relationships, when G_M indicates that they involve rare events. To perform

this decomposition we also need $E_R \subset E$, the set of rare events in M. In this section we will discuss rare events and methods for identifying them in our model.

4.1 Rare Events

Identification of rare events in a model M is not as simple as examining the supplied definition of $\Lambda(e, q)$ for all events. In addition to their locally defined rates, events may be considered rare for reasons such as competition and enabling conditions. Before examining these in detail, we first present a definition of a rare event in terms of a trajectory of model behavior. Recall from Section 2 that a trajectory is characterized by a sequence of states and events, beginning with the initial state and transitioning probabilistically as defined by equation 1. We present our definition of a rare event in terms of trajectories of model behavior. We use T^* to denote the set of all possible trajectories of model behavior, $T \in T^*$ to represent an individual trajectory, $P(T|M)$ to be the probability of witnessing a trajectory T for model M, $\text{obs}(e_R, T)$ to be the number of times event e_R was observed in the multi-set T, $\tau_{0,T}$ to be the starting time of trajectory T, and $\tau_{\omega,T}$ to be the time of the last event firing of the finite trajectory T.

Definition 7. *An event $e_r \in E$ in a model $M = (S, E, \Phi, \Lambda, \Delta)$ is rare if $\mu_{e_r} <$ some μ_{max} where $\mu_{e_r} = \sum_{T \in T^*} \mu_T \cdot P(T)$, $\mu_T = \frac{\text{obs}(e_R,T)}{\tau_{\omega,T} - \tau_{0,T}}$, and T^* is the set of all model trajectories.*

Definition 7 allows us to capture a broad set of rare events discussed in the following subsections. We call the parameter $\mu_{\max}$ the *partition parameter*, which forms the bounds between rare and non-rare events. Its selection is model-dependent and is discussed at the end of this section.

For a given event e_i, $\Lambda(e_i, q)$ may be given such that it represents a rate that is several orders of magnitude less than that of other events in the model, partitioned by the parameter μ_{max}. In these cases we can classify the local rate of e_i to be rare. In the case of an event with a state-dependent rate (i.e., where $\Lambda(e_i, q)$ varies for different $q \in 2^{S \times \mathbb{N}}$), it may be useful to create two virtual events, $e_{i,1}$ and $e_{i,2}$, with the first virtual event replacing e_i for values of $\Lambda(e_i, q)$ that constitute non-rare events, and $e_{i,2}$ replacing e_i for values that qualify as representing rare events.

Figure 2 illustrates an example in which the local rates defined by the transition rate function do not necessarily differentiate rare events from non-rare events. Assume that the rate of the event labeled e_0 is defined as $\Lambda(e_0, q) = \mu, \forall q \in 2^{S \times N}$, and that the events labeled e_1, e_2, e_3 have rates defined as $\Lambda(e_i, q) = \mu, \forall q \in 2^{S \times N}$ as well. Considering the case when the enabling function is defined for all events except e_0 as

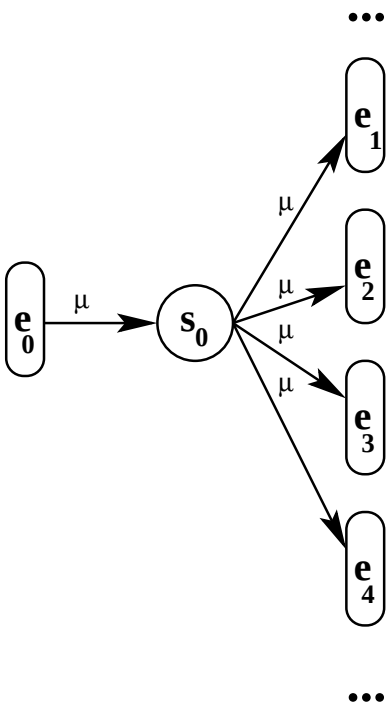


Figure 2: Models exhibiting rare events due to competition.

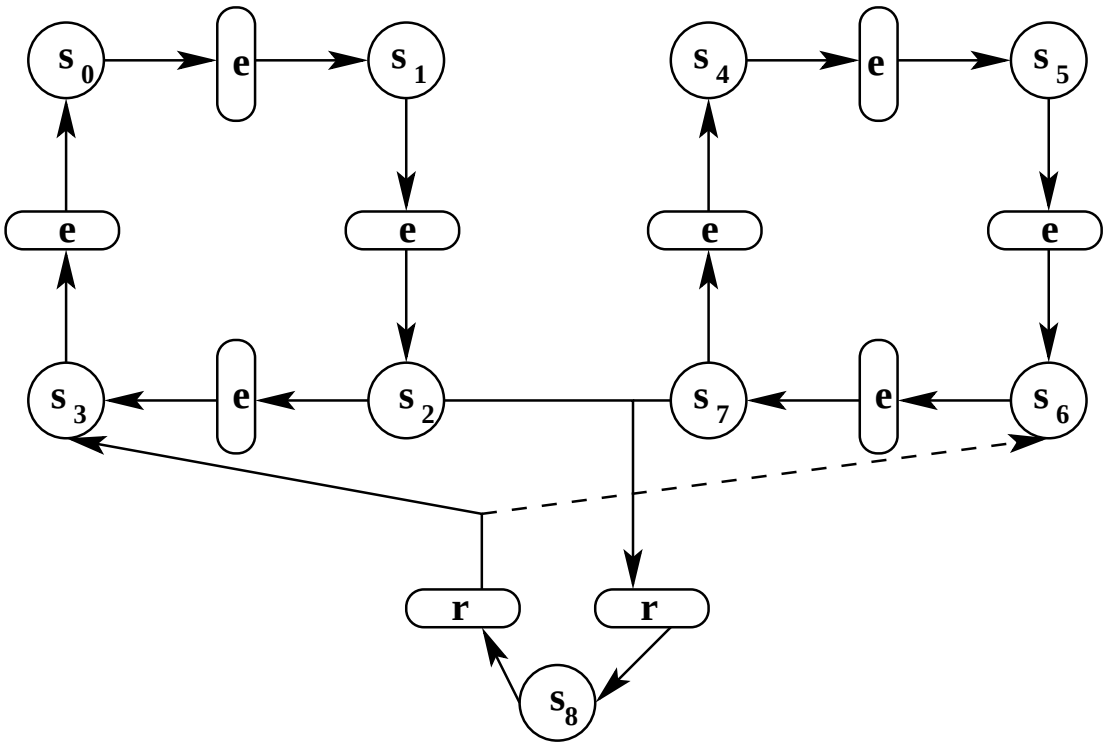


Figure 3: Models exhibiting rare events due to rare enabling conditions.

$$\Phi(e_i, s_0) = \begin{cases} 1, \text{ if } s_0 > 1 \\ 0, \text{ otherwise} \end{cases}$$

and the state transition function is defined in part by $\Delta(e_i, (s_0 = 1)) = (s_0 = 0)$.

If we imagine a similar case in which n such events are in competition, their effective rates might be much lower than the local rates defined in Λ would imply. The effective rate of each event can be easily determined using uniformization.

The final, and most difficult to identify, fashion in which events may be rare is when their enabling conditions defined by Φ are rare. Consider the model presented in Figure 3. Assume that all events labeled either e or r have the same transition rate function, and that the model begins with state variables s_0 and s_4 equal to one, with all other state variables equal to zero. Assume that an event is enabled when all state variables with outgoing arcs pointing to the event have values greater than zero, and is disabled otherwise. Additionally, assume that when an event fires, it decrements by one all state variables with outgoing arcs pointed at the event, and increments by one all state variables with incoming arcs originating in the event.

Although the rates of all events are similar, the enabling conditions of the events labeled r are true far less often than those of the events labeled e. The enabling conditions require that the submodels be "synchronized," i.e., that $s_2 = s_7 = 1$ in order to fire. The enabling condition for the second r requires that $s_8 = 1$, a condition that can only be true after the firing of a rare event, and before any other events have been fired. We call these events rare because their enabling conditions depend on a model state that is rare.

We combine these notions of how an event might qualify as rare by calculating the effective global rate of an event. For an event e_i we solve for the global rate μ_{e_i}, given a specification of the form presented in definition 3, and solve for π^*, the steady-state occupancy probability vector, as follows:

$$\mu_{e_i} = \sum_{\forall \psi_j | \phi(e_i, \psi_j) = 1} \lambda(e_i, \psi_j) \left(\frac{\lambda(e_i, \psi_j)}{\sum_{\forall e \in E | \phi(e, \psi_j) = 1}} \right) \cdot \pi^*[\psi_j] \tag{2}$$

While λ and ϕ in equation 2 are given by the model definition, π^* is not, and can be difficult to calculate. In fact, the ability to solve π^* would likely negate the need for our methods. In models we

have examined that represented real storage and high-performance computing systems, we have not yet encountered a need to find rare events that are rare due to enabling conditions, but in the interest of making our approach broad, we propose a method for approximating π^* that may yield enough accuracy to identify rare events.

4.2 Algorithm for Estimating Enabling Conditions

We propose the use of a bounded state-space exploration algorithm to approximate π^*, exploiting the fact that it is not necessarily important to derive a precise estimate of π^*. Since the events we are looking for are rarer than other events in the model, as defined by the parameter μ_{max}, our estimate need only be good enough to differentiate events with rate less than μ_{max} from other events in our model.

Given a model M whose state variables S are in some initial marking $q \in 2^{S \times \mathbb{N}}$, we utilize uniformization to transform the continuous time Markov chain (CTMC) describing M (as given in definition 3) and transform it into a discrete time Markov chain (DTMC) using equation 1. We then step through the state space, using Welford's algorithm [27] to form an estimate for π^* that we will call $\hat{\pi}^*$.

Algorithm 3 Estimate $\hat{\pi}^*$

$n \leftarrow 0$
$\pi(0)[\psi_0] \leftarrow 1$
$\forall \psi_i | i \neq 0, \pi(0)[\psi_i] \leftarrow 0$
$\forall \psi_i, v[\psi_i] \leftarrow 0$
$\forall e_i \in E, \varphi[e_i] \leftarrow 0$
$\hat{\pi}^* \leftarrow \pi(0)$
$\forall e_i \in E, \hat{\varphi}[e_i] \leftarrow 0$
while Stopping criteria not met **do**
  Let $x[]$ be the next state probability vector, given ϕ, λ, and δ, and the current state occupancy probability vector $\pi(n)$.
  $n \leftarrow n + 1$
  $d \leftarrow x - \hat{\pi^*}$
  $v \leftarrow v + (d^T d)\frac{n-1}{n}$
  $\hat{\pi^*} \leftarrow \hat{\pi^*} + d\frac{1}{n}$
  $\pi(n) \leftarrow x$
  $\forall e_j \in E, \varphi[e_j] = \sum_{\psi_k \in \Psi} \pi(n)[\psi_k] \cdot \phi(e_j, \psi_k)$
  $\varphi_d \leftarrow \varphi\hat{\varphi}$
  $\varphi_v \leftarrow (\varphi_d^T \varphi_d)\frac{1}{n}$
  $\hat{\varphi} \leftarrow \hat{\varphi} + \varphi_d\frac{1}{n}$
  $\varphi_s \leftarrow \sqrt{\varphi_v[\psi_i]\frac{1}{n}}, \forall \psi_i \in \Psi | \varphi_v[\psi_i] \neq 0$
  $c \leftarrow \varphi_s \frac{t^*}{\sqrt{n-1}}$
end while
return $\hat{\pi}^*, \hat{\varphi}, c$

The algorithm builds an estimate $\hat{\pi}^*$ of π^* through bounded exploration of the state space of the model M. Beginning with the initial state, we use equation 1 to explore all possible next states, and the probability of being in those next states during the next time step. We build a running estimate of $\hat{\pi}^*$ as well as the variance of the estimates of the state occupancy probability for each state using Welford's one-pass algorithm, and store it in the vector v.

We propose stopping criteria based on two measures. First, a certain number of steps must have been taken in the state-space exploration. At a minimum, we should take enough steps so as to form an estimate of the value of $\varphi[e_i]$, the probability of the event e_i being enabled for all $e_i \in E$. More formally,

$$\hat{\varphi}[e_i] \sim \sum_{\forall \psi_j \in \Psi} \pi^*[\psi_j] \cdot \phi(e_i, \psi_j) \tag{3}$$

Algorithm 3 calculates this estimate using a version of Welford's one-pass algorithm modified for use with vectors, and also calculates confidence bounds using t^*, the critical values for an $n-1$ sample student-t distribution for a $1 - \alpha \cdot 100\%$ confidence interval, providing us with the bounds

$$\hat{\varphi}[e_i] - c \leq \sum_{\forall \psi_j \in \Psi} \pi^*[\psi_j] \cdot \phi(e_i, \psi_j) \leq \hat{\varphi}[e_i] + c \tag{4}$$

It should be noted that our estimates need not be what might be traditionally considered "good" estimates. Provided that we can distinguish between rough classes of enabling conditions, expecting many orders of magnitude difference between rare and non-rare states, and given a reasonable estimate, $\hat{\pi}^*$, and non-overlapping values of $\varphi[e_i], \forall e_i$, a rough estimate may suffice to identify those events in our model that are rare due to rare enabling conditions.

If for some reason we cannot identify events that are rare due to rare enabling conditions, we can still identify those with locally rare rates, or rare rates due to competition, approximating μ_{e_i} as $\hat{\mu}_{e_i}$,

$$\hat{\mu}_{e_i} = \sum_{\forall \psi_j | \phi(e_i, \psi_j) = 1} \lambda(e_i, \psi_j) \left(\frac{\lambda(e_i, \psi_j)}{\sum_{\forall e \in E | \phi(e, \psi_j) = 1}} \right) \tag{5}$$

Given either μ_{e_i} based on an estimation of $\hat{\pi}^*$, or $\hat{\mu}_{e_i}$ based on the assumption that $\forall e_i, e_j \in E, \varphi[e_i] \sim \varphi[e_j]$, we can now identify a certain subset $E_R \subset E$ as rare events, given some partitioning scheme. Choice of a static parameter with which to partition has been well-studied for hybrid simulation [28]. Some algorithms even propose methods for dynamic partitioning while simulating a given system [29].

The exact choice of partitioning method is unimportant for the correctness of the general application of our technique, but some approaches may have advantages when applied to certain specific models. For the rest of our discussion we will assume a static partitioning parameter $\mu_{\max}$ that defines the maximum estimated rate that results in classification of an event as a rare event.

5 Decomposing Models with Rare Events

In Section 2 we discussed the definition of a model M and a set of reward variables Θ_M associated with the model. In Section 3 we presented an algorithm for constructing a model dependency graph G_M that accounts for all possible direct and indirect dependencies inherent in M and the reward variables Θ_M associated with it. In Section 4 we discussed methods for identifying a subset of events $E_R \subset E$ in M that can be considered rare. In this section we present an algorithm for decomposing M, based on the graph G_M, into a set of n submodels $\Xi = \{\xi_0, \xi_1, \ldots, \xi_n\}$ that can be considered independent in the absence of the firing of a rare event. Additionally, we discuss how to repartition M using G_M after a rare event has fired to produce a new set of independent submodels.

5.1 Decomposing the Model Dependency Graph

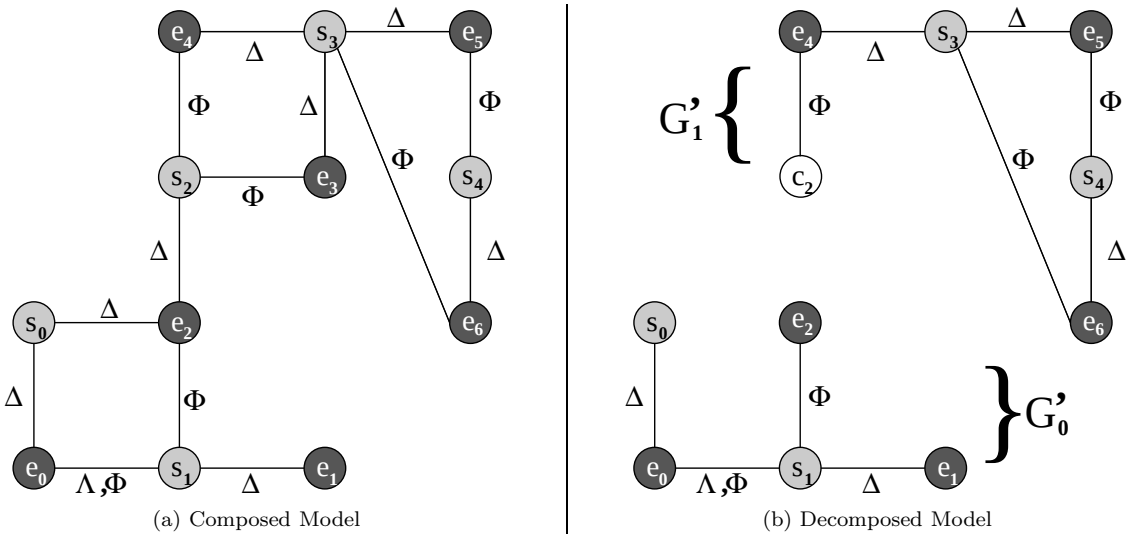
Given the model dependency graph G_M and the set of rare events E_R we produce a decomposed model dependency graph.

Definition 8. *Let G'_M denote the model dependency graph for M in which all Δ-dependencies that involve events in E_R have been removed. For every vertex associated with a state variable whose only Δ-dependencies involve events in E_R, we replace those vertices with new vertices from a set V_C, representing constant state variables whose values are equal to their initial conditions. All vertices that represent events with rates dependent on state variables that are now represented by constant vertices are examined. If such events have transition rate function specifications such that $\Lambda(e, q) = 0$ for all valid $q \in 2^{S \times \mathbb{N}}$ given V_C or have enabling function specifications such that $\Phi(e, q) = 0$ for all valid $q \in 2^{S \times \mathbb{N}}$ given V_C, they are removed. All dependencies of removed events are also removed. The process is repeated, examining all V_S and V_E iteratively until no new vertices are removed.*

We present a method for generating G'_M using G_M and E_R in algorithm 4.

Algorithm 4 Compute G'_M by removing rare-event-based dependencies.

$G'_M = (V', A', L') \leftarrow G_M$
$P \leftarrow E_R$
while $P \neq \emptyset$ **do**
  Remove all edges in A' in which one member of the edge is a vertex representing an event in P. If the edge is labeled Φ or Λ and the event vertex corresponds to an event in E_R, do not remove it.
  P $\rightarrow \emptyset$
  for all $v_i \in V'$ that represent state variables **do**
    if $!\exists v_i v_j \in A'$ such that $v_i v_j$ has label Δ **then**
      $V' \leftarrow V' \setminus v_i$
      Create a new constant vertex $v_{c_i} \in V_C$
      $V' \leftarrow V' \cup v_{c_i}$
      Associate a value equal to the initial marking of $s_i \in S$ associated with v_i with v_{c_i}
    end if
  end for
  for all $v_j \in V$ representing Events **do**
    if $\exists v_i | v_i v_j \in A'$ labeled Φ such that $v_i \in V_C$ **then**
      if $!\exists q \in 2^{S \times \mathbb{N}}$ such that q is consistent with the constant markings associated with vertices in V_C and $\Phi(e_j, q) = 1$ **then**
        $P \leftarrow P \cup v_j$
      end if
    end if
    if $\exists v_i | v_i v_j \in A'$ labeled Λ such that $v_i \in V_C$ **then**
      if $!\exists q \in 2^{S \times \mathbb{N}}$ such that q is consistent with the constant markings associated with vertices in V_C and $\Lambda(e_j, q) \neq 0$ **then**
        $P \leftarrow P \cup v_j$
      end if
    end if
  end for
  $V' \leftarrow V' \setminus P$
end while
return $G'_M = (V', A', L')$



(a) Composed Model

(b) Decomposed Model

Figure 4: Example decomposition of a model dependency graph G_M to G'_M.

The graph G'_M that results from the application of algorithm 4 to G_M and E_R is then used to determine if a valid partition of the model M exists for our technique. If G'_M defines multiple unconnected sub-graphs, $G'_M = \{g'_0 \cup g'_1 \cup \ldots\}$, a valid partition exists. If it does not, our technique is not applicable. The sub-graphs of G'_M correspond to the submodels in our partition Ξ. For a given sub-graph, $g'_i = (V'_i, A'_i)$, for each $v'_j \in V'_i$ such that $v'_j \in V_S$, we add the corresponding state variable to ξ_i. For each $v'_j \in V'_i$ such that $v'_j \in V_E$, we add the corresponding event to ξ_i. In addition, for each $\xi_i \in \Xi$ we restrict the definitions of $\Phi(e_j, q), \Lambda(e_j, q)$ and $\Delta(e_j, q)$ to $e_j \in \xi_i$ and $q \in 2^{S_{\xi_i} \times \mathbb{N}}$ such that $S_{\xi_i} \in \xi_i$.

To further explain our decomposition algorithm, we present an example model dependency graph and its decomposition in Figure 4. The dependency graph G_M is generated from a high-level model specification using algorithm 1, as shown in Figure 4a. We assume that the event represented by the vertex e_2 is a rare event in the model. Applying algorithm 4, we begin with a set $P = e_2$, remove those edges labeled Δ that involve e_2, and clear P. We note that the state variable represented by s_2 remains constant in the absence of e_2's Δ edge, and replace it with a constant vertex c_2 with value equal to its initial conditions. We then find that events e_3 and e_4 have dependencies that are marked by Φ labeled as arcs, indicating an external dependency. Assume that the enabling conditions of e_3 are not met by the constant value of c_2, but the enabling conditions of e_4 are met. We add e_3 to the now-empty P and iterate again, this time removing e_3. We do not remove s_3, despite the Δ-dependency, as s_3 has an additional Δ-dependency on e_4. At this point, P is empty, and we exit the algorithm, yielding $G'_M = G'_0 \cup G'_1$ as

shown in Figure 4.

5.2 Analyzing Reward Variable Dependencies

In Section 3 we used algorithm 2 to add reward variables and their dependence relationships to G_M. They were subsequently preserved in G'_M by algorithm 4. These dependencies prevent decomposition of otherwise independent sub-graphs by maintaining connectivity based on reward dependence. Additionally, they help us choose solution methods for submodels in Ξ.

Proposition 2. *In the absence of the firing of a rare event, the reward variable θ_i is independent from a submodel ξ_j if no direct dependence exists in G'_M from θ_i to a vertex in g'_j.*

Proof. If a direct dependence existed between a reward variable θ_i and a state or event in ξ_j then algorithm 2 would add an edge to G'_M connecting θ_i to a vertex in g'_j, and thus a path would exist. If there were an indirect dependency between θ_i and a vertex in g'_j, then a path would exist between a vertex v_k that has a direct dependency with θ_i and a vertex in g'_j. If such a path existed, then v_k would be a vertex in g'_j, and thus θ_i would have an edge connecting directly to a vertex in g'_j. □

Given G'_M, we divide all submodels in Ξ defined by the independent sub-graphs of G''_M into two sets: those upon which reward variables do and do not depend in the absence of rare events. These sets of submodels are called Ξ_R and $\Xi_{!R}$, respectively.

6 Solving the Decomposed Model

The primary contributions of our research are the methods proposed in Sections 3, 4, and 5, which provide a potential means to decompose a model M into a set of submodels $\Xi_R, \Xi_{!R} \in \Xi$ based on structural features in the model M that involve dependence relationships with rare events. While we believe our methods are useful for a variety of solution techniques, we present in this section an additional contribution in the form of an algorithm for hybrid simulation of decomposed models, and a discussion of complementary solution methods from the literature. Our hybrid simulation algorithm was designed to help study the dependability characteristics of deduplicated data storage systems (in joint work with IBM Almaden Research Center). A simplified example model based on our joint work is presented in Section 7 to illustrate potential improvements provided by our decomposition methods.

6.1 Hybrid Simulation of Rare-Event Decomposed Systems

Our study of rare-event-based decomposition methods was motivated by a desire to study the dependability characteristics of storage systems that utilize data deduplication, in a fault environment characterized by rare events. In order to estimate the value of reward variables defined for models of these systems, we have employed our decomposition methods and a hybrid simulation algorithm.

When solving our model, we view trajectories of model execution as a time series

$$\tau_0 \longrightarrow \tau_1 \longrightarrow \tau_2 \longrightarrow \tau_3 \longrightarrow \dots \tag{6}$$

where τ_0 represents our start time, and each subsequent τ_i represents the firing of a rare event. Given a model of our system, M, and the set of reward variables, Θ_M, we produce the model dependency graph G_M, set of rare events E_R, decomposed model dependency graph G'_M, and set of submodels $\Xi_R, \Xi_{!R} \in \Xi$. Using these submodel classifications and the subset $\Xi_{E_R} \in \Xi$ of submodels containing rare events, we produce two new sets of submodels,

$$\Xi_{\text{Sim}} = \Xi_R \cup (\Xi_{!R} \cap \Xi_{E_R}) \tag{7}$$

$$\Xi_{\text{Num}} = \Xi_{!R} \setminus (\Xi_{!R} \cap \Xi_{ER}). \tag{8}$$

Algorithm 5 Hybrid Simulation of M

Given a model M, reward variables Θ_M, and initial values $q_0 \in 2^{S \times \mathbb{N}}$ for M.
while Stopping criteria for our reward variables Θ_M have not been met **do**
  Set S for model M using q_0.
  Generate G_M from M using Algorithms 1 and 2.
  Generate G'_M and Ξ from G_M using Algorithm 4.
  Derive Ξ_{Sim} and Ξ_{Num} using equations 7 and 8.
  Solve Ξ_{Sim} using Discrete Event Simulation until the next event is in the set E_R.
  Generate $\pi^*_{\xi_i}$ for each submodel in Ξ_{Num}.
  Generate a random variate for each submodel $\xi_i \in \Xi_{\text{Num}}$ using $\pi^*_{\xi_i}$ to define the probability mass function of a random variable.
  Recompose Ξ to M. Use Discrete Event Simulation to solve M for the next event.
  Store the current state of the model in q_0.
end while

The set Ξ_{Sim} has all submodels that contain either a rare event or a reward dependency. The set Ξ_{Num} has all submodels that contain neither rare events nor reward dependencies. From proposition 2

we know that the evaluation of reward variables does not depend on Ξ_{Num}. Thus we need only solve the state occupancy probability for all submodels in Ξ_{Num} at the time of the next rare event firing. We do so by making the assumption that the submodels enter steady state between firings of rare events. Simulation of the model M is performed using algorithm 5.

The general improvement offered by this algorithm comes from the reduction of events that must be simulated in order to estimate the effect of rare events in the system. Bucklew and Radeke [30] give a general rule of thumb that in order to estimate the impact of an event with probability ρ, we must process approximately $100/\rho$ simulations. Our method seeks to reduce the number of events that must be processed for each computed trajectory of the simulation by eliminating those events that cannot impact our reward variables without the firing of a rare event.

The actual performance improvement offered by this algorithm varies with the model, and with the degree of dependence of the state variables and events in the model. For models whose resulting Ξ does not have the proper structure, our proposed hybrid simulator may provide no improvement, suggesting that other methods from the literature should be used for solving a set of related submodels. In general, between firings of a rare event, our method will produce a speed-up proportional to the rate at which we remove events from explicit simulation. Thus, given E' as the set of all events $e_i \in \Xi_{\text{Num}} \cup e_j \in M$ such that $e_j \notin \Xi$, our improvement is proportional to

$$\frac{\sum_{e_i \in E'} \lambda(e_i, \psi_i)}{\sum_{e_i \in E} \lambda(e_i, \psi_i)}, \forall \psi_i \in \Psi. \tag{9}$$

6.2 Limitations

Our methods work best in situations where the underlying submodel is loosely coupled, with the points of loose connection being dependent on rare events. We believe this class represents a subset of interesting models used to study dependability characteristics of high-performance computing resources [31] and large-scale storage systems [3, 15, 32]. Additional models for which our techniques are appropriate may also exist, provided they bear structural resemblance to our primary models of interest.

A limitation imposed by our hybrid simulation algorithm is our assumption that submodels in the set Ξ_{Num} reach steady state. While we believe this assumption holds for the systems we have studied, we could relax it by using approximate methods for generating the transient state occupancy probability vector instead of π^* [33].

6.3 Related Solution Methods

The methods of [26] provide an attractive application of our decomposition techniques. While the authors do not provide a means for automatic decomposition, they provide a way to approximate a decomposed system by describing the interaction of a set of submodels with an import graph that solves the resulting system via fixed-point iteration.

Provided that some of the submodels identified by our method represent highly similar sub-systems, our methods might be used with Simultaneous Simulation methods, such as those proposed by [34]. By combining a single-clock, multi-system simulation with adaptive uniformization, these methods simulate all alternative configurations of independent systems simultaneously, reducing the overhead involved in event list management.

Model solution using analytic methods can also benefit from our approach. The state spaces for our submodels are guaranteed to be smaller than that of the model as a whole, potentially decreasing the solution time by solving the submodels and appropriately combining the solutions to form a solution of the overall model. That approach is similar to the methods described in [20].

7 Case Study: Simplified Data Deduplication Storage System

In this section, we present an example based on a simplified deduplicated storage system [35] and demonstrate our technique for models that use rare events and a model dependency graph. Figure 5 provides a high-level diagram of our system. The system consists of a set of n simplified storage systems, each of which can store two objects and suffer from disk failures and fail-silent errors [15] as well as parity pollution [3]. Each storage system can suffer a maximum of two failures before losing data.

Two processes attempt to mitigate faults before they can manifest as data loss: disk rebuilds and a scrubbing process that evaluates disks in the system weekly, checking for silent failures and parity pollution. An I/O workload is used to measure the reads of corrupt data and to evaluate the impact of parity pollution.

Each subsystem can store two objects from a finite set, representing a subset of a file system. When an object appears more than once in the entire system, the first storage system with a copy stores the object instance and the rest of the storage systems store deduplicated objects as references to the original. Correlated failures occur for references when an instance is lost.

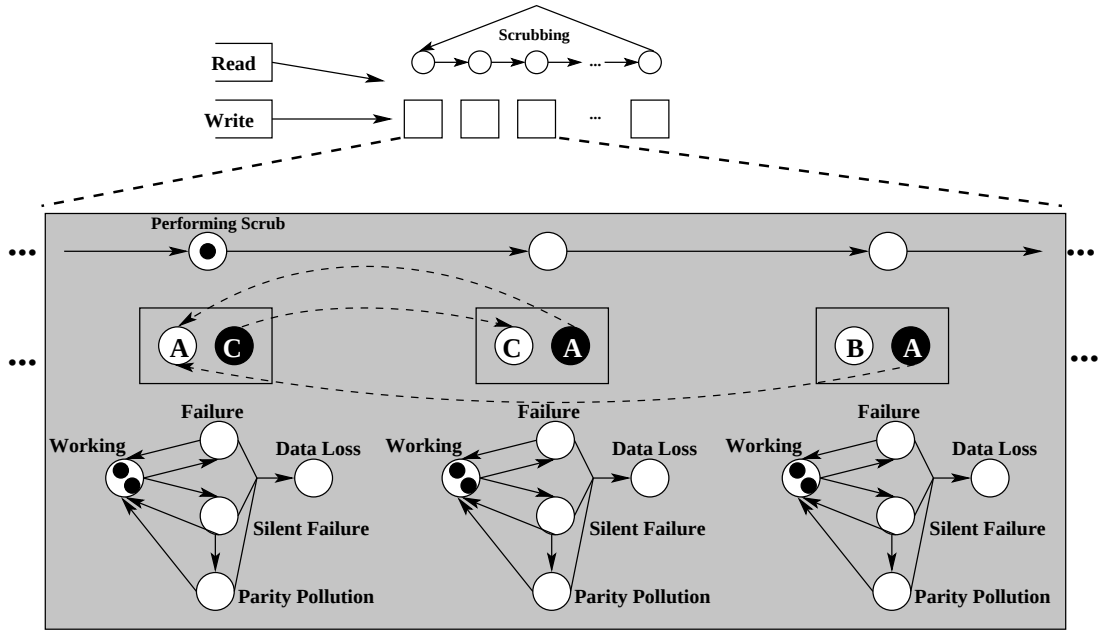


Figure 5: Simplified deduplication system modeling full disk failures, undetected disk errors, and parity pollution.

Definition 9. *Summarized System Model*

State Variables

- $file_{0,0}, \ldots, file_{0,n-1}$*: First file on each of n storage subsystems.*
- $file_{1,0}, \ldots, file_{1,n-1}$*: Second file on each of n storage subsystems.*
- $working_0, \ldots, working_{n-1}$*: Number of working drives on each of n storage subsystems.*
- $failed_0, \ldots, failed_{n-1}$*: Number of whole disk failures on each of n storage subsystems.*
- $silent_0, \ldots, silent_{n-1}$*: Number of silent disk failures on each of n storage subsystems.*
- $polluted_0, \ldots, polluted_{n-1}$*: Number of parity pollutions on each of n storage subsystems.*
- $scrubLocation$*: Current progress of the scrub.*
- $corruptedReads$*: Number of corrupted reads.*

Events

- $\texttt{diskFail}_0, \ldots, \texttt{diskFail}_{n-1}$: *Failure of a disk on each of n storage subsystems. Enabled when* $\texttt{working}_i > 0$. $\Lambda = 10^{-10}, \forall q \in 2^{S \times \mathbb{N}}$.

$$\begin{aligned} \Delta \quad = \quad & \texttt{working}_i = \texttt{working}_i - 1, \\ & \texttt{failed}_i = \texttt{failed}_i + 1. \end{aligned}$$

- $\texttt{silentFail}_0, \ldots, \texttt{silentFail}_{n-1}$: *Silent failure of a disk on each of n storage subsystems. Enabled when* $\texttt{working}_i > 0$. *Rate is given as* 10^{-12} *for all states.*

$$\begin{aligned} \Delta \quad = \quad & \texttt{working}_i = \texttt{working}_i - 1, \\ & \texttt{silent}_i = \texttt{silent}_i + 1. \end{aligned}$$

- $\texttt{rebuild}_0, \ldots, \texttt{rebuild}_{n-1}$: *Rebuilds a failed disk on each of n storage subsystems. Enabled when* $\texttt{failed}_i > 0$. $\Lambda = 0.004, \forall q \in 2^{S \times \mathbb{N}}$.

$$\begin{aligned} \Delta \quad = \quad & \texttt{working}_i = \texttt{working}_i + 1, \\ & \texttt{failed}_i = \texttt{failed}_i - 1. \end{aligned}$$

- $\texttt{advanceScrub}$: *Finishes scrubbing a storage subsystem, and advances to the next in order. Always enabled.* $\Lambda = \frac{10^{-6}}{n_{\text{disks}}}$. *Transition function is defined as*

$$\Delta = \begin{cases} \text{scrubLocation} + 1, & \textit{if scrubLocation} > 1 \\ 0, & \textit{otherwise} \end{cases}$$

- $\texttt{scrub}_0, \ldots, \texttt{scrub}_{n_{\text{disks}}-1}$: *Set of events, one per disk subsystem. Scrubs and repairs parity and silent failures. The event* $\texttt{scrub}_i$ *is enabled when*

$$\texttt{silent}_i + \texttt{polluted}_i > 0 \text{ and } \texttt{scrubLocation} = i$$

$\Lambda = 100, \forall q \in 2^{S \times \mathbb{N}}$.

$$\begin{aligned} \Delta \quad = \quad & \texttt{working}_i = \texttt{polluted}_i + \texttt{silent}_i, \\ & \texttt{polluted}_i = 0, \texttt{silent}_i = 0 \end{aligned}$$

- $\texttt{write}_0, \ldots, \texttt{write}_{n_{\text{disks}}-1}$*: Set of write events, one per storage subsystem. Enabled while* $\texttt{working}_i > 0$ $\Lambda = 1, \forall q \in 2^{S \times \mathbb{N}}$.

$$\begin{aligned} \Delta \quad = \quad & \texttt{polluted}_i = \texttt{silent}_i, \texttt{silent}_\texttt{i} = 0, \\ & \texttt{file}_{0,i} = \text{DiscreteUniform}(0, n_{\text{file objects}}), \\ & \texttt{file}_{1,i} = \text{DiscreteUniform}(0, n_{\text{file objects}}). \end{aligned}$$

- $\texttt{read}_0, \ldots, \texttt{read}_{n_{\text{disks}}-1}$*: Set of read events, one per storage subsystem. Enabled while* $\texttt{working}_i > 0$. $\Lambda = 99, \forall q \in 2^{S \times \mathbb{N}}$.

$$\Delta = \texttt{corruptedReads} = \texttt{silent}_i$$

Reward variables defined for the model include a rate reward defined when $\texttt{working}_i = 0$, which checks the state of each file on the subsystem, including whether it is a reference. A reward variable is also defined for `corruptedReads` over the lifetime of the system.

Decomposition produced a submodel that accounted for the scrubbing process, each storage subsystem, and the I/O workload, all falling into the category Ξ_{Num}. A separate submodel was generated for the fault process and disk repair state, falling into the category Ξ_{Sim}. The state occupancy probabilities were calculated for each submodel prior to a rare event firing, and upon firing G'_M was recalculated, with the scrubbing process, I/O workload, and all storage subsystems that dependent on the error (i.e., that have references to object instances on the faulty storage subsystem) composed in a single submodel. This submodel was simulated until the failure was mitigated by $\texttt{scrub}_i$ or $\texttt{rebuild}_i$, which are both identified by our method as rare events due to rare state, allowing us to recalculate G'_M and decompose the failed components on successful repair.

Model	States
3 Storage Subsystems	
Complete Model	12,288,000
Submodels ξ_0, ξ_1, ξ_2	16
Submodels ξ_3, ξ_4, ξ_5	1
Submodels ξ_6	3
10 Storage Subsystems	
Complete Model	$\sim 10^{23}$
Submodels $\xi_0, \ldots, \xi_9$	16
Submodels $\xi_{10}, \ldots, \xi_{19}$	1
Submodels ξ_{11}	10
100 Storage Subsystems	
Complete Model	$\sim 10^{222}$
Submodels $\xi_0, \ldots, \xi_{99}$	16
Submodels $\xi_{100}, \ldots, \xi_{199}$	1
Submodels ξ_{200}	100

Figure 6: Comparison of the state space of the complete model and decomposed submodels.

Figure 6 presents the state spaces of the submodels, and of the composed model, with the state variable `corruptedReads` removed. Although `corruptedReads` has potentially infinite values, producing an infinite state space, its constant nature between rare events for models in Ξ_{Num} allowed us to numerically solve large portions of the model provided we simulated those in Ξ_{Sim}. As can be seen, our methods produced a drastic improvement in the state spaces of the submodels to be solved, when compared with the complete model.

Direct comparisons for model solution performance between Discrete Event Simulation and Hybrid Simulation were not possible, since our Discrete Event Simulator did not process all planned simulated failures within the time allotted. Our decomposition-based solution was able to deliver results promptly, processing an average of around 10^5 events per rare event. To provide a better comparison between the approaches, the I/O workload model was removed, and we solved the resulting model using both methods.

Figure 7 demonstrates the increased performance our simulator provides, for various ratios of slowest events (`silent`) to fastest events (`rebuild`) in the model. At a ratio of 10^{-4}, we cannot choose an appropriate μ_{max}, as the scrubbing process and all faults are of the same order. Our method fails to find a valid decomposition, resulting in a performance that is slightly worse than that of the Discrete Event Simulator, reflecting our increased overhead. At a ratio of 10^{-5}, though we can set μ_{max} to identify `silent` as rare, our algorithm fails to find a useful decomposition, owing to the indirect dependence

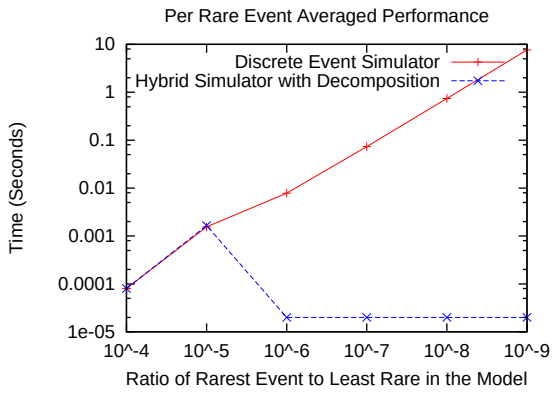


Figure 7: Comparison of performance of a Discrete Event Simulator and Hybrid Simulator with Decomposition.

between the two events which arises because of the external dependencies of each `silent` and `fail` event, when the external dependencies relate to the same state variable. The result is a slightly worse performance for our algorithm, due to overhead.

For ratios of 10^{-6} and above, we show constant performance, as our decomposition does not change. For all of the parameter values, we are able to obtain an ideal decomposition, simulating only when a rare event occurs, and before it has been mitigated.

While our example seems ideal for our algorithm, it is based on a real system that we are currently studying with our industry partners. Additionally, it is likely that our methods will provide reasonable improvements in performance for other similar systems made up of smaller subsystems, such as high performance clusters [31] and networks.

8 Conclusion

We have presented a new approach for decomposing models that contain rare events. This approach is based on the degree of independence of potential subsystems of the model with respect to rare events and the reward variables to be calculated. When portions of the model exhibit near-independence relating to rare events in the system, we can exploit these relationships to solve large, complex models efficiently and exactly. We provide an algorithm for our technique that automatically builds and examines a model dependency graph that characterizes all dependence relationships using a high-level specification of the model. All model state variables and events are converted into nodes in the graph, which are

connected based on the definition of the model event-enabling function specification, transition rate function specification, and state transition function specification. Our presented techniques can be used as the input for many solution techniques, including those discussed in Section 6. Using the Möbius simulator and state-space generator, we obtained a large reduction in the size of the state space for our example model.

Acknowledgments

This material is based upon work supported by an IBM Ph.D. Fellowship. The authors would like to thank the storage group at IBM Almaden for their input, and ongoing collaboration that enabled us to complete this technical report. We would also like to thank Jenny Applequist for her editorial comments, as well as Robin Berthier, Ryan Lefever, Elizabeth LeMay, Kristin Y. Rozier, Sankalp Singh, and Saman Aliari Zonouz for their comments.

References

[1] E. Strohmaier, J. Dongarra, H. Meuer, and H. Simon, "The marketplace of high-performance computing," *Parallel Comput. 25*, pp. 1517–1544, 1999.

[2] Y. Liang, Y. Zhang, A. Sivasubramaniam, M. Jette, and R. Sahoo, "Bluegene/l failure analysis and prediction models," *DSN*, pp. 425–434, 2006.

[3] J. L. Hafner, V. Deenadhayalan, W. Belluomini, and K. Rao, "Undetected disk errors in RAID arrays," *IBM J Research and Development 52*, no. 4, pp. 413–425, 2008.

[4] A. Z. Broder, "Identifying and filtering near-duplicate documents," in *CPM*. Springer, 2000, pp. 1–10.

[5] L. Freeman, "How safe is deduplication," NetApp, Tech. Rep., 2008. [Online]. Available: http://media.netapp.com/documents/tot0608.pdf

[6] S. Derisavi, P. Kemper, and W. H. Sanders, "Lumping matrix diagram representations of Markov models," in *DSN*, 2005, pp. 742–751.

[7] G. Ciardo and A. Miner, "Storage alternatives for large structured state space," in *TOOLS*. Springer, 1997, pp. 44–57.

[8] D. Deavours and W. H. Sanders, "On-the-fly solution techniques for stochastic Petri nets and extensions," *IEEE TSE 24*, pp. 889–902, 1998.

[9] M. Davio, "Kronecker products and shuffle algebra," *IEEE TC 30*, no. 2, pp. 116–125, 1981.

[10] P. Kemper, "Numerical analysis of superposed GSPNs," in *PNPM*, 1995, pp. 52–61.

[11] P. Buchholz and P. Kemper, "Numerical analysis of stochastic marked graph nets," *IEEE PNPM*, p. 32, 1995.

[12] E. de Souza e Silva and P. M. Ochoa, "State space exploration in Markov models," in *SIGMETRICS*, 1992, pp. 152–166.

[13] W. D. Obal II, M. G. McQuinn, and W. H. Sanders, "Detecting and exploiting symmetry in discrete-state Markov models," in *PRDC*, 2006, pp. 26–38.

[14] P. Buchholz, "Hierarchical Markovian models: Symmetries and reduction," *Performance Eval. 22*, pp. 93–100, 1995.

[15] E. Rozier, W. Belluomini, V. Deenadhayalan, J. Hafner, K. K. Rao, and P. Zhou, "Evaluating the impact of undetected disk errors in raid systems," in *DSN*. IEEE, 2009, pp. 83–92.

[16] P. Shahabuddin, "Importance sampling for the simulation of highly reliable Markovian systems," *Manage. Sci. 40*, pp. 333–352, 1994.

[17] P. W. Glynn and D. L. Iglehart, "Importance sampling for stochastic simulations," *Manage. Sci. 35*, pp. 1367–1392, 1989.

[18] H. Kahn and T. E. Harris, "Estimation of particle transmission by random sampling," in *NBS AMS*, 1951, pp. 27–30.

[19] M. J. J. Garvels, "The splitting method in rare event simulation," Ph.D. dissertation, Univ. of Twente, Enschede, 2000.

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

[21] K. S. Trivedi and R. M. Geist, "Decomposition in reliability analysis of fault-tolerant systems," *IEEE Trans. on Reliability*, vol. R-32, no. 5, pp. 463–468, 1983.

[22] W. D. Obal II, "Measure-adaptive state-space construction methods," Ph.D. dissertation, 1998.

[23] W. Sanders and J. Meyer, "A unified approach for specifying measures of performance, dependability, and performability," in *Dependable Computing for Critical Applications 4*. Springer, 1991, pp. 215–237.

[24] R. A. Howard, *Dynamic Probabilistic Systems. Vol II: Semi-Markov and Decision Processes*. New York: Wiley, 1971.

[25] J. F. Meyer, "On evaluating the performability of degradable computing systems," *IEEE TC 29*, pp. 720–731, 1980.

[26] G. Ciardo and K. S. Trivedi, "A decomposition approach for stochastic reward net models," *Performance Evaluation 18*, no. 1, pp. 37–59, 1993.

[27] L. M. Leemis and S. K. Park, *Discrete-Event Simulation: A First Course*. Prentice-Hall, 2005.

[28] T. R. Kiehl, R. M. Mattheyses, and M. K. Simmons, "Hybrid simulation of cellular behavior," *Bioinformatics 20*, pp. 316–322, 2004.

[29] H. Salis and Y. Kaznessis, "Accurate hybrid stochastic simulation of a system of coupled chemical or biochemical reactions," *J Chemical Physics 122*, no. 5, 2005.

[30] J. Bucklew and R. Radeke, "On the Monte Carlo simulation of digital communication systems in Gaussian noise," *IEEE Trans. Comm. 51*, no. 2, pp. 267–274, 2003.

[31] S. Gaonkar, E. Rozier, A. Tong, and W. H. Sanders, "Scaling file systems to support petascale clusters: A dependability analysis to support informed design choices," in *IEEE/IFIP DSN*, 2008, pp. 386–391.

[32] A. T. Clements, I. Ahmad, M. Vilayannur, J. Li, and V. Inc, "Decentralized deduplication in san cluster file systems," in *Usenix ATEC*, 2009.

[33] J. D. Diener and W. H. Sanders, "Empirical comparison of uniformization methods for continuous-time Markov chains," in *NSMC*, 1995, pp. 547–570.

[34] S. Gaonkar and W. H. Sanders, "Simultaneous simulation of alternative system configurations," in *PRDC*, 2005, pp. 41–48.

[35] B. Zhu, K. Li, and H. Patterson, "Avoiding the disk bottleneck in the data domain deduplication file system," in *FAST*. USENIX Association, 2008, pp. 1–14.