Simultaneous Simulation of Alternative System Configurations

Shravan Gaonkar and William H. Sanders
Coordinated Science Laboratory,
University of Illinois at Urbana-Champaign, Urbana, IL 61801
{gaonkar,whs}@crhc.uiuc.edu

Abstract

Simulation to obtain reliability and availability estimates has been widely used by system designers to evaluate and compare alternative choices before making design decisions. However, traditionally that approach worked only if significant computer resources were available or designers accepted a significant time delay between design iterations. In this paper, we present an alternative approach to compute measures of interest for a family of models that represent alternative design choices that is significantly more efficient than the traditional approach. The new approach combines the existing single-clock multiple-system simulation with adaptive uniformization. We achieve the speed-up by simulating all the alternative configurations of the discrete-event model simultaneously while amortizing the cost of enabled event set management. That allows us to explore and evaluate multiple configuration settings of a discrete-event model at the same time, significantly increasing the number of alternative versions of the model that are explored in a given amount of time.

1 Introduction

Growth and advances in current technology have extended system size and complexity to such an extent that architects and designers cannot rely solely on traditional evaluation methodologies to design systems. Current modeling techniques for large systems are time-consuming. They do not allow one to evaluate alternative configurations simultaneously, thereby making it expensive to apply efficient methods, like the divide and conquer approach, to pick the optimal design or system configuration. To perform sophisticated analysis of a large number of systems with varying system design parameters, it is important to develop efficient simulation methods that can evaluate a large number of alternative system configurations quickly. This paper develops an efficient parametric technique to evaluate discrete-event systems with multiple parameter values.

In recent years, researchers have focused on evaluating very large discrete-event systems using varied parallel and distributed simulation methods. However, the exploration of techniques to evaluate a large number of alternative configurations of a moderately-sized system has been limited. Such techniques are needed when a system modeler has to evaluate a system with different design parameter values until he or she obtains the optimal configuration. If the system under evaluation is scrutinized carefully, one will notice that changing the system configuration or parameter values does not alter the behavior completely. Rather, much of the system behavior is similar for most of the possible alternative configurations. This fact suggests a way to simulate multiple alternative system configurations simultaneously.

In particular, Vakili [10] and Chen et al. [1] have built simulators that determine the fastest or dominant event from the set of all the alternative configurations. This dominant Poisson process is then used to drive a single discrete-event simulation clock. This technique can achieve a speed-up, relative to standard sequential simulation, but it often suffers from inefficiencies due to the need for "pseudo transitions", particularly in the case where there are many orders of magnitude of difference between the rates in a system, such as the difference between failure and repair rates in highly available systems. These techniques also do not make use of the fact that many of the alternative configurations have similar sets of enabled events. An efficient technique that would exploit this fact would be to consolidate the common events among these configurations into a single union of events and generate a single enabled event set. Using the common enabled event set, it is possible to combine the advantages of the approach of [10] and [1] with adaptive uniformization to build an efficient single-clock multiple simulation engine. The contribution of this paper is thus a methodology that exploits the structural similarity among the alternative configurations and results in an efficient simulation algorithm that evaluates all the alternative configurations of a system simultaneously even when event rates vary greatly, as is often in the case in dependability evaluations.

The rest of the paper is organized as follows. Section 2 provides an overview of the required background related to adaptive simulation of parametric models. Our *simultaneous simulation using adaptive uniformization* algorithm, along with a proof of correctness, is presented in Section 3. Details of the simulator implementation in Möbius [2] are described in Section 4. Experiments exploring the effectiveness of the algorithm are described in Section 5. We conclude in Section 6.

2 Background

2.1 Adaptive Uniformization

This section gives a brief overview of adaptive uniformization as applied to simulation. Readers are referred to [4] and [9] for more detailed descriptions of uniformization and adaptive uniformization. We describe the concept using an example of a two-server queuing system as shown in Figure 1. In the model, a Poisson stream of jobs arrives at rate α and is routed to two exponential servers with service rates μ_{slow} and μ_{fast} with probability p and $(1-p)$, respectively. Each server has a finite buffer whose size is denoted by B_{slow} and B_{fast}. Both servers provide service using the First Come First Serve ($FCFS$) policy. When a job completes, it departs from the system. The maximum state departure rate $\lambda \equiv \alpha + \mu_{slow} + \mu_{fast}$ occurs when both the slow and fast servers are busy. When simulation of the system is done using uniformization, events are generated as Poisson processes with parameter λ. Each event is designated as an arrival event with probability $\frac{\alpha}{\lambda}$, as a potential departure from the slow server with probability $\frac{\mu_{slow}}{\lambda}$, or as a potential departure from the fast server with probability $\frac{\mu_{fast}}{\lambda}$. Whenever an event is designated as a potential service completion, it is possible that the particular server might be idle. In such situations, the event is called a *pseudo transition* and the state of the system does not change.

It is possible to have conditions that would lead to a large number of pseudo transitions. In particular, in the given example, if the routing probability p is set very close to 1, i.e., $p \simeq 1$, and $\mu_{slow} \ll \mu_{fast}$, most of the incoming jobs would be routed to the slow server, but most of the events generated would be designated as departures from the fast server. However, the fast server is idle most of the time, causing the simulator to label many of those events as pseudo transitions. So in such cases, the discrete-event simulator would become inefficient. To alleviate the problem, it is possible to adaptively uniformize based on the state the system is in [9], i.e., to change or adapt the uniformization rate depending on the state of the system as shown below:

$\lambda \equiv \alpha$	When both servers are idle
$\lambda \equiv \alpha + \mu_{fast}$	When the slow server is idle
$\lambda \equiv \alpha + \mu_{slow}$	When the fast server is idle
$\lambda \equiv \alpha + \mu_{fast} + \mu_{slow}$	When both servers are busy

The technique of changing uniformization parameter values depending on the state of the system guarantees that the simulator never fires a pseudo transition, enabling constant computational progress during the execution of the simulation model.

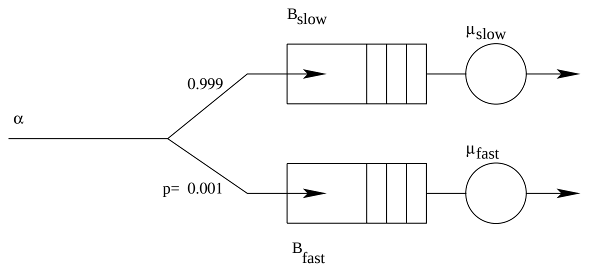

Figure 1. Two-server queuing network.

2.2 Single-Clock Multiple Simulation

In this section, we describe the existing approach taken to simulate alternative design configurations simultaneously, and provide insight into the potential drawback of this technique. Single-Clock Multiple Simulations ($SCMS$) [10] are a class of simulation techniques that exploit the commonality that exists during evaluation of the alternative configurations of a discrete-event system. In the SCMS approach, the clock ticks are generated based upon a dominant process in the system, and the events are chosen by appropriate thinning of the process for each alternative configuration of the system. In this way, the clock update mechanism and the state update mechanism are decoupled. The state update mechanism provides no feedback to the clock update mechanism regarding generation of the next events. If a single configuration of the discrete-event model is being evaluated (as in traditional discrete-event simulation), SCMS would be equivalent to uniformization-based simulation as described in the previous subsection. Note that the single-clock multiple simulation approach does not maintain an enabled event set, which means that it cannot take advantage of the adaptive uniformization.

To illustrate the simultaneous evaluation of multiple configurations using the SCMS technique, consider the example of the two-server queuing system from Figure 1. Let the service rate of the slow server be the design parameter that needs to be determined. For simplicity, suppose that we have n possible choices for the service rate of the slow server. The SCMS would define the dominant Poisson process as $\lambda \equiv \alpha + \mu_{fast} + \widehat{\mu_{slow}}$, which would be used to generate the main clock tick where $\widehat{\mu_{slow}} = max(\mu^i_{slow}); 1 \leq i \leq n$. As in the uniformization approach described earlier, each clock tick is designated as an arrival, as a potential departure from the slow server, or as a potential departure from the fast server, and this information is sent to each alternative configuration of the model. Consider a scenario where the event is designated as a potential departure service from the slow server. Each alternative configuration i with a non-empty buffer would update its state with the probability $\frac{\mu^i_{slow}}{\widehat{\mu_{slow}}}$. This probability effectively thins the Poisson process as needed for each configuration for the departure event from the slow server. Using this technique, all the alternative configurations of the two-server queuing system can be evaluated together with a single dominant clock.

The salient feature of the technique is that it uses a single clock to update all the alternate configurations and eliminates the need to maintain any event list. However, as mentioned by Vakili [10]. This technique gives rise to the possibility of generation of excessive pseudo transitions, creating a potential for inefficiency. In the next section, we describe a technique for simulating a family of alternative configurations of a system by combining certain aspects from the single-clock technique and adaptive uniformization to achieve better efficiency in simulation compared to either of the techniques used individually.

3 Simultaneous Simulation Using Adaptive Uniformization

We now describe an approach based on adaptive uniformization for simulation of discrete-event system with multiple parameter value settings. We first describe the formal model, generalized semi-Markov processes ($GSMPs$), that we use to represent the discrete-event model. We then explain how the GSMP that represents a system being studied can be modified to represent configurations with different parameter values such that all the configurations of the system can be simulated simultaneously. Finally, we describe a simulation algorithm describing the technique by which these alternate configurations can be simulated simultaneously.

3.1 Generalized semi-Markov Processes

A discrete-event simulation can be represented using a generalized semi-Markov process (GSMP) [3]. A GSMP is characterized by a triple, $GSMP = (S, E(s), p(., s, e))$, and by a set of input distributions, $\psi = \bigcup F(.; s, e)$, $e \in E$, defined as follows:

S	=	a set of physical states of a system,
E	=	$\{e_1, e_2, ..., e_k\}$ is a set of finite events for the system,
$E(s)$	=	the set of possible events when the system is in state s,
$p(s', s, e)$	=	the probability of transition from s to s' when event e occurs,
$F(.; s, e)$	=	the probability distribution of the “clock time” of event e when the system is in state s, and
ψ	=	the set of all these clock distributions F for the model.

The example discrete-event model in Figure 1 can be represented using a GSMP as follows:

S = $\{n = [n_{slow}, n_{fast}]$: n_i is the number of jobs on server $i, i = \{slow, fast\}\}$,

E = $\{a, d_{slow}, d_{fast}\}$ (a = arrival, d_i = departure from server $i, i = \{slow, fast\}$),

P = $p([n_{slow}+1, n_{fast}], [n_{slow}, n_{fast}], a) = p$ if $n_{slow} \leq B_{slow}$,
$p([n_{slow}, n_{fast}+1], [n_{slow}, n_{fast}], a)=1-p$ if $n_{fast} \leq B_{fast}$,
$p([n_{slow}-1, n_{fast}], [n_{slow}, n_{fast}], d_{slow})=1$ if $n_{slow} > 0$,
$p([n_{slow}, n_{fast}-1], [n_{slow}, n_{fast}], d_{fast})=1$ if $n_{fast} > 0$, and

ψ = $\{1\text{-}e^{(-\lambda x)}, 1\text{-}e^{(-\mu_{slow} x)}, 1\text{-}e^{(-\mu_{fast} x)}\}$.

In the above example, the parameters of interest that one could vary include the number of jobs that the servers can queue (i.e., B_{slow} and B_{fast}), the service rate of the servers (i.e., μ_{slow} and μ_{fast}), the inter-arrival rates of jobs (i.e., α), and the routing probability, p. All these parameter value variations result in alternate design configurations of the two-server model, and can be evaluated for the measures of interest simultaneously.

3.2 Creating Different Alternative Configurations of the Model

In general, the behavior of a discrete-event system is governed by two components: the state space of the system and the events that cause the state changes. Alternative configurations of a discrete-event system can be created by varying the parameter values that change its state space, its set of possible state transitions, or the event rate of the system. In particular, for a system represented as a GSMP, alternative configurations can be generated as follows:

1) The probability transition function captures the behavior of the parameters that control the state space of the system. By either varying the values of the probability that governs the probability transition function or the conditions that control the probability transition function, one can create discrete-event models that have different state spaces.

2) The input distribution function ψ captures the behavior of the parameters that control the event rate of the system. Changing parameter values of the rate parameters λ of events varies the system behavior of a discrete-event model.

Using a combination of both types of parameters, it is possible to generate a large family of different configurations of the system that can be studied simultaneously.

3.3 Construction of Equivalent GSMP′s for Alternative Configurations

To construct multiple models that can be simulated simultaneously, we construct GSMPs that augment the GSMP of each independent configuration. The goal is to have a common input distribution function ψ for all the models that would allow us to maintain a common enabled event set while simulating all the alternative configurations, thus amortizing the cost of event selection. That necessitates modifications to the probability transition functions of all the configurations to compensate for the existence of a common input distribution function. This section describes the steps necessary to make the changes for each configuration that would enable all the configurations to be evaluated simultaneously.

Consider a discrete-event model represented by a $GSMP = (S, E(s), p(., s, e))$. Let k be the number of parameters we wish to vary. Each parameter k_i is evaluated for n_j combinations, which results in $n = \prod n_j$ alternative configurations of the discrete-event model to be simulated. It is fairly simple to generate the GSMP for each of the alternative configurations, as seen in the previous section. Let each alternative configuration be denoted by $GSMP^v = (S^v, E^v(s), p^v(., s, e))$.

Each new augmented $GSMP'^v$ that meets the condition described in the last paragraph and supports simultaneous simulation, is constructed from $GSMP^v$ as follows.

1. The states S for the equivalent $GSMP'^v$ would be the same as the states S^v of the particular configuration, i.e., $S'^v = S^v$.

2. The new set of actions for $GSMP'^v$ is the union set of the actions of all the configurations ($E'^v = \bigcup_{i=1}^{n} E^i$). Note that an action $e^i \in E^i$ is said to be equivalent to $e^j \in E^j$, i.e., $e^i \equiv e^j$, if it has the same label, ignoring the timing aspect of the action.

3. The set of possible events that are enabled is the union of the possible events of all the configurations, i.e., $E'^v(s) = E$. For each event e that was originally absent for the state s, a probability transition function is added that does not change the state of the system.

4. The probability distribution $F(.; s, e)$ of each event $e \in E$ and state $s \in S$ is modified to correspond to the shortest holding time of the individual configurations. When the event is fired, the process is thinned for the other configurations to reflect its true behavior. The thinning of the Poisson process is done using the state transition probability p described later. In terms of the rate parameter for the input distribution function, λ'_e is now defined as $\lambda'_e = MAX_{i=1}^{m} \lambda^i_e$. The holding time in a state s^v, given that the event e is enabled, is given by $F(.; s^v, e) = F^i(.; s^v, e'')$, provided that $e'' \in E^i(s^v)$, $e \equiv e''$ and $\lambda_{e''} = \lambda'_e$, where i is the index of variant that has the event e''.

5. The transition probability for the new $GSMP'^v$ is the modified transition probability of the individual configuration. For each configuration that does not have the event e defined, i.e., $e \in (E'^v(s) - E^v(s))$, a pseudo transition with probability one is added. Additionally, the transition probability is appropriately thinned to account for the timing aspect associated with event e for the variant v for all $e \in E^v(s)$, i.e., $p'(s', s, e) = \frac{\lambda_e}{\lambda'_e}(p(s', s, e))$.

We aim to show that the behavior of a GSMP and its augmented version are identical. Since the GSMPs we consider have exponential distributed clock times, it suffices to show that the Markov chains of the processes represented by the original GSMP and the augmented GSMP$'$ are equivalent. We do so by showing that the generator matrices Q for both GSMP models are equal. Formally,

Proposition 1 *Let $S = \{S(t); t \geq 0\}$ and $S' = \{S'(t); t \geq 0\}$ be the process representing the original GSMP and augmented GSMP$'$, respectively. Then S is stochastically equivalent to S'.*

Proof: Since we consider GSMPs for which all the clock times are exponentially distributed, their behavior can be represented as CTMCs. By the definition of GSMP, the rate of going from state s_i to state s_j is the product of the probability transition function $p(s_j, s_i, e)$ and λ_e for each event e enabled when the model is in state s_i. Therefore, the generator matrix Q' of the CTMC that represents GSMP$'$ is given by $q'_{ij} = \sum_{e \in E'(s_i)} p'(s_j, s_i, e)\lambda'_e$. Recall that the augmented GSMP was created by adding pseudo transitions to states that do not have particular events defined (Step 5 in the construction of the augmented GSMP). For these events $e \in E'(s_i) - E(s_i)$, $p'(s_j, s_i, e) = 0$, since $i \neq j$. Therefore, the generator matrix entry for Q' is modified as follows: $q'_{ij} = \sum_{e \in E(s_i)} p'(s_j, s_i, e)\lambda'_e$. Note that $E(s_i)$ are the events enabled in the original GSMP model. Furthermore, from step 5 of the construction of the augmented GSMP, we know that $p'(s_j, s_i, e) = \frac{\lambda_e}{\lambda'_e} p(s_j, s_i, e)$. Replacing $p'(s_j, s_i, e)$ in above equation and canceling common terms, we obtain $q'_{ij} = \sum_{e \in E(s_i)} p(s_j, s_i, e)\lambda_e = q_{ij}$, where q_{ij} is the generator matrix of the original GSMP. Therefore $Q = Q'$, which implies S is stochastically equivalent to S'. $\diamondsuit$

We have shown that the original GSMP and the modified GSMP$'$ are stochastically equivalent. Since all the augmented GSMP$'$s have the same distribution functions ψ, we can simulate all alternative configurations of the model simultaneously.

In the case of SCMS using uniformization [10], use of a Poisson process Λ that drives the process S', where $\Lambda = \sum \lambda_e$ for $e \in E'(s)$, leads to the possibility of a large number of pseudo transitions due to events $e' \in (E'(s) - E(s))$ for a given state of the system. We alleviate this problem by considering a non-homogeneous Poisson process ($NHPP$) Λ'_n that drives the process S', where n is the n^{th} transition epoch. The constant uniformization rate for every epoch is determined by the events that are enabled in each of the configurations of the model being evaluated. As argued in [6], that uniformization approach is valid even if one thins a non-homogeneous Poisson process. In effect, in our technique, an NHPP with a piecewise constant epoch is used to uniformize after the firing of each event. To be conservative and prevent incorrect uniformization, care is taken that the adaptive rate is always greater than or equal to the actual possible transition rates of all the enabled events. In this way, the updating of the Λ_n is done as follows after each epoch: $\Lambda_i = \sum \lambda_e$ where $e \in \cup_{i=1}^{n} E(s^i)$.

Note that $E(s^i)$ is the set of events from the original representation of the discrete event model. It is always true that $\cup_{i=1}^{n}(E(s^i)) \subset E'(s^v)$ for any variant v. There is always the possibility that $\cup_{i=1}^{n}(E(s^i)) = E'(s)$, i.e., the system could have all the events of all the variants enabled at all times. That could cause the adaptive uniformization to behave just like uniformization, except that the construction of the adaptive uniformization parameter will always ensure that there is useful computation by at least one of the configurations, i.e., the firing of an event in adaptive uniformization will change the state of at least one of the configurations, which might not be the case with the traditional uniformization technique. Hence, our technique will always guarantee progress in simulation for at least one of the configurations. The next section describes a simulation algorithm that uses this approach.

3.4 Algorithm: Simultaneous Simulation Using Adaptive Uniformization

Continuing with the discussion from the previous section, consider a scenario in which we want to simulate N alternative configurations of a system parameterized by its design parameter values. As we have discussed earlier, we obtain the new configurations from the original model by modifying their probability transition functions $p(s', s, e)$, input distribution functions ψ, or initial states.

The simulation algorithm we developed (See Algorithm 1) has three basic components: an algorithm to generate the next event, an algorithm to update the state of all the configurations simultaneously for the occurred event, and an algorithm to update the enabled event set for the clock-generating algorithm based on the new state of the configurations. This algorithm is repeated until a desired confidence interval width is achieved through execution of multiple batches (in the case of steady state simulation) or replication (in the case of terminating simulation).

To begin with, the enabled event set is empty. The simulation algorithm initially iterates through all the events in the model, adding them to the enabled event set (EES) if they are enabled by any of the configurations. Once the initial enabled event set is built, the simulator executes the basic components in a loop until a terminating condition is satisfied. In each iteration of the loop, the algorithm generates the next event epoch using an exponential random variable that uses the adaptive uniformization rate. Then an event is picked randomly from the EES that is weighted by its firing rate compared to the adaptive uniformization rate. This event is fired to update the state of all the alternative configurations. Only those configurations that enabled the fired event have their state updated. The EES is updated to remove disabled events and add new enabled events that are due to the updating of the states of some or all of the alternative configurations. A new adaptive uniformization rate is recomputed for the updated EES and the iteration is repeated.

Algorithm 1 Simultaneous simulation using adaptive uniformization

```
1:  Let
      $EES$        = $\emptyset$, empty set
      $N$          = number of alternate configurations,
      $v$          = index of the $v^{th}$ configuration,
      $s_0^v$      = initial state of each configuration,
      $D(e)$       = dependency list that maintains the set of
                     enabled events enabled due firing of event $e$,
      $n$          = index to the $n^{th}$ event epoch,
      $\tau_n$     = $n^{th}$ event epoch,
      $e_n$        = event fired in the $n^{th}$ event epoch,
      $u$          = $U(0,1)$, uniform random variable,
      $R_k^v$      = $k^{th}$ reward measure defined on variant $v$,
      $erv$        = exponential random variable with rate 1,
      $\Lambda_n$  = adaptive uniformization rate.
2:  $\forall e \in \bigcup_{v=0}^{N} E(s_0^v)$, $EES = EES + \{e\}$.
3:  $\Lambda_0 = \sum \lambda_{e_j}$ where $e_j \in EES$.
4:  $n = 0$, $\tau_0 = 0$.
5:  repeat
6:    Generate next event
        (a)  $\tau_{n+1} = \tau_n + \frac{erv}{\Lambda_n}$.
        (b)  $P[0] = 0$.
        (c)  $for(j = 1;$ j$\leq$ |EES|$; j++)$ $P[j]$ = $P[j-1]$ + $\frac{\lambda_{ej}}{\Lambda_n}$.
        (d)  $e_n = e^j$ from $e \in EES$
             iff $(P[j-1] \leq u < P[j])$.
7:    Update state
        (a)  $\forall v$ with $e_n \in E(s_n^v)$ enabled, set $s_n^v$ to the next
             state $s'^v_{n+1}$ if $u > p(s'^v_{n+1}, s_n^v, e_n)$.
8:    Update EES
        (a)  $\forall e \in EES$, $EES = EES - \{e\}$,
             if $e \notin \bigcup E(s_{n+1}^v)$.
        (b)  $\forall e' \in D(e_n)$, $e' \in \bigcup E(s_{n+1}^v)$,
             $EES = EES + \{e\}$.
        (c)  $\Lambda_{n+1} = \sum \lambda_j$ where $e_j \in EES$.
9:    $\forall v, \forall k$, compute $R_k^v$.
10:   $n = n + 1$
11: until a defined terminating condition.
```

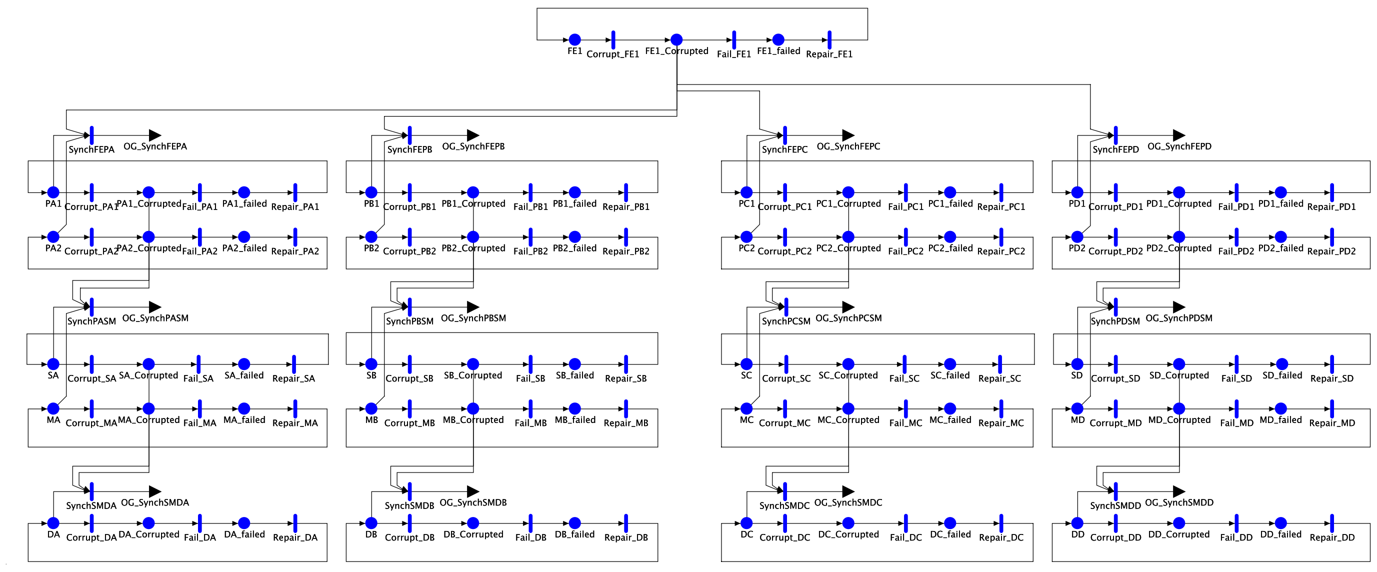

Figure 2. SAN model of the distributed information server

4 Implementation in Möbius

We have implemented the algorithm, describe in Section 3.4, as a C++ module extending the simulation features in the Möbius [2] tool. We implemented our approach by first separating the state of the model and the clock event generation mechanism. We replicate the state of the model for each alternative configuration. The event generations and management for all the configurations are merged to obtain a single set of events of all the configurations, while maintaining the information on timing and enabling conditions of each event in the individual configuration. For each configuration, after each event is fired from the event management module, the event is checked to determine whether it is an actual state transition or a pseudo state transition. The state of the configuration is updated based on the outcome of the classification of the event. To further optimize the algorithm, the simulation keeps track of only those configurations that are active for a particular event, thus eliminating the need to test of all configurations when an event is fired. The computational savings from the amortization of the cost of event management in a simultaneous simulation of all the configurations is quite substantial, as shown in the next section using a model of a distributed information service system.

5 Experimental Evaluation

We analyze our simulation technique by studying the availability of an information service system adapted from [5] and [8]. The model represents different components of an information service and the interaction and propagation of faults across the components. We evaluate the model for varied configurations and show that our simulation algorithm is efficient and scalable, thus providing a good framework for design optimization techniques.

5.1 Model Details

The distributed information service system has a single front-end module that interacts with four processing units. Each processing unit has two processor units, one unit of memory, a switch, and a back-end database. Each of the units can be in any of the following four states: *Working*, *Corrupted*, *Failed*, and *Repaired*. The units cycle through those states, as shown in Figure 2 using a SAN [7] model.

5.2 Fault Model

The model describes how the fault propagates through the various components as each of them is corrupted. Each component can become corrupted internally. While corrupted, some of the components can corrupt other units. Errors are propagated based on the following rules about the effects that a particular subset of units have on other components when in the *Corrupted* state.

The corrupted front end may propagate an error to either of the two working processors in any of the four processing units. The propagation occurs through the common event between the front end and processors. After the failure is propagated, the front end might still remain in the corrupted state and could possibly corrupt the processors on the other *Working* processing units. When both of the processors of a processing unit are in the *Corrupted* state, they may corrupt the *Working* switch or the memory unit. Like the front -end, the processor might remain in the *Corrupted* state until it fails. Both the memory unit and the switch unit in

Table 1. Comparison of traditional and our new simulation techniques for accuracy

component		traditional simulator			simultaneous simulation		
proc 1	proc 2	Availability	Standard error	Computational cost (seconds)	Availability	Standard error	Computational cost (seconds)
5	7	$8.464\times e^{-02}$	$5.600\times e^{-05}$	10.40	$8.461\times e^{-02}$	$5.126\times e^{-05}$	
5	70	$8.999\times e^{-02}$	$4.716\times e^{-05}$	10.40	$8.995\times e^{-02}$	$4.725\times e^{-05}$	
5	700	$9.055\times e^{-02}$	$4.601\times e^{-05}$	10.40	$9.050\times e^{-02}$	$4.327\times e^{-05}$	
5	7000	$9.056\times e^{-02}$	$4.557\times e^{-05}$	10.40	$9.060\times e^{-02}$	$4.589\times e^{-05}$	
50	7	$9.227\times e^{-02}$	$4.214\times e^{-05}$	10.61	$9.229\times e^{-02}$	$3.686\times e^{-05}$	
50	70	$9.829\times e^{-02}$	$2.068\times e^{-05}$	10.21	$9.831\times e^{-02}$	$2.438\times e^{-05}$	
50	700	$9.892\times e^{-02}$	$1.656\times e^{-05}$	10.21	$9.893\times e^{-02}$	$1.965\times e^{-05}$	23.90
50	7000	$9.899\times e^{-02}$	$1.597\times e^{-05}$	10.21	$9.899\times e^{-02}$	$1.702\times e^{-05}$	
500	7	$9.306\times e^{-02}$	$4.017\times e^{-05}$	10.40	$9.309\times e^{-02}$	$2.969\times e^{-05}$	
500	70	$9.917\times e^{-02}$	$1.447\times e^{-05}$	10.21	$9.919\times e^{-02}$	$1.328\times e^{-05}$	
500	700	$9.982\times e^{-02}$	$6.820\times e^{-05}$	10.40	$9.982\times e^{-02}$	$5.320\times e^{-05}$	
500	7000	$9.988\times e^{-02}$	$5.588\times e^{-05}$	10.21	$9.988\times e^{-02}$	$5.306\times e^{-05}$	
5000	7	$9.316\times e^{-02}$	$3.989\times e^{-05}$	10.21	$9.318\times e^{-02}$	$2.795\times e^{-05}$	
5000	70	$9.926\times e^{-02}$	$4.601\times e^{-05}$	10.21	$9.927\times e^{-02}$	$1.189\times e^{-05}$	
5000	700	$9.991\times e^{-02}$	$4.922\times e^{-05}$	10.21	$9.991\times e^{-02}$	$4.777\times e^{-05}$	
5000	7000	$9.997\times e^{-02}$	$2.823\times e^{-05}$	10.21	$9.997\times e^{-02}$	$2.115\times e^{-05}$	
Total Time				164.90			23.90

their *Corrupted* state can corrupt the back-end database unit by propagating their error to it. Both the memory unit and switch unit can remain in the *Corrupted* state independently before they move to the *Failed* state.

The distributed information server is said to be available if the front end is able to communicate with the back-end database. The model parameters used in the experiments can be found in [5]. The availability of the system is measured at the transient time point of 0.1 given that all the components were in *Working* state at time point 0. We use the traditional Möbius simulator to evaluate the configurations to compare the accuracy and scalability with our technique. In order to have accurate comparison between our technique and standard discrete-event simulation, we ran a simulation of each configuration for one million batches. We show that our approach evaluates the measures of interest accurately for all configurations of the model and is scalable to the number of configurations.

Both simulators were run on an AMD Athlon XP 2700+ processor running at 2.2 GHz with 1Gb RAM with Redhat Linux 9.0. The implementation was compiled using a g++ 3.2.2 with optimization level -03. We describe the experiments we conducted in the following sections.

5.3 Correctness and Efficiency

To illustrate the correctness and efficiency of our approach, we varied the individual corruption rate of processor 1 and processor 2 by a multiplicative factor of 10. The corruption rate of other components, such as the memory, the switch, the front-end and the database were fixed. Thus, we obtained sixteen alternative configurations of the system. Table 1 shows the expected instant-of-time availability measures for these configurations. The table compares the traditional serial simulation to our technique. As one can see from the table, the availability measures obtained from our technique and the traditional method fall in each other's confidence intervals. The results were obtained at a 95% confidence level.

5.4 Scalability

In this set of experiments, the corruption rates of the memory, switch, front -end and the database were varied by a multiplicative factor of 0.1 starting from values used in the previous experiment to obtain 64, 256, 1024, and 4096 configurations respectively. Table 2 shows the speed-up obtained by running the all the alternative configurations to obtain their availability. We notice that our technique obtains an average fivefold speed-up compared to traditional simulation, as we simultaneously simulate up to 4096 configurations.

Note that in certain cases the speed-up can be as much as an order of magnitude, but decreases for more than 256 alternative configurations. The decrease in speed-up can be attributed to two factors. First, one should note that the relative length of the trajectory that is required to compute the instant-of-time availability is very short (0–0.1 time units) in the system we have described. Thus, most of the computation time is spent in initializing the simulation of batches rather than executing events. Since a million replications are run, the cost of initializing the simulation was the largest

Table 2. Scalability of traditional simulator vs our technique

Number of alternate configurations	Total Simulation Time Traditional Simulation	Simultaneous Simulation	Speedup
1	10.21	10.59	0.96
4	41.44	13.37	3.10
16	164.90	23.90	6.90
64	665.64	68.46	9.72
256	2648.00	165.91	15.96
1024	10559.00	1617.00	6.52
4096	41978.00	8088.00	5.19

overhead for this particular example. Second, we noted that when the number of configurations increases beyond a certain threshold, the speed-up obtained by the locality of reference for memory access by the processor to update the state of the model or to compute the reward measures is lost. Due to the use of large arrays to represent the state of the system of each configuration, the overhead of updating the state of the system and computing the reward measures decreased the speed-up of the simulation. Thus, the speed up is comparatively moderate for the example model used to describe our technique.

Coming back to the issue of pseudo transitions, one should note that, unlike the SCMS, our algorithm insures that at any given point in time at least one configuration of the discrete-event model will be performing useful work. It is always possible to have a family of models in which one or more alternative configurations have events that are fired at very different time scales that could potentially cause other configurations to have significant numbers of pseudo event transitions. However, our technique will make sure that we are progressing in the simulation for at least those highly skewed configurations.

6 Conclusions

In this paper, we discussed a new approach to simulate alternative configurations of dependability models simultaneously by combining adaptive uniformization and the SCMS technique. We showed that a significant speed-up can be obtained by this new approach due to the amortization of the cost of event set management, and that the approach achieves considerable speed-up due to correlation among the trajectories for most of the alternative configurations. We expect that our technique opens up new research issues and ideas in optimization of simulations, sensitivity analysis, and parameter optimization of systems.

The utility of simulation of different models and statistical analysis of the outputs lies in comparing the alternatives before the actual implementation or deployment of the system. Even though there is a coupling between the state update mechanism and the clock-event generation mechanism, the structural and behavior at similarity that configurations display provides an opportunity to reduce the computational cost of updating and maintaining the enabled event set. The state-updating mechanisms of the configurations are independent, allowing comparison of the configurations, and enabling the possibility of making decisions at the run time of the simulations. We thus have provided a fast, efficient way to evaluate a large number of alternative design choices.

7 Acknowledgment

This research was supported in part by Pioneer Hi-Bred International, Inc. and Hewlett-Packard. We appreciate their support. We would also like to thank Ms. Jenny Applequist for her editorial assistance.

References

[1] C.-H. Chen and Y.-C. Ho. An approximation approach of the standard-clock method for general discrete-event simulation. *Control Systems Technology*, 3(4):309–318, 1995.

[2] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster. The Möbius modeling tool. In *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pages 241–250, September 11–14 2001.

[3] P. W. Glynn. On the role of generalized semi-Markov processes in simulation output analysis. In *WSC '83: Proceedings of the 15th conference on Winter simulation*, pages 39–44, Piscataway, NJ, USA, 1983. IEEE Press.

[4] P. Heidelberger and D. M. Nicol. Conservative parallel simulation of CTMC using uniformization. *IEEE Trans. Parallel Distrib. Syst.*, 4(8):906–921, 1993.

[5] V. V. Lam, P. Buchholz, and W. H. Sanders. A structured path-based approach for computing transient rewards of large CTMCs. *Proceedings. First International Conference on the Quantitative Evaluation of Systems (QEST).*, pages 136–145, 2004.

[6] P. A. W. Lewis and G. S. Shedler. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Research Logistics Quarterly*, 26(3):403–413, 1979.

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

[8] R. R. Muntz and J. Lui. Computing bounds on steady-state availability of repairable computer systems. *Journal of the ACM*, 41(4):676–707, 1994.

[9] D. M. Nicol and P. Heidelberger. Parallel simulation of Markovian queueing networks using adaptive uniformization. *SIGMETRICS*, 21(1):135–145, 1993.

[10] P. Vakili. Massively parallel and distributed simulation of a class of discrete event systems: A different perspective. *ACM Trans. Model. Comput. Simul.*, 2(3):214–238, 1992.