Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

A COMPONENT-LEVEL PATH-BASED SIMULATION APPROACH FOR EFFICIENT ANALYSIS OF LARGE MARKOV MODELS

Vinh V. Lam

Coordinated Science Laboratory
1308 W. Main Street
University of Illinois
Urbana, IL 61801, U.S.A.

Peter Buchholz

Informatik IV
Universität Dortmund
D-44221 Dortmund, GERMANY

William H. Sanders

Coordinated Science Laboratory
1308 W. Main Street
University of Illinois
Urbana, IL 61801, U.S.A.

ABSTRACT

Markov models are used in many industrial applications, but, for very large models, simulation is often currently the only viable evaluation technique. However, simulation techniques that are based on evaluating trajectories at the level of individual states and transitions can be inefficient because they have to keep track of many details. Moreover, since they use statistical methods, estimating solutions at higher confidence intervals requires the evaluation of an increasingly large number of trajectories which often leads to poor performance. On the other hand, analytical path-based techniques can be used for computing guaranteed bounds on the true solutions, but they can have poor performance because they must evaluate many paths to obtain reasonable bounds. In this paper, we present a path-based simulation approach for evaluating models at the component, rather than individual state/transition, level. At this level of abstraction, the approach can compute more accurate solutions than traditional discrete-event simulation techniques can in a given amount of time. In addition to presenting the approach, we compare its performance and effectiveness against a path-based analytic technique.

1 INTRODUCTION AND MOTIVATION

Model-based evaluation is an effective means to gain insight into the behavior of many computing systems. It can be used during many stages of system development, from conceptual design to prototype evaluation to actual-system operation and maintenance. At all these stages, models of performance, availability, and reliability may be used to predict the operating characteristics of systems. Frequently, models can be expressed as continuous-time Markov chains (CTMCs). However, it is generally necessary to model a system at a fairly high level of detail in order to obtain results efficiently. As the amount of detail and the level of complexity in a model increase, computing solutions by numerical methods becomes increasingly difficult due to the well-known state-space explosion problem.

Simulation is an alternate solution approach that is usually used for analyzing very large models. It has no explicit state-space storage constraint because it analyzes a model directly without the use of the underlying state space. It does this by executing a representative set of trajectories over which the model may evolve over time. The trajectories are chosen based on their relative likelihood of occurrence in the model, and the solution of the model is estimated using a statistical analysis of the chosen trajectories.

In many simulation approaches, the trajectories are defined at the level of individual states and transitions. At this low level of representation, it is necessary to keep track of many state variables of a model to determine which transitions are enabled and which events may occur next. Furthermore, even though multiple trajectories can be evaluated in parallel on a multiprocessor machine, each one must still be evaluated serially on a uniprocessor machine. That constraint limits the number of trajectories that can be evaluated in a given amount of time. When a high confidence level is required in a solution, a large number of trajectories have to be evaluated. Thus, solving a model for solutions with high confidence levels is a time-consuming task.

In this paper, we present a new approach for simulating continuous-time discrete-state Markov models. The approach extends existing simulation and path-based analytical approaches by simulating models at the component level. Instead of representing the states of a model explicitly, as is traditionally done in many other simulation methods, our approach represents them with a set of probability vectors corresponding to the probabilities of being in particular states within each component of a model. By traversing "trajectories" at this higher level of abstraction, the approach improves the efficiency and quality of a solution relative to a solution computed using traditional discrete-event simulation or analytical path-based methods.

The presentation of our paper proceeds as follows. In Section 2, we briefly explain how to represent a model at the component level. Then, in Section 3, we present a path-based simulation approach for executing a trajectory at the component level. Next, in Section 4, we illustrate how the approach can be used for evaluating the availability and reliability of a distributed information service system. Additionally, we present experimental results to compare the performance and effectiveness of the approach against the analytical path-based approach presented in (Lam, Buchholz, and Sanders 2004). Finally, we conclude with a summary and a discussion of future work in Section 5.

2 COMPONENT-LEVEL ANALYSIS OF MODELS

Many large models, like their physical system counterparts, are constructed from smaller logical submodels or components. A large model is composed from the submodels either by means of *state sharing*, whereby the submodels coordinate their transitions through shared states (e.g., Derisavi, Kemper, and Sanders 2003), or by means of *action synchronization*, whereby they coordinate through shared events. For our approach, we consider the latter.

Suppose a given model has state space $\mathcal{RS} = \{0, \ldots, n-1\}$ (such that $|\mathcal{RS}| = n$) and is decomposable into J components, which are numbered from 1 to J. In the sequel, we use parenthesized superscripts and subscripts, $i,j \in \{1, \ldots, J\}$, to denote particular components, unless the context is clear (in which case, we do not use parentheses). Let component i have state space $\mathcal{RS}_i = \{0, \ldots, n_i - 1\}$. Then the state space of the entire model can be composed from the state spaces of the components such that the relation $\mathcal{RS} \subseteq \times_{i=1}^{J} \mathcal{RS}_i$ holds true.

In many simulation methods, a state of a component is represented by a vector of its state variables; the state of the model is represented by a vector of component states. The next events that *may* occur are determined by observing the values of the state variables of the current state. When all conditions are satisfied to enable an event, it is scheduled and put on a list of enabled events. Then, one event from the list is chosen to occur. After the occurrence of an event, the affected state variables are updated, and a transition to the next state takes place. Afterward, events previously scheduled on the list may become disabled and are removed from the list.

The costs of determining the next events, managing the event lists, and updating states and state variables can (when all state holding times are exponentially distributed) be eliminated or lessened by keeping track of the probabilities that a component will occupy individual states instead of keeping track of the states that the component may occupy. Essentially, instead of using a vector of state variables for each component, a vector of state probabilities is used to keep track of the probabilities that the component will occupy the states in its state space. The *state probability vector* of the whole model can be represented in a similar manner and can be computed from the state probability vectors of the components. If this representation is used, it is not necessary to determine which next events are enabled; it is not necessary to manage an event list; and updating states involves only a (more efficient) vector-matrix operation.

More formally, define $\mathbf{p}_0^{(i)}$ to be the initial state probability (row) vector of component i and $\mathbf{p}^{(i)}$ to be the vector after some number of events that affect i have occurred. The events that affect i can be classified as either *local* to i or *global*. The local events are those occurring within component i. The global events are those associated with the synchronization activities that affect multiple components in the whole model. Each synchronization activity can be further classified as either a *true synchronization event*, in which *all* involved components take a synchronized transition, or a *disabled synchronization event*, in which *some* involved components cannot take the synchronized transition because their present states disallow the transition.

The effects of these events on the components can be described by the *probability transition matrices* (Stewart 1994). Each matrix describes how a component transitions among its states and the probabilities that those transitions will be caused by the event. Let us define the following matrices corresponding to these types of events for component i: $\mathbf{P}_l^{(i)}$ is the local transition matrix, $\mathbf{E}_t^{(i)}$ is the synchronized transition matrix for synchronization event t, and $\overline{\mathbf{E}}_t^{(i)}$ is the disabled synchronized transition matrix for synchronization event t. $\mathbf{D}_t^{(i)} = \text{diag}(\mathbf{E}_t^{(i)}\mathbf{e}^T)$, where $\mathbf{e}$ is an n_i-dimensional vector of ones, is simply the diagonal matrix of $\mathbf{E}_t^{(i)}$. We summarize these matrices notationally as follows:

$$\mathbf{\Phi}^{(i)}_{\boldsymbol{\pi}(k)} = \begin{cases} \mathbf{P}_l^{(i)} & \text{if } \boldsymbol{\pi}(k) = l_i \\ \mathbf{E}_t^{(i)} & \text{if } \boldsymbol{\pi}(k) = t \text{ for } t \in \mathcal{T}_S \\ \mathbf{D}_t^{(i)} & \text{if } \boldsymbol{\pi}(k) = \bar{t}_j \text{ for } i < j \text{ and } t \in \mathcal{T}_S \\ \overline{\mathbf{E}}_t^{(i)} & \text{if } \boldsymbol{\pi}(k) = \bar{t}_i \text{ for } t \in \mathcal{T}_S \\ \mathbf{I}_{n_i} & \text{otherwise} \end{cases} .$$

By using these vectors and matrices, the effect of a series of events on a particular component can be computed simply by a series of products of the corresponding component vector and matrices.

It is possible to compute various measures of interest from a model by using the Markov reward model (Howard 1971) to specify how reward is computed when the model visits some states. For computing rate reward, define the column vector $\mathbf{r}^{(i)}$ to be the reward vector for component i. $\mathbf{r}^{(i)}$ is an n_i-dimensional vector whose entries specify the reward associated with the corresponding states. Further, let $\boldsymbol{\pi}^{(i)}$ be a sequence of events that affect i. $\boldsymbol{\pi}^{(i)}$ can also be considered as a *set of paths* over which i evolves over time, and it is an abstraction of the state-level trajectory. Then the product of $\mathbf{p}_0^{(i)}$ and the matrices corresponding to the events in $\boldsymbol{\pi}^{(i)}$ yields the state probability vector, $\mathbf{p}^{(i)}[\boldsymbol{\pi}^{(i)}]$, after the occurrence of $\boldsymbol{\pi}^{(i)}$. The reward from component i over this path is computed as

$$R^{(i)}(\boldsymbol{\pi}^{(i)}) = \mathbf{p}^{(i)}[\boldsymbol{\pi}^{(i)}] \cdot \mathbf{r}^{(i)}. \tag{1}$$

At a global model level, let $\boldsymbol{\pi}$ be a path of events over which the model evolves. Observe that $\boldsymbol{\pi}$ is a splicing of $\boldsymbol{\pi}^{(i)}$ for $i = 1 \dots J$. For a measure specified by a tensor product (Stewart 1994) of the component reward vectors, the reward over $\boldsymbol{\pi}$ is computed as

$$R(\boldsymbol{\pi}) = \prod_{k=1}^{J} R^{(i)}(\boldsymbol{\pi}^{(i)}). \tag{2}$$

The probability of the path $\boldsymbol{\pi}$ is given by

$$Prob(\boldsymbol{\pi}) = \prod_{k=1}^{|\boldsymbol{\pi}|} \frac{\lambda_{\boldsymbol{\pi}(k)}}{\Lambda}, \tag{3}$$

where $\lambda_{\boldsymbol{\pi}(k)}$ is the rate of occurrence of the event corresponding to $\boldsymbol{\pi}(k)$ and Λ is the sum of the rates of enabled transitions in the model. Let $\beta(\Lambda s, k)$ denote the Poisson density describing the probability of having k occurrences of some events at rate Λs. Then the expected reward of the model at time s can be computed by

$$E[R_s] = \sum_{k=0}^{\infty} \beta(\Lambda s, k) \sum_{\boldsymbol{\pi} \in \mathcal{P}^k} Prob(\boldsymbol{\pi}) R(\boldsymbol{\pi}). \tag{4}$$

Equation (4) states that the expected instantaneous reward is just the weighted sum of the path rewards over all possible paths, where the weights are the probabilities of the paths.

Note that computation of the exact solution requires evaluation of infinitely many paths as shown in Equation (4). Guaranteed lower and upper bounds on the exact solution can be computed by evaluating a finite number of paths; the more paths that are evaluated, the tighter the bounds will be (as shown in Lam, Buchholz, and Sanders 2004). In general, to get useful bounds for practical models, a large number of paths must be evaluated. In the next section, we present a new path-based simulation approach that evaluates many fewer paths than the analytical path-based bounding approach of Lam, Buchholz, and Sanders (2004), yet it can efficiently compute solutions with high confidence. The approach is not only more effective than existing ones, it is also more efficient.

3 THE PATH-BASED SIMULATION APPROACH

This section presents the component-level path-based simulation approach. It is more efficient than traditional simulation methods because it is partially based on numerical analysis and it does not manage any event list. It is also often faster than the analytical path-based approach because it can estimate a solution by evaluating fewer paths. After presenting the approach, we use it to evaluate one model for the availability measure and another model for the reliability measure. We then compare the performance and effectiveness of the approach in computing these solutions against the analytical path-based bounding approach.

The basic idea behind the path-based simulation approach is that it is possible to estimate the solution of Equation (4) by sampling a set of random paths. Formally, let $\mathcal{SP}$ be a *multiset* of (sets of) paths from which samples will be drawn. Note that $\mathcal{SP}$ is a (small but representative) subset of the set of all possible paths, and the same path may be drawn multiple times from it. A path $\boldsymbol{\pi} \in \mathcal{SP}$ must be chosen with probability

$$\beta(\Lambda s, |\boldsymbol{\pi}|) \cdot Prob(\boldsymbol{\pi}). \tag{5}$$

Then the unbiased estimators for the mean and variance can be computed by

$$\begin{aligned} \hat{R}(\mathcal{SP}) &= \frac{1}{|\mathcal{SP}|} \sum_{\boldsymbol{\pi} \in \mathcal{SP}} R(\boldsymbol{\pi}) \\ \hat{S}^2(\mathcal{SP}) &= \frac{1}{|\mathcal{SP}| - 1} \sum_{\boldsymbol{\pi} \in \mathcal{SP}} \left(R(\boldsymbol{\pi}) - \hat{R}(\mathcal{SP}) \right)^2. \end{aligned} \tag{6}$$

In order to keep the estimators unbiased, it is necessary to exercise caution in choosing random paths according to their probabilities of occurrence as given in Equation (5). It is possible to choose a path randomly according to the probability by first drawing a random variate from the Poisson distribution that has rate Λs. For an efficient realization, we use the work of Fox and Glynn (1988) in our implementation. A Poisson random variate is used to choose the length of each sampled (abstract) path. Next, the events making up the path must be chosen according to their probabilities.

Note that they cannot be statically determined as in Equation (3), but instead they must be computed dynamically as the model evolves according to the path $\boldsymbol{\pi}$. That is, the probabilities of the events to be chosen to form the path $\boldsymbol{\pi}$ are computed according to the states that the model occupies. These probabilities are computed by

$$ProbT(\boldsymbol{\pi}(k)) = \frac{\lambda_{\boldsymbol{\pi}(k)}}{\Lambda} \prod_{j=1}^{J} \mathbf{p}^{(j)} \cdot \boldsymbol{\Phi}^{(j)}_{\boldsymbol{\pi}(k)} \cdot \mathbf{e}^T_{n_j}, \qquad (7)$$

where $\boldsymbol{\pi}(k) \in \{l_j, t, \bar{t}_j\}$ for $j = 1 \ldots J$ and $t \in \mathcal{T}_S$. Equation (7) is simple to compute and can be optimized so as to require only a scalar product. Thus, $ProbT(\boldsymbol{\pi}(k))$ forms a distribution over the set of possible next events.

Using Equation (6), an approximate $100(1-\alpha)\%$ $(0 < \alpha < 1)$ confidence interval for the mean is computed by

$$\hat{R}(\mathcal{SP}) \pm t_{|\mathcal{SP}|-1,1-\alpha/2}\sqrt{\frac{\hat{S}^2(\mathcal{SP})}{|\mathcal{SP}|}}, \qquad (8)$$

where $t_{|\mathcal{SP}|-1,1-\alpha/2}$ is the *Student t-distribution* having $|\mathcal{SP}| - 1$ degrees of freedom.

4 EVALUATION OF THE APPROACH

We evaluate our approach by studying its performance in analyzing a model of a distributed information service system adapted from the model in (Muntz and Lui 1994). The example model describes the propagation of faults across the components of the system. We augment the original model with synchronized transitions among the components to describe how faults are propagated through the system. In addition, we increase the number of front-end modules in order to model the occurrence of a fault only when a majority of the modules are corrupted. We also model double redundancy in the processing units by adding an additional module for every module in the original processing units. After describing the model, we discuss the performance and accuracy of the approach.

4.1 Model Description

The information service system consists of six front-end modules that interact with four processing units. Each processing unit consists of redundant components, including two processors, two switches, two memory units, and two databases. Each of the components has its own repair facility. The components all go through the cycle of *Working*, *Corrupted*, *Failed*, and *Repaired*. For the reliability model, the repair facility is excluded. The *stochastic activity network* (SAN) model of the system is shown in Figure 1. Many of the model parameter values are given in (Lam, Buchholz, and Sanders 2004), except for the values of the redundant components because there are too many to list all of them.

Fault propagation in the system is modeled as follows:

- When a majority of the six front-end modules are corrupted, the front-end is considered faulty, and it may subsequently propagate the error to any of the four processing units in which there are two working processors. Propagation occurs via the synchronized activities between the front-end and the processors in the processing units. The front-end or any of the processors may disable the synchronized activities. After propagating the error to a processing unit, the front-end may remain in the faulty state and continue to propagate errors to other working processing units until the majority fails or are repaired or there are no more working processing units to propagate the error.
- When both processors in a processing unit are corrupted, they may propagate the error to their working switches via a synchronized activity. Any of the involved components may disable the synchronized activity. After the error propagation, the processors remain in the corrupted state until they fail.
- When both switches of a processing unit are corrupted, they may propagate their errors to the working memory units via a synchronized activity. Any of these components may disable the activity. After propagating the error, the switches remain in the corrupted state until they fail.
- When both memory units of a processing unit are corrupted, they may propagate their errors to the working databases via a synchronized activity. Any of these components may disable the activity. After propagating the error, the memory units remain in the corrupted state until they fail.

The model has five components: the front-end and 4 processing units. Each unit contains two processors, a switch, a memory unit, and a database. We analyze the model to compute the system reliability and availability. We compute the reliability measure at transient time point 1.0 and the availability measure at transient time point 0.1 when all components in the model are in the working state.

4.2 Experimental Results

We conducted all of our experiments on a workstation that had the AMD Athlon XP 2700+ processor running at 2.17 GHz with 512 MB of RAM. The operating system was Red Hat Linux 9.0 with mounted file systems. We compiled our implementation using the compiler g++ 3.3 with optimization flag -O3 only.

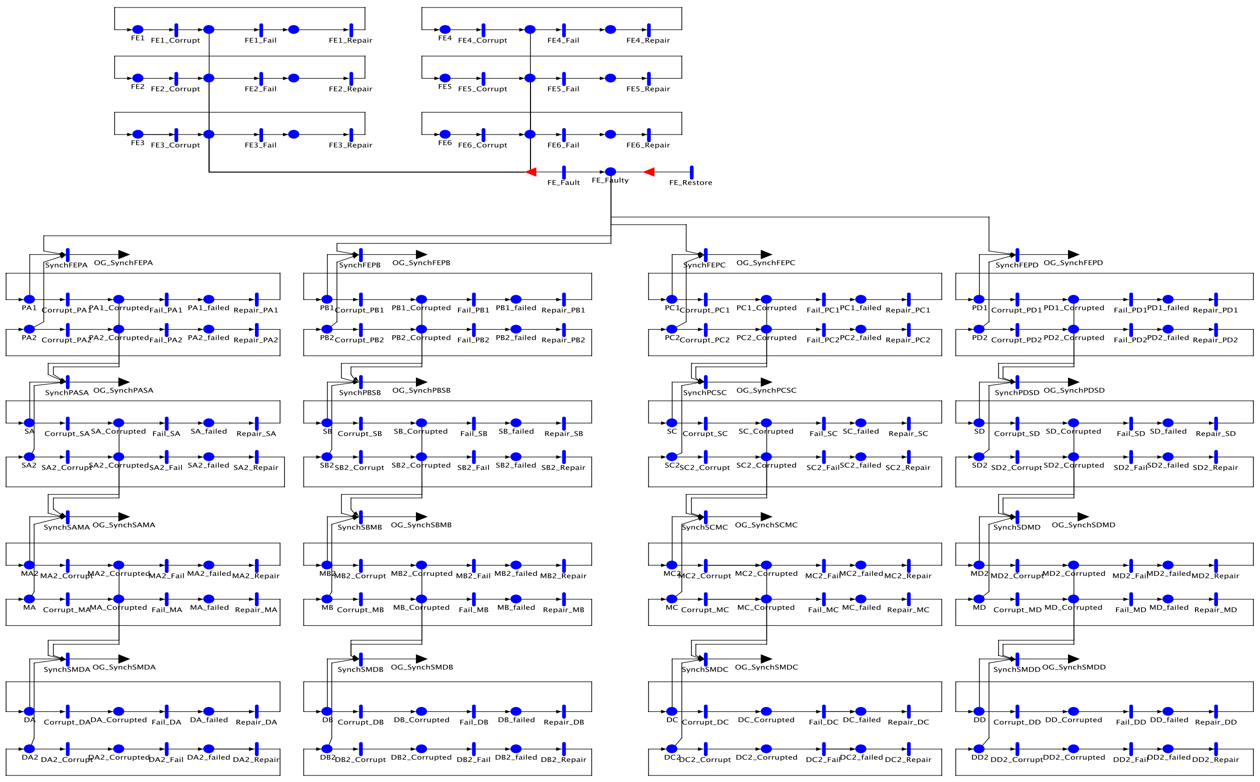


Figure 1: SAN Model for Evaluating the Availability of a Distributed Information Service System

In order to evaluate the quality of the simulation results, we initially analyzed the model for the guaranteed lower and upper bounds on the exact solutions using the approach in (Lam, Buchholz, and Sanders 2004). For the point availability measure, the bounds are $[9.786854 \times 10^{-1},\ 9.985875 \times 10^{-1}]$ and take 6135.23 seconds to compute. For the reliability measure, the bounds are $[9.836687 \times 10^{-1},\ 9.852736 \times 10^{-1}]$ and take 487.09 seconds to compute.

The path-based simulation results are $9.9839562118 \times 10^{-1} \pm 5.3630001139 \times 10^{-5}$ for the point availability (computed in 12.738 seconds) and $9.8353716323 \times 10^{-1} \pm 9.9889749668 \times 10^{-4}$ for reliability (computed in 10.960 seconds).

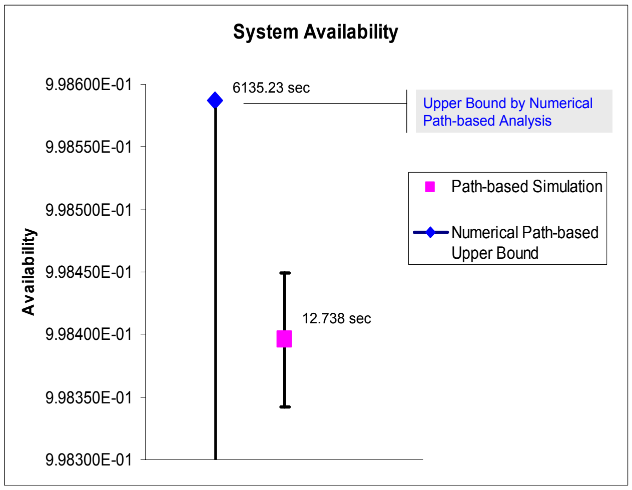
The simulation results for point availability were obtained at a 95% confidence level and $6.0E-5$ absolute confidence interval width. The results for reliability were obtained at a 95% confidence level and $1.0E-3$ absolute confidence interval width. For all simulation experiments, our simulator terminated only when the computed confidence intervals fell below these specified ones.

Figures 2(a) and (b) present the above results in a graphical format for easier comparison of the performance and quality of the solutions obtained using the analytical path-based approach and the path-based simulation approach. In the figures, the bars are not lined up vertically so that the error bars can be distinguished clearly. The vertical axes show the measures that were computed, and the exact solution lies somewhere between the upper and lower bounds computed by the analytical path-based approach. The number next to each bar indicates the time, in seconds, taken by the corresponding approach.

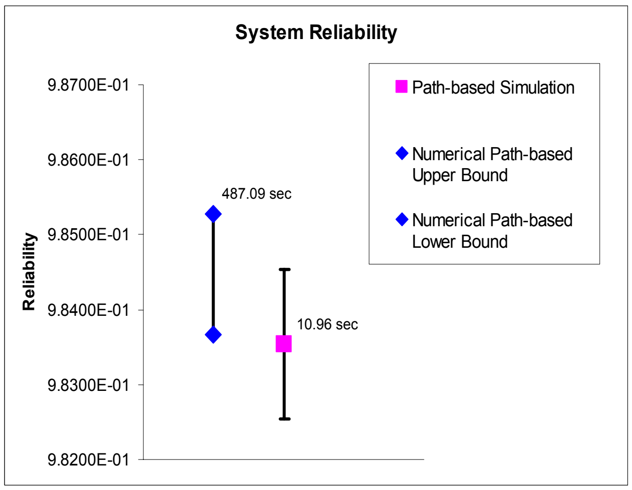
As shown by the intersection of the error bars for the path-based simulation approach and the bounding bars, the path-based simulator is capable of estimating the true solutions quite accurately. In Figure 2(a), note that the analytical approach took more than 6100 seconds to compute the bounds, and yet they are not tight. Thus, given the quality of the estimated solutions, the path-based simulator takes much less time than the analytical approach for both models.

5 CONCLUSION AND FUTURE WORK

In this paper, we presented a novel path-based simulation approach for analyzing very large Markov models. The approach raises the abstraction level of trajectory analysis to the higher level of component analysis. In doing so, we gain two benefits: first, sets of trajectories are simultaneously traversed to improve the quality of the solution for a given number of traversals, and second, the approach works more efficiently by not having to manage event lists. As shown in Figure 2, the solutions estimated by the path-based simulation approach are quite accurate as compared with the guaranteed bounds on the true solutions. In addition, the simulation approach is much faster than the analytical approach by several orders of magnitude.



(a) Availability



(b) Reliability

Figure 2: Comparison of the Performance and Quality of the Solutions Obtained Using the Analytical Path-Based Approach and the Path-Based Simulation Approach

Although the results shown in this paper for the path-based simulation approach look promising, there is much work left to be done to further improve the approach. Currently, a sample estimate is computed from an abstract path of a specific length. Since each abstract path is computed from intermediate subpaths, we can obtain a smaller variance in the estimate by utilizing the intermediate results from the subpaths. Moreover, at the component level, equivalent paths can be identified and exploited easily to reduce further the number of paths that have to be computed. Finally, evaluated paths can be memorized so that they do not have to be re-evaluated (i.e., sampling without replacement). This capability will help the approach to explore a larger model space and will improve the convergence toward the specified error tolerance more quickly. In all, these improvements will help to improve further the quality of the estimated solution and the performance of the approach.

ACKNOWLEDGMENTS

This material is based upon work supported by Pioneer Hi-Bred and by the National Science Foundation under Grant No. 0086096. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. The research of Peter Buchholz has been partially supported by the Deutsche Forschungsgemeinschaft as part of the Collaborative Research Center SFB 358. We would also like to thank Ms. Jenny Applequist for her editorial assistance.

REFERENCES

Derisavi, S., P. Kemper, and W. H. Sanders. 2003. Symbolic state-space exploration and numerical analysis of state-sharing composed models. In *Proceedings of NSMC '03: The Fourth International Conference on the Numerical Solution of Markov Chains*, 167–189.

Fox, B. L., and P. W. Glynn. 1988, April. Computing Poisson probabilities. *Communications of the ACM* 31 (4): 440–445.

Howard, R. A. 1971. *Dynamic probabilistic systems, vol. ii: Semi-markov and decision processes*. John Wiley & Sons, Inc.

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

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

Stewart, W. J. 1994. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press.

AUTHOR BIOGRAPHIES

VINH V. LAM is a Ph.D. student in the Department of Computer Science at the University of Illinois, Urbana-Champaign. He received his B.S. (1998) and M.S. (2000) in computer science from the University of Illinois. His research interests include simulation and modeling, stochastic analysis, and verification. His e-mail address is <lam@uiuc.edu> and his Web address is <http://www.perform.csl.uiuc.edu>.

PETER BUCHHOLZ holds a diploma degree in Computer Science (Dipl.-Inform., 1987), a doctoral degree (Dr.rer.nat., 1991) and a habilitation degree (1996), all from the Uni-

versity of Dortmund, where he has been with the Computer Science Faculty in several positions until 1999. In April 1999 he joined the Department of Computer Science at Dresden University of Technology, where he is an Associate Professor for modeling and simulation. His research interests include techniques for performance and functional analysis of discrete event dynamic systems, numerical analysis techniques for Markov chains and the design and development of software tools for the qualitative and quantitative analysis of complex systems. In the mentioned areas he has published more than 60 papers in refereed journal or conference proceedings and has served on various program committees of international conferences. His e-mail address is `<peter.buchholz@cs.uni-dortmund.de>` and his Web address is `<http://ls4-www.cs.uni-dortmund.de/QM/MA/pb/pie.html>`.

WILLIAM H. SANDERS is a Donald Biggar Willett Professor in the Department of Electrical and Computer Engineering and the Coordinated Science Laboratory at the University of Illinois. He is the Director of the Information Trust Institute. He is a Fellow of the IEEE and the ACM. He serves as the Vice-Chair of IFIP Working Group 10.4 on Dependable Computing. In addition, he serves on the editorial boards of IEEE Transactions on Reliability and Performance Evaluation, and is the Area Editor for Simulation and Modeling of Computer Systems for the ACM Transactions on Modeling and Computer Simulation. Dr. Sanders's research interests include performance/dependability evaluation, dependable computing, and reliable distributed systems. He has published more than 150 technical papers in these areas. He is a co-developer of three tools for model-based evaluation: METASAN, UltraSAN, and Möbius. Möbius and UltraSAN have been distributed widely to industry and academia; more than 300 licenses for the tools have been issued. His e-mail address is `<whs@uiuc.edu>` and his Web address is `<http://www.perform.csl.uiuc.edu>`.