Simulation versus Analytic-Numeric Methods: Illustrative Examples

B. Tuffin
IRISA/INRIA
Campus Universitaire de Beaulieu
35042 Rennes Cedex, France
btuffin@irisa.fr

P. K. Choudhary
Dept. of Electrical and Computer Engg.
Duke University
Durham, NC 27708-0291, U.S.A.
pchoudhary@gmail.com

C. Hirel
Dept. of Electrical and Computer Engg.
Duke University
Durham, NC 27708-0291, U.S.A.
chirel@ee.duke.edu

K. S. Trivedi
Dept. of Electrical and Computer Engg.
Duke University
Durham, NC 27708-0291, U.S.A.
kst@ee.duke.edu

ABSTRACT

Performance along with dependability analysis is a tremendous challenge in the design or improvement of modern complex systems. Two different classes of solution methods are generally used: analytic-numeric methods and simulation methods. As most of the literature explains, the choice between them depends more on the analyst's background than on the system itself. In this paper, we illustrate the advantages and drawbacks of each method on real problems and compare the results. Finally we conclude the paper providing some hints to choose a solution method depending on the model. We use SPNP, a Petri net analysis package, and CSIM 19, as simulation package to model and evaluate systems.

Keywords

Analytic-numeric methods, Performance evaluation, Simulation.

1. INTRODUCTION

Performance and dependability evaluation of modern systems becomes a challenging problem due to the complexity involved. Several solution techniques are available in the literature. One of the most commonly used techniques is the analytic one which produces accurate results. Unfortunately, it becomes inapplicable quickly, due to the size and complexity of models or due to non-Markovian nature of the problem involved. In such cases, approximation methods are applied. Even these approximation methods may become inefficient in most cases, and then simulation becomes inevitable.

In the current literature, authors have either used analytic-numeric methods or simulation, but making this choice is often arbitrary. There is a need for careful comparison of these two methods. In fact, depending on model parameter settings, it may be useful to switch between the two. We illustrate the advantages and drawbacks of these methods in order to help a modeler decide which one might be the most efficient for his/her model. Generally people consider simulation when the assumptions made by analytical models are not appropriate. In this paper, we show that even when all assumptions are appropriate for an analytic numeric solution, storage requirements preclude such a solution and hence one may have to resort to simulation. This has been pointed out earlier in the literature, but never been explicitly illustrated. This paper discusses the relative merits of analytic-numeric methods and simulation when a predefined (time) batch size is fixed.

We consider here two real life examples: a client/server system and a cable modem termination system (CMTS) for the purpose of illustration. Client/server system is a system where a server station receives requests from its client stations, processes the requests and replies to the requesting stations ([16]). We compare the solution methods while varying the number of stations connected to the server. We restrict ourselves to a single model, with a note that a study on other models has led to similar conclusions. In a CMTS system, we calculate the capacity oriented availability(COA) for the system. The CMTS system we consider here consists of multiple primary CMTS nodes and single backup secondary CMTS node. For more details see ([18]).

The analysis of the client-sever example is made using SPNP (stochastic Petri net package) ([10, 15]). For analysis of CMTS system we use SHARPE ([14]) and CSIM 19 ([1]). The systems are modeled by stochastic reward nets (SRNs) ([20]), an extension of generalized stochastic Petri nets ([3]). SPNP includes analytic-numeric methods as well as discrete event simulation ([11, 15, 25]). SHARPE is a analytic-numeric solver which can solve both state-space (meaning models for which the state space has to be generated) and non-state space models including Markov chains and stochastic Petri nets. CSIM 19 is a process-oriented discrete event simulator implemented in C/C++ as a library of routines which perform necessary operations.

The layout of the paper is as follows. Section 2 deals with the

Valuetools '07, October 23-25, 2007, Nantes, France
Copyright 2007 ICST 978-963-9799-00-4.

analysis of the client/server system. The paradigm used is SRNs, for which solution methods are presented in subsection 2.1 (the description of the paradigm is left to appendix A.1, along with the description of the software SPNP, CSIM and SHARPE). The description of the system as a SRN is presented in subsection 2.2, and simulation and analytic-numeric methods are compared for the transient behavior in subsection 2.3 and for steady-state in subsection 2.4. In Section 3 we describe the CMTS system. Again, the full description of the paradigm and software used is left to Appendix A. The transient and steady state result is described in subsections 3.3 and 3.4. Finally, we discuss results and conclude in Section 4.

2. CLIENT/SERVER EXAMPLE

We consider a client/server system where a server station receives requests from its client stations, processes the requests and replies back to the client stations. This is a common feature in distributed computing. The analysis of such systems becomes difficult by various kinds of dependencies in the system ([16]). Generally Markov chain is able to capture the dependencies but a hand construction of such a Markov chain is often tedious and infeasible. For this reason, the use of SRNs as model of representation may be helpful. Before going into further details with the model description, we briefly review the solution methods associated with SRNs. The complete description of SRNs is left to appendix.

2.1 Brief introduction to solution methods for stochastic reward nets (SRNs)

Petri nets are formal graph models particularly well suited for representing the flow of information and control in systems with concurrency and synchronization characteristics ([22]). We will limit ourselves here to discrete Markovian SPNs.

The first class of solution methods is the analytic-numeric one. To apply these methods, several restrictions must be applied on the previous model. The first major limitation is that the distributions of transition firing times must generally be exponential or immediate. There exist less general restrictions (see for instance ([7, 8]) where no more than one non-exponential distribution is enabled in any marking, leading to a Markov regenerative process), but they are still restrictive. One can argue that general distributions can be approximated by phase type distributions ([21]). Nevertheless this can also increase the size of the state space. The second assumption made is that the resampling policy when a transition is disabled by the firing of another transition and becomes enabled again later on is preemptive repeat different (this has been relaxed by some authors ([5])). Given these assumptions, a Markov chain is constructed either directly or via the *reachability graph* from a Petri net model. Several matrix analysis methods may then be used to solve the problem, such as, for instance, steady-state SOR (successive over relaxation), steady state Gauss-Seidel, steady-state power method ([9, 23, 24]) or transient solution using uniformization ([19]).

Simulation methods ([13]), on the other hand can be used when the above assumptions are not satisfactory for model solution, when the storage requirements exceeds the memory capacity or when the computation time for analytic-numeric technique is very long. Since generation of reachability graph is not needed in simulation, this solution method requires less memory. Also the computation time can be reduced by decreasing the number of replications which in turn reduces the accuracy of result. However the simulation time can be quite often very long. Also the simulation methods (except regenerative or perfect simulation) are generally transient simulations which tend to introduces a bias. In the following examples we compare analytic-numeric methods with the standard discrete event simulation.

2.2 Problem description

For our illustration, we consider a distributed system consisting of N workstations and one file server interconnected by a local area network. For a complete description of the model, see ([16]). To avoid confusion in the use of the word “token" which is used both in ring network and in PN, we will refer it either as the “network token" or the “PN token", instead of just “token". We assume that the client-station generates requests which follow an exponential distribution with rate λ and that the transmission time of this request is also exponentially distributed with rate μ. To simplify the analysis all other times are also assumed to be exponentially distributed. This includes the time for the network token to move from a station to another one (rate γ), the request processing time for the server (rate η) and the reply transmission time (rate β). Figure 1 shows the SRN model for a token ring network-based system with five client stations.

Places P_{CkI} ($1 \leq k \leq N$) represent the condition that station k is idle and transition tak that a request is generated at station k. Then the PN token moves to place P_{CkA} where the client is waiting for the network token to arrive (condition represented by place P_{CkS}). When the network token arrives at station k, the transmission of the request can be processed (transition tsk). Then a PN token is put in place $P_{C(k+1)P}$ (or P_{SP} if $k = N$, or P_{C1P} if the current station is the server), which means the network token is waiting to move to the next station (or to the server for P_{SP}). This move is made by the firing of the corresponding transition tkp or tsp. Place P_{SI} represents the condition that the client's request has arrived at the server, where it is served by firing transition tsa. Place P_{SA} represents the condition that the request is completed. When the server has received the network token (condition represented by place P_{SS}) , it commences to transmit an answer (by firing transition tss). Next we describe the modeling of the server's buffer. Places P_{Wk} represent the condition that a request is waiting for its reply at k^{th} slot of the queue from the tail. The multiplicity of input arcs from transitions tsk to P_{W1} is k, which identifies the requesting stations. The firing of transition sk ($1 \leq k \leq N$) means that the server sends a reply to station k. A token in place P_{SW} means that the service is completed. In ([16]), an approximation of this model is described to reduce the state space size. This is done by considering a tagged client and lumping the remaining clients into one super-client. The SRN for this approximation for the system with $N = 5$ stations is given in Figure 2. For a detailed description of this model, see ([16]). Table 1 describes the state space and storage requirements for both the exact and approximate models, and when a memory overflow is obtained on a Sun SparcStation Ultra 60 with 640Mb of real memory and 982Mb of swapping memory. Nonzero entries are the number of nonzero elements in the infinitesimal generator of the underlying continuous time Markov chain. We observe that the state space size of the exact model increases quickly and becomes too large to construct the reachability graph for small values of N ($N = 8$). On the other hand, the approximate model reachability graph can be generated for larger values of N, but limited to $N = 31$ on the same computer.

2.3 Transient behavior

2.3.1 *Cumulative measure*

Fig. 3 gives the results obtained when computing the transient cumulative probability that the server is idle (i.e., places P_{SA} and

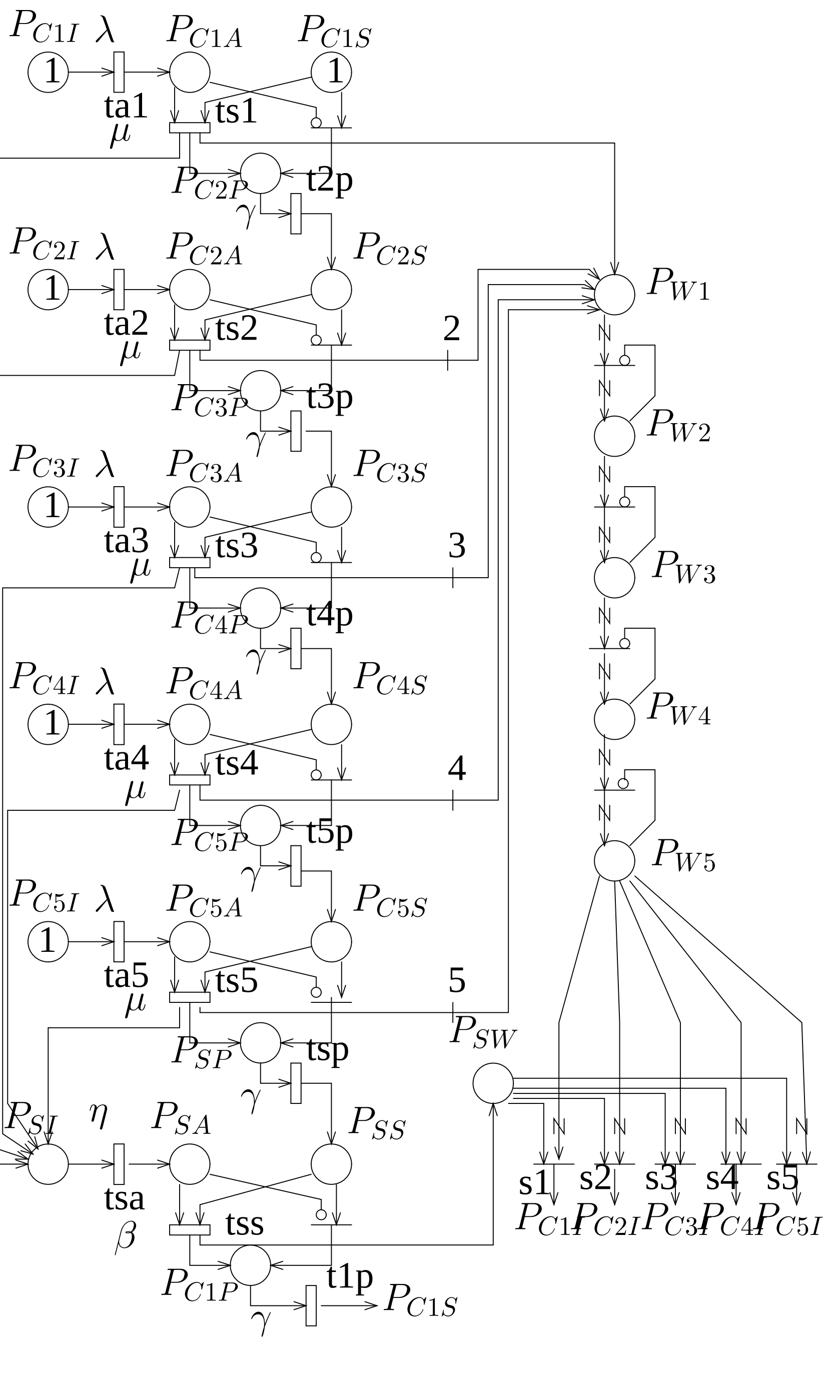

Figure 1: SRN the accurate token ring network ($N = 5$)

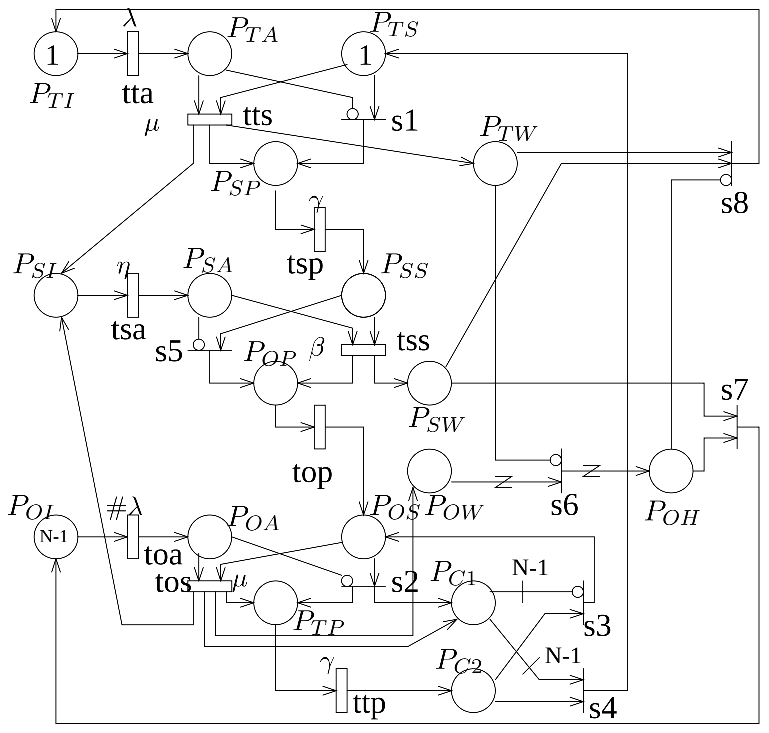

Figure 2: SRN the approximate token ring network ($N = 5$)

N	No. of states (exact model)	Nonzero Entries (exact model)	No. of states (approx. model)	Nonzero Entries (approx. model)
3	476	1004	274	562
4	3416	7960	790	1754
5	26672	66192	1880	4400
6	228880	591568	3920	9524
7	2160160	5736992	7420	18536
8	Memory overflow	Memory overflow	13044	33292
10			34210	90050
15			209240	574700
20			785470	2204850
25			2230150	6343250
30			5281780	15156400
31			6172720	17738324
32			Memory overflow	Memory overflow

Table 1: Storage requirements for the client/server example as a function of N

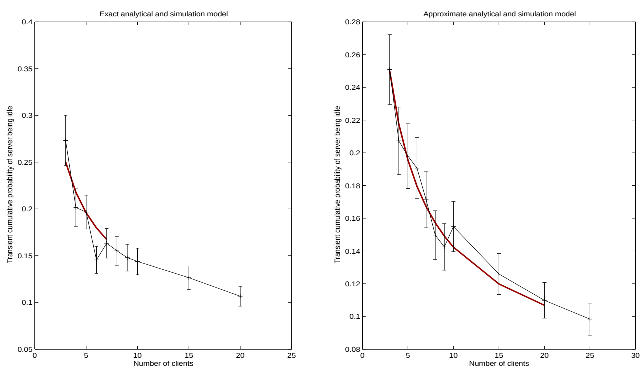

Figure 3: Results from the analytic-numeric and simulation methods for the cumulative behavior. Analytic-numeric methods are plotted in bold, while simulation are plotted with thinner lines, and confidence interval.

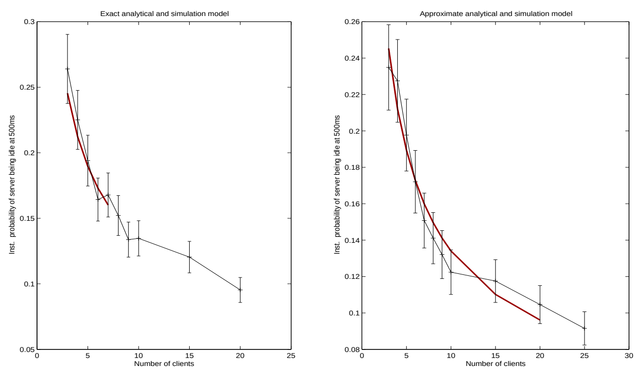

Figure 4: Analytic-numeric results and 10% half-width relative error 95% confidence interval simulation results for the instantaneous behavior at $t = 500$ ms. Analytic-numeric methods are plotted in bold, while simulation are plotted with thinner lines, and confidence interval.

P_{SI} are empty) up to time $t = 500$ ms, using analytic-numeric methods and simulation (with arbitrary 10% relative error and 95%-confidence interval) when the number N of workstations is varying. In these computations, we assume (other values given similar results) that $1/\eta = 2.0$ ms, $1/\mu = 0.1$ ms, $1/\beta = 3.2$ ms, $1/\gamma = 0.01/(N+1) + 0.0024$ ms and that λ is varied with N so that the offered load $\rho = N\lambda[(1/\mu) + (1/\beta)]$ is fixed to 0.9. Also, note that $t = 500$ ms is much larger than the largest average firing time.

The analytic-numeric method uses uniformization technique ([19]) (with absolute precision 10^{-8}) whereas the simulation method is the standard discrete event simulation using independent replications. The number of replications is unknown as the simulation is stopped only when a 10% half-width relative error 95% confidence interval is reached. We do not give all the results because the computation time can be very long for large values of N. We observe that the numerical values given by the approximate model are very close to the exact ones. Of course, we obtain exact results only for small values of N so that we can not be sure that the results are very good also for larger values, but simulation results show that the approximation is still accurate as N increases. The simulation results are in the range of the analytic-numeric ones. Only in the case of the exact model with $N = 6$ the exact value is not included in the confidence interval, probably due to the 5% risk of the interval. In simulation, we get a region where the solution might lie with certain probability, instead of exact answer: one drawback of simulation. In conclusion about Fig. 3, simulation allows us to solve larger models, but this is at the cost that we obtain only a confidence interval, i.e., we have only a given probability that the true value is contained in the interval.

Fig. 5 shows the computation times (including state space generation) required for the methods to obtain the results. The computation time for the analytic-numeric solution of the exact model increases very fast with the state space size. The increase is slower for the approximate model. Simulation time to obtain the 10% relative error confidence interval becomes competitive for $N = 5$ and is close to the analytic-numeric time for smaller N. For this example, simulation provides a fast solution (and even the only one as soon as there is a memory overflow while using analytic-numeric methods; see Fig.1). The reason for different simulation times between the exact and the approximate model is due to the number

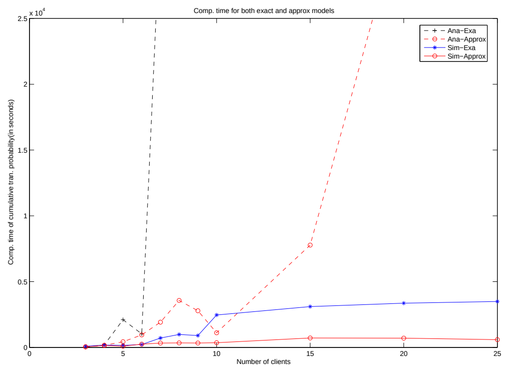

Figure 5: Computation times to obtain for the cumulative estimation.

of events to simulate per run. Recall that the state space does not have no role in this difference as we do not generate it. It is specific to the model: in the exact model, more transitions occur, which results in a bigger number of events per run. This leads to important observation that simulation of approximate models can be much faster than simulation of exact models.

2.3.2 *Instantaneous behavior*

We now compute the transient probability that the server is idle (i.e., places P_{SA} and P_{SI} are empty) at given time t, using analytic-numeric methods and simulation (with 10% relative error 95%-confidence interval) with the number N of workstations varying. All other parameters have the same values as before. The results for $t = 500$ ms are displayed in Fig. 4 and the computation times are given in Fig. 6. Applying the numerical solution method to the exact model requires a long time for $N = 5, 6, 7$ and after that it cannot be applied. The method performs well on the approximate model (restricted to the fact that $N < 31$), but the running time increases quickly with N. Comparing with simulation, we find that the value obtained with analytic-numeric methods fall into the corresponding confidence intervals. If we compare the simulation results for the models, we see that the approximate model gives about

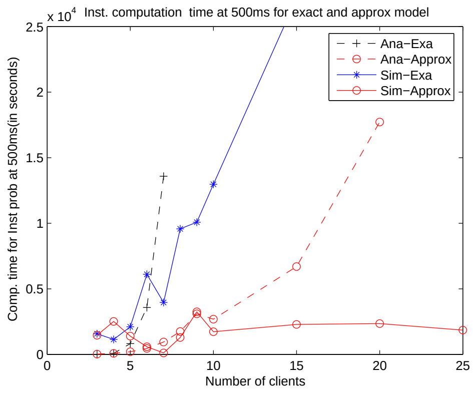

Figure 6: Computation times for the instantaneous behavior at $t = 500$ ms.

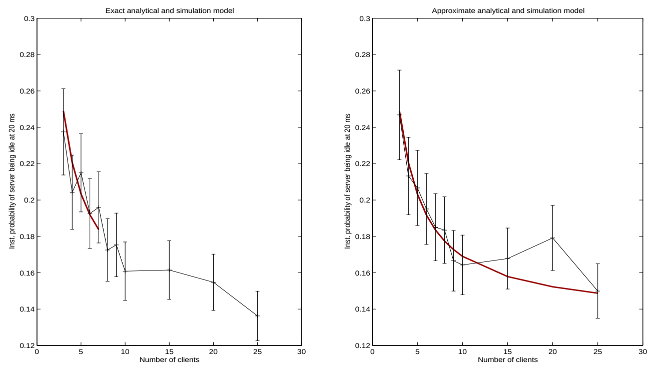

Figure 7: Analytic-numeric results and 10% half-width relative error 95% confidence interval simulation results for the instantaneous behavior at $t = 20$ ms. Analytic-numeric methods are plotted in bold, while simulation are plotted with thinner lines, and confidence interval.

the same results as the exact one even for large values of N. The problem here is that simulation times are very long. Since to obtain just one replication at time $t = 500$ ms, we need to simulate the whole path to this time which can become very long. We also note as in the previous subsection that the simulation of the approximate model is much quicker because we have less number of events to deal with. We suggest then to use the simulation of the approximate model as soon as N gets close to 15 as then simulation time is almost half that of the numeric method one.

Next we consider the results for a smaller time horizon, say for instance $t = 20$ ms. The results are presented in Fig. 7 and the computation times in Fig. 9. Similar remarks can be applied to the results as in the case $t = 500$ ms (but in one case, approximate model with $N = 20$, the exact value is not included in the confidence interval, due to the statistical risk), with the difference that the running times are smaller because the time horizon is smaller. Simulating the approximate model is the best method when N gets close to 20 and the simulation times are then less than 20 minutes.

As a conclusion of this subsection, simulation for instantaneous behavior is better when “small” time horizons are used. Anyway, it becomes the only solution when the state space is big (here when

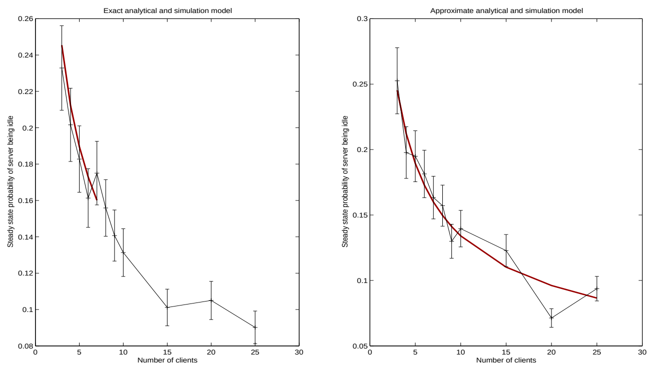

Figure 8: Analytic-numeric and 10% half-width relative error 95% confidence interval simulation results for the steady-state behavior. Analytic-numeric methods are plotted in bold, while simulation are plotted with thinner lines, and confidence interval.

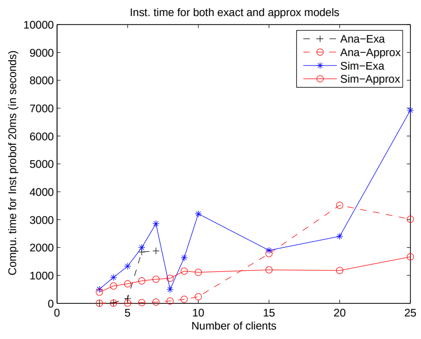

Figure 9: Computation times for the results for the instantaneous behavior at $t = 20$ ms.

$N > 31$ from Table 1).

2.4 Steady-state behavior

We now compute the steady-state probability of the server being idle (i.e., places P_{SA} and P_{SI} are empty) using analytic-numeric methods and simulation (with 10% relative error 95%-confidence interval) with the number N of workstations varying.

The analytic-numeric method used is steady-state SOR. For simulation, we first use the standard discrete event simulation with batch technique. In this, a single long simulation path is generated till we are sure that steady-state is reached (a warm-up can be used). This single run is decomposed into several batches which can be considered independent, so that a statistical analysis is performed using the method of batch means ([24]).

The results for analytic-numeric methods and the simulation using batch methods (with a batch size of $t = 5$ ms) are displayed in Fig. 8. The computation times are presented in Fig. 10. The analytic-numeric solution performs well when applied on the exact model until $N = 6$. Beyond $N = 6$, the running time is long. Use of the approximate model is much better as the results are very close and the running times are much smaller (18 seconds as compared with 3 hours and 50 minutes for $N = 7$). But even the approximate model can not be used if $N > 31$. Simulation of the exact model is powerful for small values of N but the simulation

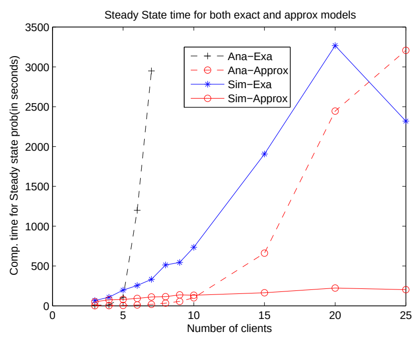

Figure 10: Computation times to obtain the results for the steady-state behavior.

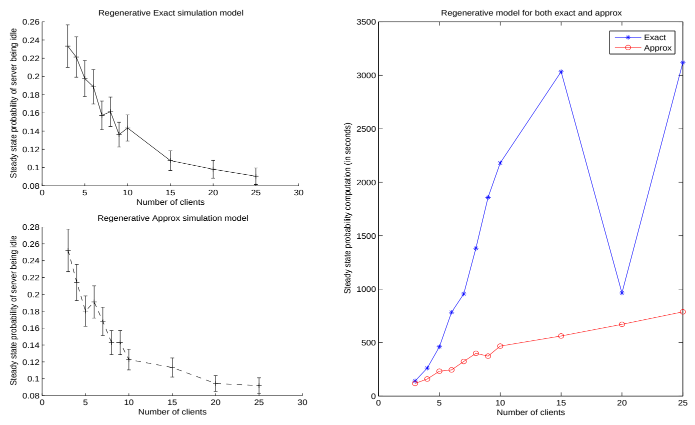

Figure 11: Regenerative simulation results and computation times for the steady-state behavior.

time increases with N. On the other hand, simulation of the approximate model requires very small running times (about 3 minutes 30 seconds for $N = 20$ or $N = 25$). Thus the simulation of the approximate is a very powerful technique.

One inherent problem while using batch means method is the choice of the batch size. It has to be big enough so that successive blocks are nearly independent. In our case, the exact value gets closer to the edge of the confidence interval as N increases. This suggests that the batch size should be increased with N. Indeed, when N increases the number of tokens in the system is larger and hence more events have to occur within a batch to decorrelate blocks.

Regenerative simulation is also applied to perform steady-state simulation ([13]). The advantage is that the variance estimation is unbiased. The drawback is that we need a regeneration point (each state is suitable in the Markovian case) where a cycle will begin. Each cycle is then independent and a statistical analysis can be performed. Confidence intervals and computation times for regenerative simulation are given in Fig. 11. The exact values look less at the edge of the confidence intervals. One observation, at least in our client-server model is that batch method is quicker than regenerative simulation. Moreover, as N increases, the mean length of a cycle increases, making it more difficult to estimate the variance of the estimator.

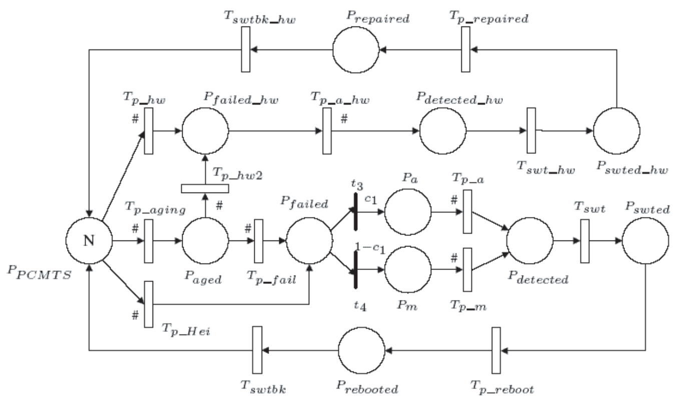

Figure 12: SRN model for the CMTS system

Note that another way to perform regenerative simulation is using A-cycles ([6]). Instead of using a state as a regeneration point, we use a subset of states, which increases the number of regenerations. This estimator is biased (because the initial state is not generated from the regenerative subset with the steady-state distribution, which would be too difficult) and the cycles are dependent. The analysis is then more tricky and more approximate than the classical regenerative method.

3. AVAILABILITY OF CABLE MODEM TERMINATION SYSTEM (CMTS)

We describe the model of cable modem termination system and solve such model analytic-numerically and by discrete event simulation. In this example we will calculate capacity-oriented availability and calculation time required. The CMTS serves as the backbone of the hybrid fiber coaxial (HFC) cable network and provides connectivity between the cable network and the Internet for both upstream and downstream traffic. In addition, the CMTS is also responsible for management services such as billing, authorization, quality of service (QoS) control, and protocol conversion. The high availability of CMTS is crucial for cable operators to provide carrier-class services to all its subscribers. Due to the importance of CMTS in cable modem systems, hardware redundancy and the corresponding resilient software features are introduced in CMTS to achieve high availability as a traditional approach. The cluster comprises N primary CMTS (PCMTS) nodes, one secondary CMTS (SCMTS) node [18].

3.1 Paradigm and solution methods

We solve capacity oriented availability for CMTS problem analytic-numerically by using SHARPE ([14]) and by discrete event simulation using CSIM 19. We model the system as stochastic reward net(SRN) in SHARPE. SHARPE is a reliability/availability and performance evaluator. CSIM 19 is a commercial process-oriented discrete event simulator and it provides software libraries for C and C++. More details about SHARPE (which also uses SRNs to model systems) and CSIM 19 are described in Appendices A.3 and A.4.

3.2 Representation of the CMTS in this paradigm

Figures 12 and 13 show the SRN model of the CMTS system. In this model, a PCMTS and a SCMTS have the same failure behavior, i.e., they may fail due to either hardware failures or software fail-

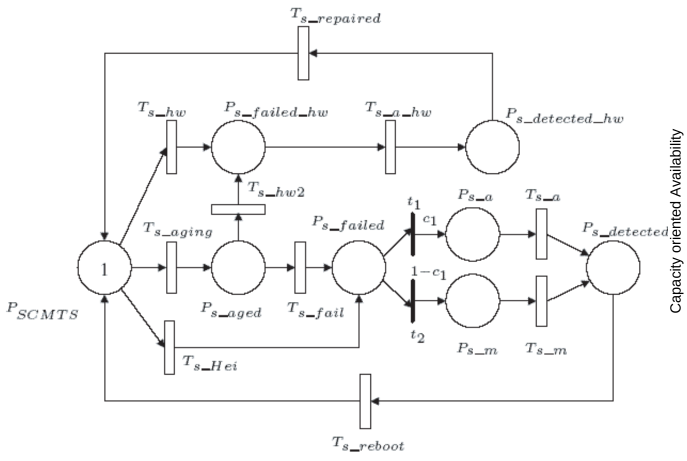

Figure 13: SRN model for the SCMTS system

λ_{hw}	Hardware failure rate	1/53328
λ_{hei}	Software failure (caused by Heisenbugs) rate	1/10000
λ_{age}	age Software aging rate	1/200
λ_f	Software failure rate (caused by aging-related bugs)	1/200
λ_{swt}	Node switching rate	120
λ_{rbt}	Node reboot rate	10
λrep	Node repair rate	1/4
λ_a	Automatic failure detection rate	120
λ_m	Manual failure detection rate	1/4
λ_{c1}	Coverage of automatic failure detection	0.95

Table 2: Rates for the CMTS model

ures. In the later case, failures may be caused by either Heisenbugs or aging-related bugs. The two submodels differ in their recovery behaviors. Whenever a PCMTS needs to repair/reboot, it has to switchover the ongoing tasks to the SCMTS. However, the repair action of the SCMTS is carried out without considering the state of any PCMTS. This dependency is reflected in the enabling functions and priorities of various transitions (see [18] for details).

As the failure of a PCMTS node does not affect the operations on other PCMTS nodes, the CMTS cluster is still available with regard to those customers connected with other operational PCMTS nodes. The whole CMTS cluster becomes unavailable only when all the $N+1$ nodes fail. Although system availability can be computed according to this definition, we choose capacity oriented availability (COA) as our availability measure since it reflects the perception from customers:

$$COA = \frac{N_a}{N} \tag{1}$$

where N_a denotes the average number of available PCMTS nodes and N is the total number of PCMTS nodes in the cluster. To obtain capacity oriented availability we assign reward rate functions to our model as

$$COA = (\#P_{PCMTS} + \#P_{aged} + \#P_{swted} + \#P_{swted\_hw})/N. \tag{2}$$

For more details see ([18]).

3.3 Transient behavior

Figure 14 shows the result obtained from CSIM 19 simulation and SHARPE for time $t = 80000hrs$. It also shows the actual calculation time taken for both methods to obtain the transient simulation at $t = 80000hrs$. In both cases 95% confidence interval has been overlayed with the simulation result. The state space for SRN grows quickly as we increase the number of primary PCMTS. SHARPE SRN model calculates capacity oriented availability quickly when number of primary PCMTS is below 5. But we see that beyond 5, the state space of CMTS model explodes and the analytic model is not able to solve the model. On the other hand simulative model scales very well to increase in number of primary PCMTS nodes. We see that simulation calculation time increases marginally as we increase number of primary nodes. This strengthens the belief that when large state space exists it makes more sense to use simulation as means of model solving.

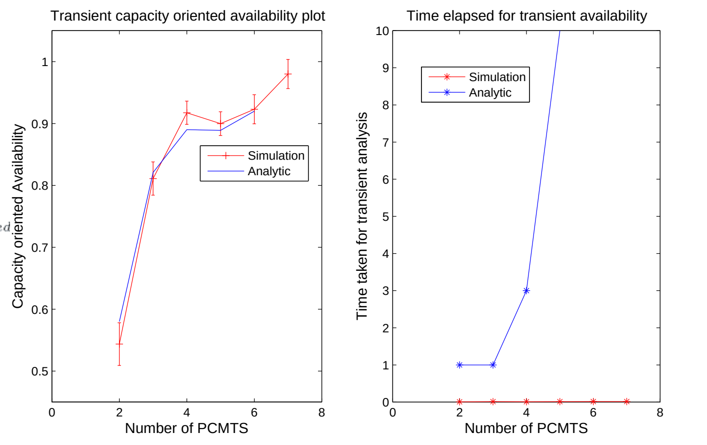

Figure 14: Transient capacity oriented availability (COA) at $t = 80000$

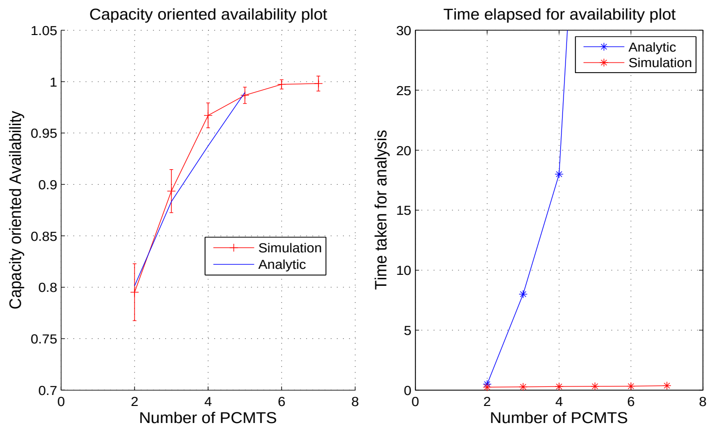

Figure 15: Capacity-oriented availability and computation times for the steady-state behavior

3.4 Steady-state behavior

Figure 15 shows the results from CSIM simulation and SHARPE. 95% confidence interval is plotted for each simulation run. It also shows the actual calculation time taken for both methods to obtain the steady state simulation. We observe that the underlying state space generated by the SRN model explodes quickly and the SHARPE analytic-numeric engine is not able to solve the system for $N > 5$. Whereas CSIM simulation scales well to increase in redundancy for primary CMTS modes. We also note that actual calculation times also scales well for discrete vent simulation case whereas it increases rapidly for analytic-numeric case.

4. CONCLUSION

To choose between simulation and analytic-numeric methods, we make the following recommendations (This follows a similar study on other examples/models):

- When the system is *non-Markovian* (especially without regenerative structure), very few analytic-numeric methods are available. Simulation is then the natural and often the only possibility.

- When the system is *Markovian*
  - When the state space is big (and no approximate model close to the exact model reducing the state space is available), the reachability graph can not be generated. Simulation is then again the only possibility (in our example if $N > 31$).
  - In the other cases, a choice is very specific to the application. Nevertheless, from our experience and the example of this paper, we can say that for "small" state spaces analytic-numeric methods perform well. When the state space increases, there is always a point where simulation time (for a given precision) is smaller (the worst situation is when the reachability graph is too big to be generated). But we can point out that the natural switch from analytic-numeric to simulation methods is faster for steady-state behavior, then cumulative transient behavior and finally instantaneous transient behavior (except for very small time horizons). Usually, simulation takes long time to obtain good results for long horizon instantaneous behavior.

    Note that even if the running time of analytic-numeric methods is a little larger than the one of simulation, it is still relevant to use them because they give an accurate result instead of a confidence interval. How different it should be is very subjective and depends on the user.

    Moreover, using approximate models can be very helpful even when using simulation, as pointed out in this paper. Indeed, even if the state space is not generated as in analytic-numeric methods, an approximate model can tremendously reduce the computation time per run by reducing the number of events.

5. REFERENCES

[1] CSIM 19 Simulator: `http://www.mesquite.com/`, 2005.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

[3] M. Ajmone Marsan, G. Conte, and G. Balbo. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.

[4] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, NJ, third edition, 2001.

[5] A. Bobbio, V. Kulkarni, M. Puliafito, A. an d Telek, and K. Trivedi. Preemptive Repeat Identical Transitions in Markov Regenerative Stochastic Petri Nets. In *Proc. of Petri Nets Performance Models 1995*, pages 113–122. IEEE CS Press, 1995.

[6] C. Chang, P. Heidelberger, and P. Shahabuddin. Fast Simulation of Packet Loss Rates in a Shared Buffer Communications Switch. *ACM Transactions on Modeling and Computer Simulation*, 5(4):306–325, October 1995.

[7] H. Choi, V. Kulkarni, and K. Trivedi. Markov Regenerative Stochastic Petri Nets. *Performance Evaluation*, 20(1-3):337–357, 1993.

[8] H. Choi, V. Kulkarni, and K. Trivedi. Transient analysis of deterministic and stochastic petri nets. In M. A. Marsan, editor, *Application and Theory of Petri Nets 1993*, volume 691 of *Lecture Notes in Computer Science*, pages 166–185. Springer Verlag, 1993.

[9] G. Ciardo, A. Blakemore, P. Chimento, J. Muppala, and K. Trivedi. Automated Generation and Analysis of Markov Reward Models using Stochatic Reward Nets. In C. Meyer and R. Plemmons, editors, *Linear Algebra, Markov Chains and Queuing Models*, volume 48 of *IMA Volumes in Mathematics and its Applications*, pages 145–191. Springer-Verlag, Heidelberg, 1993.

[10] G. Ciardo, J. Muppala, and K. Trivedi. SPNP: Stochastic Petri net Package. In *Proc. Third International Workshop on Petri Nets and Performance Models, PNPM'89*, pages 142–151. IEEE CS Press, 1989.

[11] G. Ciardo, D. Nicol, and K. Trivedi. Discrete-Event Simulation of Fluid Stochastic Petri-Nets. *IEEE Transactions on Software Engineering*, 25(2):207–217, 1999.

[12] G. Ciardo and K. Trivedi. A Decomposition Approach for Stochastic Reward Net Models. *Performance Evaluation*, 18(1):37–59, 1993.

[13] G. Fishman. *Monte Carlo: Concepts, Algorithms and Applications*. Springer-Verlag, 1996.

[14] C. Hirel, R. A. Sahner, X. Zang, and K. S. Trivedi. Reliability and performability modeling using SHARPE 2000. In *Proceedings of the 11th International Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, pages 345–349. Springer-Verlag, London, UK, 2000.

[15] C. Hirel, B. Tuffin, and K. Trivedi. SPNP Version 6.0. In B. Haverkort, H. Bohnenkamp, and C. Smith, editors, *Computer performance evaluation: Modelling tools and techniques; 11th International Conference; TOOLS 2000, Schaumburg, Il., USA*, volume 1786 of *Lecture Notes in Computer Science*, pages 354–357. Springer Verlag, 2000.

[16] O. Ibe, H. Choi, and K. Trivedi. Performance Evaluation of Client-Server Systems. *IEEE Transactions on Parallel and Distributed Systems*, 4(11):1217–1229, 1993.

[17] A. M. Law and W. D. Kelton. *Simulation Modeling and Analysis*. McGraw Hill Higher Education, third edition, 2000.

[18] Y. Liu, Y. Ma, L. H. Han, J.J, and K.S. Trivedi. A proactive approach towards always-on availability in broadband cable networks. *Computer Communications*, 28(1):51–64, January 2005.

[19] M. Malhotra, J. Muppala, and K. Trivedi. Stiffness-tolerant methods for transient analysis of stiff Markov chains. *Microelectronics Reliability*, 34(11):1825–1841, 1994.

[20] J. Muppala and K. Trivedi. Composite Performance and

Availability Analysis using a Hierarchy of Stochastic Reward Nets. In G. Balbo and G. Serazzi, editors, *Computer Performance Evaluation, Modelling Techniques and Tools*, pages 335–350. Elsevier, Amsterdam, 1992.

[21] M. Neuts and K. Meier. On the use of phase type distributions in reliability modelling of systems with two components. *Oper. Res. Spektrum*, 2(4):227–234, 1981.

[22] J. Peterson. *Petri nets and the Modeling of Systems*. Prentice-Hall, EnglewoodCliffs, NJ, 1981.

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

[24] K. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications*. John Wiley & Sons, 2002. Second Edition.

[25] B. Tuffin and K. Trivedi. Implementation of importance splitting techniques in stochastic Petri net package. In B. Haverkort, H. Bohnenkamp, and C. Smith, editors, *Computer performance evaluation: Modelling tools and techniques; 11th International Conference; TOOLS 2000, Schaumburg, Il., USA*, volume 1786 of *Lecture Notes in Computer Science*, pages 216–229. Springer Verlag, 2000.

APPENDIX

A. STOCHASTIC REWARD NETS AND SOFTWARES FOR MODELING

A.1 Stochastic Reward Nets

Stochastic reward net (SRN) [12] is an extension of Petri net (PN). SRN has been widely used in the area of performance and dependability analysis due to its conciseness and clarity in visual and conceptual presentation. A PN is a bipartite directed graph with two types of nodes: places and transitions. Each place may contain an arbitrary (natural) number of tokens. Each transition may have zero or more input arcs, coming from its input places; and zero or more output arcs, going to its output places. A transition is enabled if all of its input places have at least as many tokens as required by the multiplicities of the corresponding input arcs. Generalized stochastic Petri nets (GSPNs) [2] extend the PNs by assigning a firing time to each transition. Transitions with exponentially distributed firing times are called timed transitions while the transitions with zero firing times are called immediate transitions. For a given GSPN, an extended reachability graph (ERG) is generated with the markings of the reachability set as the nodes and some stochastic information attached to the arcs, thus connecting the markings to each other. In order to make more compact models of complex systems, several extensions are made to GSPN, leading to the SRN. One of the most important features of SRN is its ability to allow extensive marking dependency. In an SRN, each tangible marking can be assigned with one or more reward rates. Parameters such as the firing rate of the timed transitions, the multiplicities of input/output arcs and the reward rate in a marking can be specified as functions of the number of tokens in any place in the SRN. Another important characteristic of SRN is the ability to express complex enabling/disabling conditions through guard functions. This can greatly simplify the graphical representations of complex systems. For an SRN, all the output measures are expressed in terms of the expected values of the reward rate functions. To get the performance and reliability/availability measures of a system, appropriate reward rates are assigned to its SRN. As an SRN can be automatically transformed into a Markov reward model (MRM) [20], steady state and/or transient analysis of the MRM produces the required measures of the original SRN. In this paper, we use the tools SPNP [10, 15] and SHARPE [14] to specify and solve the SRN models.

A.2 SPNP

In the SPNP package ([10, 15]), the model type used for input is a stochastic reward net (SRN). SRNs incorporate several structural extensions to GSPNs such as marking dependencies (marking dependent arc cardinalities, guards, etc.) and allow reward rates to be associated with each marking. The reward function can be marking dependent as well. They are specified using CSPL (C based SRN Language) which is an extension of the C programming language with additional constructs for describing the SRN models. SRN specifications are automatically converted into a Markov reward model which is then solved to compute a variety of transient, steady-state, cumulative, and sensitivity measures. For SRNs with absorbing markings, mean time to absorption and expected accumulated reward until absorption can be computed. Both analytic-numeric and simulation techniques are available

A.3 CSIM 19

Simulation of CMTS model is done using CSIM 19 [1]. It is a process-oriented discrete event simulator implemented in C/C++ as a library of routines which perform necessary operations. It provides methods for calculating confidence interval by batch means analysis [4]. It provides methods for random variates generation from several distributions. Run length control [17] for terminating simulation is also possible when desired accuracy has been achieved and there is no need to simulate the model further.

A.4 SHARPE

SHARPE, (Symbolic Hierarchical Automated Reliability and Performance Evaluator, [14]) is a tool for specifying and analyzing performance, reliability and performability models. It has been installed at over 250 sites. It is a toolkit that provides a specification language and solution methods for most of the commonly used model types for performance, reliability and performability modeling. Model types include combinatorial one such as fault-trees and queuing networks and state-space ones such as Markov and semi-Markov reward models as well stochastic Petri nets. Steady-state, transient and interval measures can be computed. Output measures of a model can be used as parameters of other models. This facilitates the hierarchical combination of different model types.