Tool supported reliability analysis of finite-source retrial queues

János Sztrik
Faculty of Informatics
University of Debrecen
4012 Debrecen
Hungary
sztrik.janos@inf.unideb.hu

Dmitry Efrosinin
Institute of Mathematics
Johannes Kepler University
4040 Linz
Austria
dmitry.efrosinin@jku.at

Abstract

This paper deals with the role of performance modeling tools. It introduces 4 major tool developer centers and shows how a given tool can be applied to reliability investigations of finite-source retrial queueing system in steady state. Some numerical examples are given demonstrating the effect of failure and repair rates of server on the mean response time of sources.

1 Introduction

The argument for performance engineering methods and tools to be employed in reliability analysis of complex systems has always been that such systems cannot be designed or modified efficiently without recourse to some form of predictive model, just as in other fields of engineering. To be practical, performance engineering relies on tools to render its use accessible to the non-performance specialist, and in turn these depend on sound techniques that include analytical methods, stochastic models and simulation. Many mathematical techniques have been developed to derive various measures from Markov reward models which form the basis of almost all performability models and tools.
The aim of the present paper is to introduce some tools and to show how a certain tool can be applied to reliability investigations of finite-source retrial queueing system in steady state. Some numerical examples are given demonstrating the effect of failure rate of server on the mean response time of sources.

2 Some Recent Modeling Tools

In the following 4 major tool developer centers are introduced briefly.

2.1 Tools at Faculty of Informatics, University of Dortmund, Germany

This traditionally famous center has developed several software packages which can be downloaded from the site: `http://ls4-www.informatik.uni-dortmund.de/tools.html`. Parallel to their methodological work the center continuously developed and used tools for performance evaluation and performability. Their intention is to provide facilities for a model description close to the original system specification and hiding details of the analysis techniques. Such tools map the model specification automatically to an analysable model.

2.2 Möbius Tool

Möbius^{TM} is a software tool for modeling the behavior of complex systems.It is one of the major research projects of the Performability Engineering Research Group (PERFORM) in the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign, USA. Many advanced modeling formalisms and innovative and powerful solution techniques have been integrated in the Möbius framework. The package can be found at `http://www.mobius.uiuc.edu/`.

2.3 MOSEL Tool

Performance modeling tools usually have their own textual or graphical specification language which depends largely on the underlying modeling formalism. MOSEL (MOdeling,Specification and Evaluation Language) tool, developed at the University of Erlangen, Germany, is based on the following idea: Instead of creating another tool with all the components needed for system description, state space generation, stochastic process derivation, and numerical solution, it focuses on the formal system description part and exploit the power of various existing and well tested packages for the subsequent stages. In order to reuse existing tools for the system analysis, the environment is equipped with a set of translators which transform the MOSEL model specification into various tool-specific system descriptions (Begain, Bolch, and Herold 2001).
MOSEL-2 provides means by which many interesting performance or reliability measures and the graphical presentation of them can be specified straightforwardly. All information can be found at `http://www4.informatik.uni-erlangen.de/Projects/MOSEL/`.

2.4 PRISM Tool

The PRobabIliStic Model checker PRISM provides a high-level modeling language for describing systems that exhibit probabilistic behavior, with models based on continuous-time Markov chains (CTMCs) as well as discrete-time Markov chains (DTMCs) and Markov decision procedures (MDPs). For specifying system properties, PRISM uses the continuous stochastic logic (CSL) for CTMCs and probabilistic computation tree logic (PCTL) for DTMCs and MDPs, both logics being extensions of computation tree logic (CTL), a temporal logic that is used in various classical model checkers for specifying properties. The package can be downloaded from `http://www.prismmodelchecker.org`.

3 Reliability Analysis of Finite-Source Retrial Queues

To show an example for using MOSEL we analyze a retrial queueing system with the following assumptions. Consider a single server queueing system, where the primary calls are generated by K, $1 < K < \infty$ homogeneous sources. The server can be in three states: idle, busy and failed. If the server is idle, it can serve the calls of the sources. Each of the sources can be in three states: free, sending repeated calls and under service. If a source is free at time t it can generate a primary call during interval $(t, t + dt)$ with probability $\lambda dt + o(dt)$. If the server is free at the time of arrival of a call then the call starts to be served immediately, the source moves into the under service state and the server moves into busy state. The service is finished during the interval $(t, t + dt)$ with probability $\mu dt + o(dt)$ if the server is available. If the server is busy at the time of arrival of a call, then the source starts generation of a Poisson flow of repeated calls with rate ν until it finds the server free. After service the source becomes free, and it can generate a new primary call, and the server becomes idle so it can serve a new call. The server can fail during the interval $(t, t + dt)$ with probability $\delta dt + o(dt)$ if it is idle, and with probability $\gamma dt + o(dt)$ if it is busy. If $\delta = 0, \gamma > 0$ or $\delta = \gamma > 0$ *active or independent breakdowns* can be discussed, respectively. If the server fails in busy state, it either *continues servicing* the interrupted call after it has been repaired or the interrupted request *transmitted to the orbit*. The repair time is exponentially distributed with a finite mean $1/\tau$. If the server is failed two different cases can be treated. Namely, *blocked sources* case when all the operations are stopped, that is neither new primary calls nor repeated calls are generated. In the *unblocked (intelligent) sources* case only service is interrupted but all the other operations are continued (primary and repeated calls can be generated). All the times involved in the model are assumed

	K	λ	μ	ν	δ, γ	τ
Figure 1	6	0.8	4	0.5	x axis	0.1
Figure 2	6	0.1	0.5	0.05	x axis	0.1
Figure 3	6	0.8	4	0.5	0.05	x axis
Figure 4	6	0.1	0.5	0.05	0.05	x axis

Table 1: Input system parameters

to be mutually independent of each other. More information about this type of systems can be read in (Almási, Roszik, and Sztrik 2005), (Artalejo and Gomez-Corral 2008), (Falin and Templeton 1997).

Our main objective is to demonstrate how the steady state performance measures, such as mean number of requests staying in the orbit or in the service, the mean response time of calls depend on server's failure and repair rates. To achieve this goal MOSEL is used to formulate and solve the problem.

The system state at time t can be described with the process
$X(t) = (Y(t); C(t); N(t))$, where $Y(t) = 0$ if the server is up, $Y(t) = 1$ if the server is failed, $C(t) = 0$ if the server is idle, $C(t) = 1$ if the server is busy, $N(t)$ is the number of sources of repeated calls at time t. Because of the exponentiality of the involved random variables this process is a Markov chain with a finite state space. Since the state space of the process $(X(t), t \geq 0)$ is finite, the process is ergodic for all reasonable values of the rates involved in the model construction, hence from now on we will assume that the system is in the steady state. Knowing the steady state probabilities the main performance measures can be obtained by the usual arguments.

3.1 Numerical examples

We used the tool SPNP which was able to handle the model with up to 126 sources. In this case, on a computer containing a 1.1 GHz processor and 512 MB RAM, the running time was approximately 1 second.

In Figures 1,2 we can see the mean response time, and mean number of calls staying in the orbit or in the service for the reliable and the non-reliable retrial system when the server's failure rate increases. In Figures 3,4 the same performance measures are displayed as the function of increasing repair rate. The input parameters are collected in Table 1.

3.2 Comments

In Figure 1, we can see that in the case when the request returns to the orbit at the breakdown of the server, the sources will have always longer response times. Although the difference is not considerable it increase as the failure rate increase. The almost linear increase in $E[T]$ can be explained as follows. In the blocked (non-intelligent) case the failure of the server blocks all the operations and the response time is the sum of the down time of the server, the service and repeated call generation time of the request (which does not change during the failure) thus the failure has a linear effect on this measure. In the intelligent case the difference is only that the sources send repeated calls during the server is unavailable, so this is not an additional time.

In Figure 2 we can see that the mean number of calls staying in the orbit or in service does not depend on the server's failure rate in continuous, non-intelligent case, it coincides with the reliable case. It is because during and after the failure the number of requests in these states remains the same. The almost linear increase in the non-continuous, non-intelligent case can

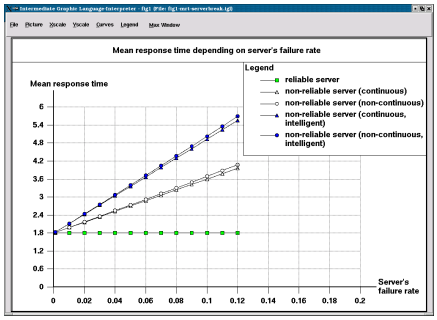

Figure 1: $E[T]$ versus server's failure rate

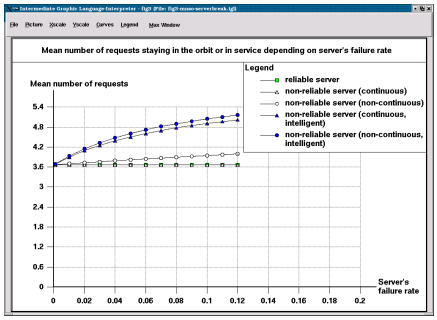

Figure 2: M versus server's failure rate

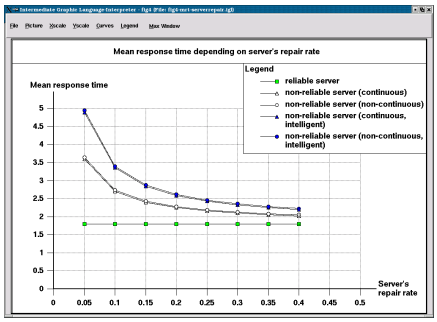

Figure 3: $E[T]$ versus server's repair rate

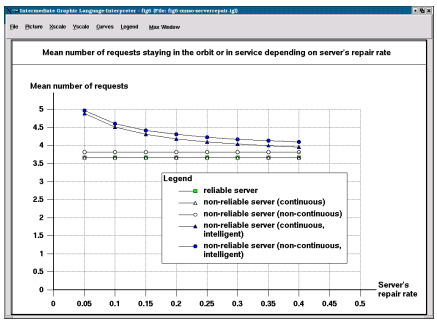

Figure 4: M versus server's repair rate

be explained with that if the server failure occurs more often the server will be idle more often after repair until a source repeats his call.

In Figure 3, we can see that if the request returns to the orbit at the breakdown of the server, the sources will have longer response times like in Figure 1. The difference is not considerable too, and as it was expected the curves converge to the reliable case.

In Figure 4, it can be seen that the mean number of calls staying in the orbit or in service does not depend on the server's repair rate in continuous, non-intelligent case, it coincides with the reliable case like in Figure 2.

Acknowledgements

Research is partially supported by Hungarian Scientific Research Fund-OTKA K60698/2005, and Austro-Hungarian Scientifc Cooperation OMAA 72öu6.

References

Almási, B., J. Roszik, and J. Sztrik (2005). Homogeneous finite-source retrial queues with server subject to breakdowns and repairs. *Mathematical and Computer Modeling 42*.

Artalejo, J. and A. Gomez-Corral (2008). *Retrial Queueing Systems*. Berlin: Springer.

Begain, K., G. Bolch, and H. Herold (2001). *Practical performance modeling, application of the MOSEL language*. Boston: Kluwer Academic Publisher.

Falin, G. and J. Templeton (1997). *Retrial queues*. London: Chapman and Hall.