Modeling a Communication System with Randomly Changing Parameters with MOSEL

Béla Almási,
lecturer
University of Debrecen, Debrecen, Hungary
`almasi@math.klte.hu`

János Sztrik
professor
University of Debrecen, Debrecen, Hungary
`jsztrik@math.klte.hu`

September 1, 2001

Abstract

This paper is concerned with a queuing model to calculate the steady-state behavior of a finite source communication system. The clients are heterogeneous, and random environments change the different parameters. Thus the source times of the messages are supposed to be exponentially distributed random variables with parameters depending on the client station and on the state of the clients varying environment. Similarly, the processing times of the requests are assumed to be exponentially distributed random variables with parameters depending on the source of the request and on the state of a varying environment. Although the model was studied earlier by different authors, analytical results are very hard to find. In this paper we introduce a software tool which can be used to calculate analytical results for the model and we show some numerical results for test purposes.

1 Introduction.

The machine interference problem has been analyzed in many forms over the past years. Recently the mathematical models of the machine repair problem have been efficiently used to describe computer and communication systems (see [1], [3], [5] and [10]).
The precise implementation and practical modeling often requires the introduction of the random environment, sometimes referred as to Markov-modulation, where system parameters are subjected to randomly occuring fluctuations or bursts. This situation may be attributed to certain changes in the physical environment such as personal changes and work load alterations. Gaver *et al.* [7] proposed an efficient computational approach for the analysis of a generalized structure involving finite state space birth-and-death processes in a Markovian environment.
This paper deals with a First-Come, First-Served (FCFS) queueing model to analyze the behaviour of heterogeneous finite-source queueing system with a single server. The request sources and the server are supposed to operate in random environments, thus allowing the arrival and service processes to be Markov-modulated ones. Each request of the sources is characterized by its own exponentially distributed source and service time with parameter depending on the state of the corresponding environment, that is, the request generation and service rates are subject to random fluctuations. Our aim

is to get the usual stationary performance measures of the system, such as utilizations, mean queue lengths, average response times.
In this paper we describe the mathematical model and introduce a software tool to produce analytical computational results for the investigated model. The MARKMOD software package was built on MOSEL and SPNP (see [6], [9]) and gives an easy way to use Markov-modulated queueing systems for modeling real life computer and communication systems. Finally, we show some numerical examples to illustrate the efficiency of the software tool.

2 The Queueing Model.

Consider a finite-source queueing system with N heterogeneous clients (sources) and a single server. The sources and the server operation is influenced by random environments. The server and the clients are collected into M groups $(1 \leq M \leq N+1)$. The members of a group may operate in a common random environment. The environmental changes are reflected in the values of the access and service rates that prevail at any point of time. The main objective is to adapt these parameters to respond to random changes effectively and thus maintain derived level of system performance.
The members of group p are assumed to operate in a random environment governed by an ergodic Markov chain $(\xi_p(t), t \geq 0)$ with state space $(1, \ldots, r_p)$ and with transition density matrix

$$\left(a^{(p)}_{i_p j_p}, i_p, j_p = 1, \ldots, r_p, a^{(p)}_{i_p i_p} = \sum_{k \neq i_p} a^{(p)}_{i_p k} \right).$$

Whenever the environmental process $\xi_p(t)$ is in state i_p the probability that client c (a member of group p) generates a request in the time interval $(t, t+h)$ is $\lambda_c(i_p)h + o(h)$, $p = 1, \ldots, M$. Each request is transmitted to a server where the service immediately starts if it is idle, otherwise a queueing line is formed. The service discipline is First-Come, First-Served (FCFS). Assuming, that the server belongs to group 1 and the environmental process $\xi_1(t)$ is in state i_1 the probability that the service of the request originating from client c is completed in time interval $(t, t+h)$ is $\mu_c(i_1)h + o(h)$.
If a given source has sent a request it stays idle and it can not generate another one. After being serviced each request immediately returns to its source and the whole procedure starts again. All random variables involved here and the random environments are supposed to be independent of each other.
Similar models were studied earlier by different authors (see [7], [11], [12]), but usually simulation tools were used to calculate numerical results. In the followings we shortly introduce a software tool for analytical investigations.

3 MOSEL - The language environment of the implementation.

This section gives a short description of MOSEL - the most important basics of our software tool. MOSEL (MOdeling Specification and Evaluation Language) is a language, developed at the University of Erlangen. The MOSEL system uses a macro-like language (see [9] and [5]) tuned especially to describe stochastic petri nets. We give a short overview on the structure of a MOSEL program, which is useful for studying the MOSEL implementation of the different models discussed later in this paper.
The MOSEL programs consist of four parts: the declarations, the node definitions, the transition rules and the results. We can put comments into the MOSEL program enclosed in /* ... */ (like in C). In all parts we can use shortcuts, to make the program shorter. In the shortcut we enclose the indices (or an interval of indices) in < ... >, then in the same line we can reference to the actual index value with a # character (see below the examples). The example program lines shown in this section do not form a running program, they only illustrate the language.

3.1 The declaration part.

In this part we can define constants, declare variables and enumerated types. The enumerated types can be used to describe the set of the states of a node. Macro definition is also possible using the string keyword. Here is a small example:

```
/*================= Declaration part begins ===================*/

/*======= Definition of a constant MAXNT with value 9 ========*/
#define MAXNT 3

/*========== Definition of a macro NT with value 3 ===========*/
#string  NT  3:

/*== Definition of input variables prgen1, prgen2, prgen3 ===*/
/*=========================== and prrun1, prrun2, prrun3 ===*/
/*= Here we use shortcuts, to make the declarations shorter =*/
<1..$NT> VAR double prgen#; <1..$NT> VAR double prrun#;

/*============= Here is an enum type example ================*/
enum  term_states { busy, waiting };
```

3.2 The node part.

In this part we define the nodes for the system. We can use the constants and enum types defined in the declaration part. We can also give initial values for the nodes, as you can see in the following examples:

```
/*================ The NODE part begins =====================*/

/*== Definition of nodes named term with states term_states ==*/
<1..$NT> NODE term#[term_states] = busy;

/*========= Definiton of a node with capacity NT =============*/
NODE que[$NT] = 0;
```

3.3 The transition rule part.

This is the most important part of the MOSEL program, which describes the system's behavior using FROM ... TO style rules. Please refer to [4] for further details. Here is a small example of the rule part:

```
/*================= The rule part begins ===================*/

/* If the state of term1 is busy, then we can move a job to the
   CPU with rate prgen1, this sets the state of term1 to waiting. */
if term1 == busy {
              FROM  term1[busy]   TO  cpu, term1[waiting]  WITH prgen1;
}
```

```
/* From the queue we can service the job of term1 with rate
prrun1,
   and set the state of term1 to busy. MOSEL automatically checks
   the assumption if term1 == waiting and CPU > 0. */
FROM  cpu, term1[waiting]  TO   term1[busy]  WITH prrun1;
```

3.4 The result part.

This section calculates the output results. The results are specified by equations, giving the name of the "output variable" on the left side, and the formula on the right side. On the right side we can use the word PROB to refer to the steady-state probability of the given state:

```
/*================= The result part begins ===================*/

/*========  If a terminal is busy, then it is utilized =======*/
<1..$NT> RESULT>> if ( term# == busy ) term#util += PROB;

/*==== Calculate the average length of the queue of the CPU ====*/
RESULT>> quelength = MEAN cpu;
```

4 The MARKMOD Software Tool.

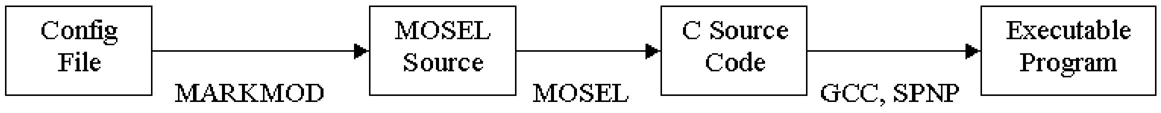

Figure 1.

The software tool introduced here can produce analytical results for the the described model. The system consists of three parts: MARKMOD, MOSEL, SPNP (see Figure 1.) and can be used in any unix-like environment.

The program MARKMOD implements the mentioned Markov-modulated queueing model. The input of MARKMOD is a very simple and well commented parameter file (text file), from which the program generates a MOSEL source file. The generated MOSEL source is very complicated and not so easy to understand, needs deep knowledge of the language. This MOSEL source file is the input for the MOSEL system (i.e. "MOSEL compiler"). The output of the MOSEL system is a C program, which uses the SPNP library system to solve the Markov chains for the model. SPNP is a powerful mathematical library (see [6] and [13]), written in C and containing well-tuned Markov chain solvers. Using SPNP we can generate an executable program from the MOSEL output with a C compiler. This executable program will produce numerical results for the investigated model.

The three systems work together in the backround. The only neccessary user interaction is to edit the configuration file (see the numerical example below) and to issue the MARKMOD command. Then we will get the numerical results automatically.

5 Numerical Examples.

5.1 Random environment.

In this case we consider the model introduced by [7] in section 6. Gaver *et al.* used the terminology of machine interference problem to describe the same mathematical model (i.e. the word “repairmen” was used instead of server and “machine” instead of client). The system contains $N = 5$ clients connected to a server. All the clients and the server are collected into one group affected by a random environment. The random environment has two states denoted by 1 and 2. The clients are homogeneous with job generation rate 0.12 if the environment is in state 1, and 0.06 if it is in state 2. The service rate of the jobs is 1.0 in both environments.
Using the notations of [7] the rate of changes in the environment is $\alpha_1 = 0.5$ and $\alpha_2 = 1.0$
The MARKMOD configuration file of this model is the following:

```
5           # No. of client stations
1           # No. of background processes
2           # No. of states of the bg. processes
05          # No. of stations influenced by the bg. proc.-s, (0 = server)
0.12  0.06  # Job gen. intensities for station 1
0.12  0.06  # Job gen. intensities for station 2
0.12  0.06  # Job gen. intensities for station 3
0.12  0.06  # Job gen. intensities for station 4
0.12  0.06  # Job gen. intensities for station 5
1.00  1.00  # Job run. intensities for station 1
1.00  1.00  # Job run. intensities for station 2
1.00  1.00  # Job run. intensities for station 3
1.00  1.00  # Job run. intensities for station 4
1.00  1.00  # Job run. intensities for station 5
0.5         # Gen. Matrix for bg.pr. 1 (without main diagonal) - line1
1.0         # Gen. Matrix for bg.pr. 1 (without main diagonal) - line2
```

Performance Measures

Length of the server’s queue	0.6418
Utilization of the server	0.4342
Utilization of the clients	0.8716
Response Time	1.4782

Comparing these numerical results to the ones derived from [7] we can see, that the results are the same. The running time of our software tool is about 5 seconds on a 600MHz Pentium PC, that means our software is suitable for such a problems.

5.2 Terminal system with server breakdowns.

In this case we investigate a client-server system with server’s breakdowns. We consider a queueing system consisting of n clients connected with a server. Client i has exponentially distributed request generation and processing times with rate λ_i and μ_i respectively. Let us assume that the server is subject to random breakdowns stopping the server’s service and the client’s work too. The failure-free operation times and repair times are exponentially distributed random variables with parameters α and β. The detailed model description can be found in [1] and [2].
To implement this system in MARKMOD we collect the clients and the server into one group influenced by a background process. The background process has two states: 0 means that the server is operational, 1 means that the system is down.

Input parameters

$n = 4$	$\alpha = 0.25$	$\beta = 0.45$

i	1	2	3	4
λ_i	0.500	0.400	0.300	0.200
μ_i	0.900	0.800	0.600	0.500

The MARKMOD configuration file of this example is shown here:

```
4               # No. of client stations
1               # No. of background processes
2               # No. of states of the bg. processes
04              # No. of stations influenced by the bg. proc.-s, (0 = server)
0.5  0.001      # Job gen. intensities for station 1
0.4  0.001      # Job gen. intensities for station 2
0.3  0.001      # Job gen. intensities for station 3
0.2  0.001      # Job gen. intensities for station 4
0.9  0.001      # Job run. intensities for station 1
0.7  0.001      # Job run. intensities for station 2
0.6  0.001      # Job run. intensities for station 3
0.5  0.001      # Job run. intensities for station 4
0.25            # Gen. Matrix for bg.pr. 1 (without main diagonal) - line1
0.45            # Gen. Matrix for bg.pr. 1 (without main diagonal) - line2
```

Performance measures

Length of the server's queue: 2.187
Utilization of the server: 0.5809
Clients characteristics:

i	1	2	3	4
$Utilization_i$	0.245	0.267	0.301	0.350
$Responsetime_i$	5.022	5.452	5.866	6.462

The correct MARKMOD implementation of the model can be seen with this example, because the performance measures are the same that was in [1] in case 3. (The difference is less than one percent). The running time of the experiment was about 10 seconds, which confirms, that the software tool is really appropriate for such problems.

6 Conclusion.

In this paper a markov-modulated finite source non-homogeneous queueing model has been treated to analyse a client-server communication system. Also a software tool is introduced (based on MOSEL and SPNP) which can be used to calculate analytical results for the model. Furthermore some numerical example illustrate the problem in question and confirms, that the tool is useful for performance analysis of such systems.

7 Acknowledgement.

The authors are very grateful to professor M. Arató for his helpful comments.

Research is partially supported by German-Hungarian Bilateral Intergovernmental Scientific Cooperation, OMFB-DLR No. 21-2000, by the Hungarian Scientific Research Found OTKA T0-34280/2000, by the Research and Development Found FKFP grant 0191/2001 and by the Békési György Scholarship Found.

References

[1] **Almási, B** A Queuing Model for a Non-Homogeneous Terminal System Subject to Breakdowns, *Computers and Mathematics with Applications* Vol. 25, No. 4, 105-111, (1993)

[2] **Almási, B** Response Time for Finite Heterogeneous Nonreliable Queueing Systems, *Computers and Mathematics with Applications* Vol. 31, No. 11, 55-59, (1996).

[3] **Almási, B. and Sztrik, J.** Optimization problems on the performance of a non-reliable terminal system, *Computers and Mathematics with Applications* Vol. 38, No. 3, (1999).

[4] **Bolch, G. and Herold, H.** MOSEL MOdeling Specifocation and Evaluation Language *Technical Report,* University Erlangen (1999).

[5] **Begain, K.; Bolch, G.; Herold, H.** Practical Performance Modeling Kluwer, (2001).

[6] **Ciardo, G.; Muppala, J.K.** Manual for the SPNP Package Version 3.1, Duke University, Durham, NC, USA, (1991).

[7] **Gaver D.P., Jacobs P.A., Latouche G.** Finite birth-and-death models in randomly changing environments, *Advances in Applied Probability* 16, pp. 715-731, (1984).

[8] **Haverkort B.** *Performance of Computer Communication Systems,* John Wiley & Sons, (1998).

[9] **Herold, H.** MOSEL An Universal Language for Modeling Computer, Communication and Manufacturing Systems. *Phd Dissertation,* University Erlangen, (2000).

[10] **Kameda, H.** A finite-source queue with different customers. J. ACM 29, 478-491, (1982).

[11] **Sztrik J., Kouvatsos D.D.** Asymptotic Analysis of a Heterogeneous Multiprocessor System in a Randomly Changing Environment, *IEEE Transactions on Software Engineering* 17, pp. 1069-1075, (1991).

[12] **Sztrik J., Moeller O.** A tool for simulation of Markov-modulated finite-source queueing systems, *Proceedings of Messung Modellirung und Berwertung (MMB99), Trier, Germany* pp. 109-114, (1999).

[13] **Trivedi, K.S.; Ciardo, G.** A Decomposition Approach for Stochastic Reward Net Models, Duke University, Durham, NC, USA, (1991).