



6th International Conference **APLIMAT 2007**

Faculty of Mechanical Engineering - Slovak University of Technology in Bratislava

Session: Modeling and Simulation

A DISTRIBUTED TOOL TO SOLVE MARKOV CHAINS

BYLINA Jarosław, (PL), BYLINA Beata, (PL)

Abstract. The paper presents a tool for solving complicated Markov models. Two issues are concerned: efficient generation of a Markov chain infinitesimal generator from a queuing model description and numerical finding of steady state probabilities with the use of GMRES method. The tool is a distributed application for a computer cluster. The results of the application are presented for a simple model.

Key words and phrases. queuing models, Markov chains, distributed algorithms, projection methods, GMRES.

Mathematics Subject Classification. Primary 68W15, 60J22; Secondary 65F50.

1 Introduction

The last years brought enormous development of telecommunication and computer networks. Networks became complex structures which are not only used to transfer data but also to store them, to process them and to make them available to users very fast. The users themselves become more and more demanding, and their requirements are more and more differentiated. Thus, it is necessary to guarantee users the required conditions of the network usage – by a good project which can be done with the use of network modelling during the designing phase.

Markovian queuing models are a special case of queuing models. Such a Markovian model can be described with a (continuous- [1, 7] or discrete-time [5, 10]) Markov chain. Markov chains are ‘memory-less’, so the more accurate model we need, the more states appear in the Markov chain and the number of states grows exponentially (this growth is sometimes called ‘a state explosion’).

On the other hand there are many ways to compute needed characteristics of a model described by a Markov chain [2, 6] – e.g. steady states probabilities, which is the most interesting problem for the authors. Because of the state explosion (mentioned above) computing those probabilities is a very time- and space-consuming task. Presented here is a distributed application, working on clusters of many machines.

2 The Distributed Application

The application presented in this paper is intended for generating a Markov chain transition rate matrix \mathbf{Q} (for a continuous-time Markov chain; for a discrete-time Markov chain a stochastic matrix \mathbf{P} is generated and then converted to \mathbf{Q} [9]) from a given model and for finding steady states probabilities vector \mathbf{p} from

$$\mathbf{p}\mathbf{Q} = \mathbf{0}, \quad (1)$$

where $\mathbf{p} \geq \mathbf{0}$ and $\|\mathbf{p}\|_1 = 1$. Thus it is composed of two parts – a generating part and a computing part.

The application is written in ANSI C [8] and compiled with gcc under the Linux operating system. It is designed to work on a network (cluster or grid) of computers connected with the TCP/IP protocol.

The description of the analysed model should be prepared in the form of a (rather short) source C file. The main part of that file is a function which describes states adjacency (that is, which state goes after which).

2.1 Initialization

After the compilation the application consists of two executable files: ‘master’ and ‘slave’. To start the application, one is to run the master on a chosen machine and the slave on all the others. All the processes open TCP/IP sockets to listen to other machines and then each slave connects to the master which sends data needed to continue. The master also sends all slaves’ addresses to each of them – thus a logical network of slaves is established. Next, the slaves start generating the matrix \mathbf{Q} .

2.2 Distributed Generation

A distributed algorithm for the generation of the transition rate matrix \mathbf{Q} of a Markov chain from a queuing model was described in [4]. Its idea is following.

First, the slaves receive data needed for the generation from the master. Among various data they get descriptions of ‘pools of states’. These pools are separate sets of chain states (represented by state vectors) and the pools are mapped to the slaves one-to-one, containing together all the possible states. The pools cannot be given by enumeration – because before the generation they are unknown. They must be described with simple statements (which are simple to check, like ‘if the first element of the state vector is even, then the state belongs to the pool number one’ or ‘if the second element of the state vector is zero, then the state belongs to the pool number three’) and they should satisfy the following conditions: (a) all the pools should have roughly the same size; and (b) the states from different pools should have possibly little transitions among them.

The condition (a) is needed because when distributed computations are conducted the overall time needed to accomplish them strongly depends on the size of the data stored in the most loaded machine. The states from each pool are stored in one machine, hence the condition (a). Moreover, the transitions among the states from different pools (that is: different machines)

$$\mathbf{Q} = \begin{pmatrix} \mathbf{Q}_0 \\ \hline \mathbf{Q}_1 \\ \hline \vdots \\ \hline \mathbf{Q}_{p-2} \\ \hline \mathbf{Q}_{p-1} \end{pmatrix}$$

Figure 1: The division of the matrix \mathbf{Q} in the distributed algorithm

mean messages between different slaves (see below) – that means more communication and the communication is the most time-consuming task in distributed algorithms.

After these jobs, each of the slaves generates the states adjacent to this slave’s own states (belong to its pool). Obviously, there will appear some states from other pools – they must be sent to its owners (and it is the cause for the condition (b) above). During those tasks, the master supervises the slaves and when there are no states to generate, it receives from them the numbers of their pools and sends them information about common indexing of the states.

2.3 Distributed GMRES

The algorithm GMRES was implemented as a distributed master-slave algorithm [3]. All vector operations (there are a lot of them in GMRES) are conducted in the master machine, vectorized with the use of the BLAS (Basic Linear Algebra Subroutines) library [12] in the ATLAS (Automatically Tuned Linear Algebra Software) implementation [11].

A main operation in the GMRES algorithm is a matrix-vector multiplication. This is the operation which is distributed: the master sends a part of a vector to each of the slaves; then the slaves multiply the part of the matrix \mathbf{Q} by a part of a vector and send results to the master; at last the master combine them in a complete result vector.

3 A Working Example and Tests

In this paper a simple model was chosen for the application testing – namely an abstract queuing model of an ineractive computer system [9] – shown in Figure 2. The model consists of a central processing unit \mathbf{C} , a secondary memory device \mathbf{S} , a filing device \mathbf{H} and N terminals \mathbf{T} which correspond to N tasks circulating among the service stations \mathbf{C} , \mathbf{T} , \mathbf{S} and \mathbf{H} . Service times in all the service stations are exponential (with intensities $\mu_C, \mu_T, \mu_S, \mu_H$, respectively). The state of such a model can be described with a vector (n_C, n_T, n_S, n_H) where all its elements are numbers of tasks staying in the corresponding service stations.

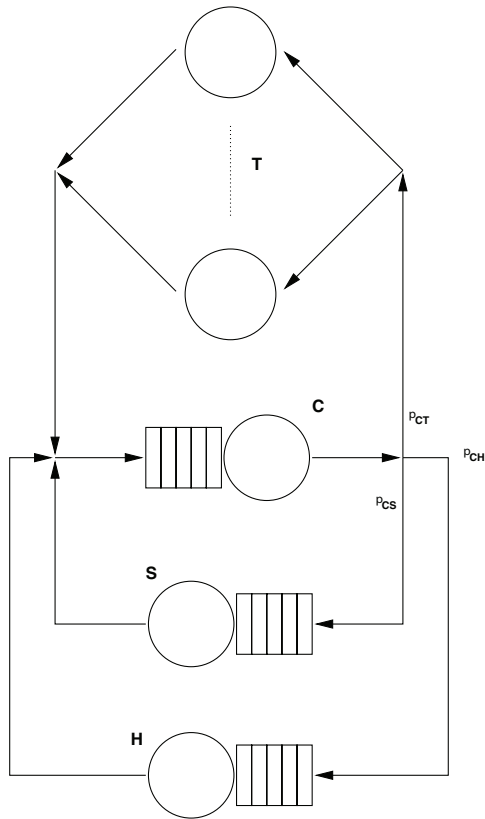


Figure 2: A simple queuing model

The model has to be described in the language C with the use of a short function which for any given state vector returns all the possible adjacent states.

The behaviour of the application was tested in a distributed environment consisting of five (one master and up to four slaves) machines (with Intel Pentium4 2.80GHz processors and 512MB RAM) connected with Ethernet 1Gb/s with Debian GNU/Linux operating system.

Figure 3 presents (for various N) the times of the generation of the matrix \mathbf{Q} (solid lines with crosses), the solving times (dashed lines) and the total working times (black squares) as functions of number of the slave machines.

As one can see from the figures, the generation times are long for big models and the solving times are even longer when a single slave is used. In Figure 3(c) the solving times for less than four slaves is so long that it cannot be shown in the plot. In Figure 3(d) either the solving times and the generation times cannot be shown for less than four slaves. However, when more slaves are used the computations speed up significantly. Not only additional processors are the cause of that, but first of all additional RAM – because both the parts (generation and solving) require quite a lot of memory, and when there is not enough real RAM, the system starts using virtual RAM (which is much slower).

For little models additional slaves do not speed up the computations – moreover they can slow them down because there is relatively more communication between slaves than for bigger models.

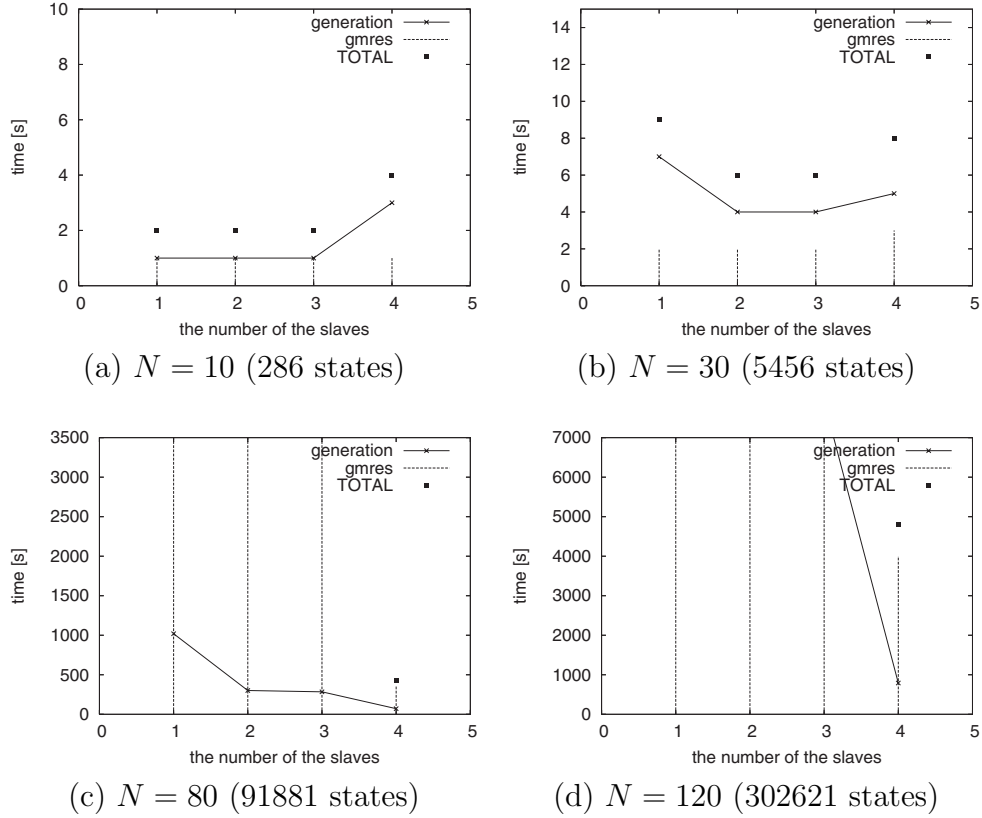


Figure 3: Working times of the application for the presented model with various parameters N

4 Conclusion

The figures show that when the tool described above is used, the additional computers can speed up the generation if the number of the states is sufficiently large. However, the solving times decrease only for big models and if enough machines are used.

The experiments show that the described tool can speed up the generation and, to some degree, solving queuing models. The application can handle bigger models (even more than 10^6 states) than shown in the paper. Moreover, the application can be more efficient when the generated Markov chain is structured and the statements describing pools (described in the section 2.2) are carefully chosen.

References

- [1] AKIMARU, H., KAWASHIMA, K.: *Teletraffic: theory and application*, Springer-Verlag, 1993.
- [2] BAIOCCHI, A., MELAZZI, N. B., LISTANTI, M., ROVERI, A., WINKLER, R.: *Loss Performance Analysis of a ATM Multiplexer Loaded with High-Speed ON-OFF Sources*, IEEE-JSAC, vol. 9, no. 3, April 1991.

- [3] BYLINA, J.: *A distributed approach to solve large Markov chains*, Proceedings from EuroNGI Workshop: New Trends in Modeling, Quantitative Methods and Measurements, Jacek Skalmierski Computer Studio, Gliwice, 2004, pp. 145–154.
- [4] BYLINA, J., BYLINA, B.: *Distributed generation of Markov chains infinitesimal generators with the use of the low level network interface*, Proceedings of 4rd International Conference Aplimat 2005, part II, Bratislava, 2005, pp. 257–262.
- [5] DOMANSKA, J., CZACHORSKI, T.: *Wplyw samopodobnej natury ruchu na zachowanie mechanizmu ciekącego wiadra*, Studia Informatica, Vol. 24, Nr 2A (53), pp. 199–212.
- [6] GUAN, L., WOODWARD, M. E., AWAN, I. U.: *Stochastic Approach for Modeling Multi-Class Congestion Control Mechanisms Based on RED in TCP/IP Networks*, EU Network of Excellence (NoE) Euro-NGI, P36/1.
- [7] JAIN, R.: *The Art of Computer Systems Performance Analysis. Techniques for Experimental Design, Measurement, Simulation and Modeling*, John Wiley & Sons, New York, 1991.
- [8] KERNIGHAN, B. W., RITCHIE, D. M.: *The C Programming Language, Second Edition*, Prentice Hall Inc., 1988.
- [9] STEWART, W.: *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Chichester, West Sussex, 1994.
- [10] WOODWARD, M. E.: *Communication and computer networks. Modelling with discrete-time queues*, Pentech Press, London, 1993.
- [11] <http://www.netlib.org/atlas/>
- [12] <http://www.netlib.org/blas/>

Current address

dr Jarosław Bylina, dr Beata Bylina

Department of Computer Science, Institute of Mathematics

Maria Curie-Skłodowska University

Pl. M. Curie-Skłodowskiej 1, 20–031 Lublin, Poland

jmbylina@hektor.umcs.lublin.pl

beatas@hektor.umcs.lublin.pl