Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 27/11/2026 08:05:02

Annales UMCS Informatica AI 2 (2004) 25-29

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

http://www.annales.umcs.lublin.pl/

Distributed generation of Markov chains infinitesimal generator matrices for queuing network models

Jarosław Bylina*

Department of Computer Science, Marie Curie-Skłodowska University,
pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland

Abstract

In this paper we want to present problems connected with the generation and storing of huge sparse matrices, arising during modelling the queuing networks with the use of Markov chains. We also want to present an algorithm for distributed generation of such matrices.

1. Introduction

A distributed algorithm for solving huge and sparse linear systems that appear in the Markov chain analysis of queuing network models was presented in [1]. That algorithm requires the linear system coefficients matrix **Q** to be distributed (nearly) evenly among machines before the start of the computations.

One solution to that problem is to generate the matrix **Q** on one computer and then distribute it among others. However, this approach is rather time-consuming – because we use only one machine of the whole cluster/grid and the distribution after the matrix generation is expensive from the communicational point of view – and space-consuming – because we try to store the whole huge matrix on one machine which can be expensive or even impossible. One of the advantages of the algorithm described in [1] was the starting distribution of the matrix **Q** (during its generation) to use the space on the machines in the best manner.

We achieve better results when we design a distributed algorithm for generating the matrix **Q** after which there will be a respective part of the matrix on each of the machines. In our algorithm each computer will be responsible for generating its own part of the matrix. Of course, there will be some communication but it will be reduced to the minimum.

* *E-mail address*: jmbylina@hektor.umcs.lublin.pl

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 27/11/2026 08:05:02

2. Markov chains in the queuing networks analysis

We designed our algorithm to generate the *transition rate matrix* (also known as the *infinitesimal generator matrix*) for queuing networks.

A queuing network is a system of connected *service stations* (shown in diagrams as circles with optional queues) with *customers* moving instantly among them. A customer spends some random time being served within a service station, then it travels to a next station according to *routing probabilities* which determines chances for a customer which path it will take. An example of a queuing network is presented in Figure 1.

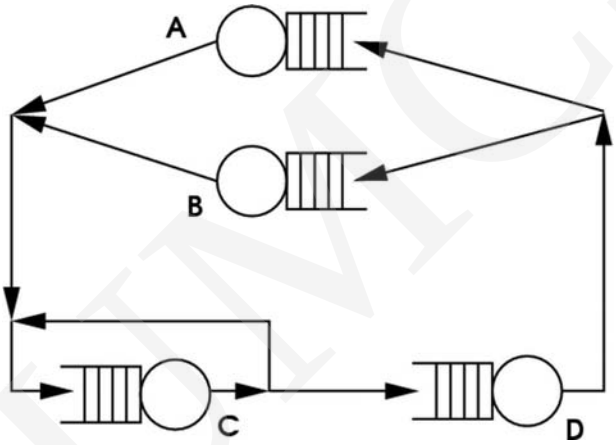

Fig. 1. A scheme of a queuing network example

To start investigating the behaviour of such a system as a Markov chain we have to choose a representation for the set of *states*. The most common way is to represent each system state as a vector whose components describe completely the states of all elements of a queuing network. For the queuing network in Figure 1 with the constant number N of customers and with the exponential (that is 'without memory') distribution of the service times we can define the system state with a four-element vector (a,b,c,d) where a, b, c and d are the numbers of customers waiting and being served in service stations A, B, C and D, respectively, and where $a+b+c+d = N$.

Next, we have to enumerate all potential transitions among states and define for them the *transition rates* q_{ij} (independent of time for homogeneous Markov chain, most interesting for us):

$$q_{ij} = \lim_{\Delta t \to 0} \frac{p_{ij}(\Delta t)}{\Delta t} \quad \text{for} \quad i \neq j \,,$$

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 27/11/2026 08:05:02

$$q_{ii} = -\sum_{j \neq i} q_{ij} ,$$

where $p_{ij}(\Delta t)$ is the probability that if the system is in the state i the transition occurs to the state j in the time Δt. This is how the transition rate matrix $\mathbf{Q} = (q_{ij})$ is created.

The last step (being performed by the algorithm from [1]) in the analysis is to compute probability vector $\pi(t)$ whose components $\pi_i(t)$ are the probabilities that the system is in the state i at the time moment t. We are interested in finding the long-run (independent of time) probability vector $\pi = (\pi_1, ..., \pi_n)$ from:

$$\pi \mathbf{Q} = 0 , \quad \pi \geq 0 , \quad \sum_i \pi_i = 1 .$$

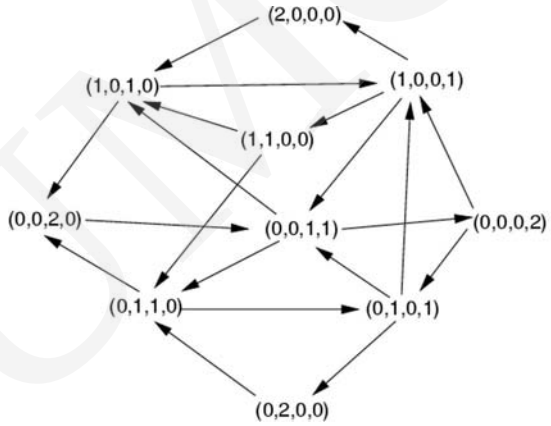

Fig. 2. A transition graph of a queuing network

3. The generation of the transition rate matrix

There are various manners to generate the transition rate matrix. There are methods – fast but not very general – based on indexing functions [2,3], probabilistic methods (giving incorrect transition matrices with a non-zero probability) [4] and others. We are interested in one of the most general methods – the Breadth First Searching algorithm (BFS), which allows us to enumerate all states, to number them, to enumerate all potential transition among states and to compute the transition rates.

The BFS algorithm is an algorithm to traverse all the edges of a directed graph. We start with a list having a single (arbitrary) vertex (that is: a state vector) and we investigate all edges with the given vertex as a starting point. For each of these edges we add to the list its ending point – but only if it is not in the list yet. Next we go to the next vertex in the list and so on, while there are vertices not yet traversed in the list.

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 27/11/2026 08:05:02

The algorithm described above can be used to generate the transition rate matrix for a queuing model. In this case it does not search the graph but it generates the transition graph (such a graph for the queuing network in Figure 1 for $N = 2$ is presented in Figure 2) and the transition rates. Such a version of the BFS algorithm is presented below.

1. Initialize L as a numbered list that contains only one, arbitrary state vector w (a graph vertex) of the transition graph.
2. Initialize X as an empty list.
3. For each vector v representing the state that is allowed as a next state after the state w:
   a) if v is not a member of your vector pool and is not a member the list X then attaches v to X else if v is a member of your vector pool and is not a member of the list L then attach v to L,
   b) compute the transition rate from w to v and remember it as Q[ind(w, L), ind(v, L)] or as QX[ind(w, L), ind(v, X)] – it depends on the list in which the v is (where ind(a, B) is an index of a in the list B).
4. If w is not the last element of L then assign the next element of L to w and go to 3.
5. Send the elements of X (that have not been sent yet) to the master.
6. Wait for the answer from the master.
7. If the answer will not be the 'ending signal', then attach the received elements (state vectors generated by other slaves and received by the master) to L and go to 4.
8. Together with the ending signal the master sends the data (mainly sizes and indices) that the slave uses to compose its own part of the matrix Q from QL and QX.

The vector pool is a key idea for our algorithm. The vector pools are subsets of the state space, their intersections are empty and their union is the whole state space. Moreover, the pools should be of almost the same count because they are assigned to the slaves 'one-to-one' and the algorithm described in [1] works best in this case. The problem is that we have to know the division into the pools *before* our distributed algorithm starts.

Our method to solve this problem is to describe each of the vector pools with simple conditions, which can be distributed among the slaves before the actual algorithm starts. Exemplary conditions for our graph in Figure 2 could be 'the first element of the vector is zero'/'the first element of the vector is not zero'

Pobrane z czasopisma Annales AI- Informatica **http://ai.annales.umcs.pl**
Data: 27/11/2026 08:05:02

(one pool of 6 and the other of 4 elements) or 'the 'one of the vector elements is 2/'no vector element is 2 and the first one is zero'/'no vector element is 2 and the first one is not zero' (4/3/3). The slaves can easily check if the generated state vectors are in their pools with such conditions, but on the other hand, we should choose these conditions very carefully – to minimize differencies between quantities of the vector pools.

5. Conclusion

The presented above (section 4) distributed algorithm for generating the transition rate matrix can be easily implemented with the language Ada [5] or C with the use of MPI [6]. It can improve the distributed algorithm for solving sparse linear systems from [1] making it a useful tool for solving the Markovian models appearing during modelling various networks with queuing networks. The problem which must be solved separately for each model is the choice of a suitable division into vector pools. We are going to investigate this question further.

References

[1] Bylina J., *Distributed solving of Markov chains for computer network models*, Annales Informatica, UMCS Lublin, 1 (2003) 15.

[2] Czachórski T., Niemiec M., Pecka P., *Analizator Modeli Równoległych AMOR*, IITiS PAN (1995), in Polish.

[3] Niemiec M., Pecka P., *Metoda odwzorowywania stanów w przestrzeń liczb naturalnych w markowowskich modeli systemów komputerowych*, Archiwum Informatyki Teoretycznej i Stosowanej, 9(1-4) (1997) 131, in Polish.

[4] Knottenbelt W., *Generalised Markovian analysis of timed transition systems*, Master's thesis, University of Cape Town South Africa, (1996).

[5] Stpiczyński P., *Implementacja rozproszonej pamięci wspólnej przy pomocy zdalnego wywołania procedur*, Obliczenia naukowe – Wybrane problemy, Polskie Towarzystwo Informatyczne, Lublin, (2003), 47, in Polish.

[6] Bylina B., *Komunikacja w MPI*, Informatyka Stosowana S2/2001, V Lubelskie Akademickie Forum Informatyczne, Kazimierz Dolny, (2001) 31, in Polish.