STUDIA INFORMATICA 2008
Volume 29 Number 4B (81)

Jarosław BYLINA, Beata BYLINA
Maria Curie-Skłodowska University, Institute of Mathematics

ANALYSIS OF AN ALGORITHM OF DISTRIBUTED GENERATION OF MATRICES FOR MARKOVIAN MODELS OF A CONGESTION CONTROL MECHANISM

Summary. We consider the parallel generation of matrices corresponding to models of congestion control mechanisms' behavior. We develop a piece of software for a cluster architecture and analyze its performance times, amount of communication, each processor's load. The resulting application is scalable and also produces a substantial speedup and efficiency.

Keywords: Markovian models, queuing models, parallel algorithms, distributed algorithms

ANALIZA ALGORYTMU ROZPROSZONEJ GENERACJI MACIERZY DLA MARKOWSKICH MODELI MECHANIZMU KONTROLI ZATŁOCZENIA

Streszczenie. Rozważamy równoległą generację macierzy odpowiadających modelom zachowania mechanizmów kontroli zatłoczenia. Rozwijamy oprogramowanie dla architektury klastrowej i analizujemy czasy jego działania, ilość komunikacji, obciążenie każdego z procesorów. Otrzymana aplikacja jest skalowalna i daje znaczące przyśpieszenie oraz efektywność.

Słowa kluczowe: modele Markowa, modele kolejkowe, algorytmy równoległe, algorytmy rozproszone

1. Introduction

Stochastic processes – and Markov chains particularly – provide a formal way of capturing and analyzing the dynamic behavior of computer and communication systems. Such

models can be specified using queuing networks [11]. A vital task in modeling with Markov chains is finding stationary probability distribution of the states of the model/chain.

In the implementation practice, the act of finding the stationary probability distribution vector consists of two stages: the generation of the transition rate matrix **Q** and the numerical solution of a linear system. Due to the very large number N of states typical for many real-world applications, there has been an increasing interest in developing parallel algorithms for handling Markov chains.

There is a big body of previous work on the efficient parallel calculation of steady-state probabilities in large Markov chains [1, 2, 3, 9]. In those papers the matrix **Q** is generated in one machine (one network node) and then is distributed among p processors.

The algorithm analyzed here was described in details in [4, 5, 7, 8]. In this paper we adopt the algorithm to work on computer cluster and investigate its behavior in this environment.

This paper is an improved end expanded version of our earlier work [6].

The rest of this paper is organized as follows. Section 2 presents some congestion control mechanisms. Section 3 shows some details of the described algorithm. Numerical tests are presented in Section 4 and some conclusions are in Section 5.

2. Congestion control models

The simplest mechanism of congestion control in such a buffer is a *tail-drop* mechanism. It is a passive mechanism, that is, it does not make any decision, as far as the moment of dropping packets is concerned – they are always dropped when there is no room for new packets. Moreover, in the tail-drop mechanism the packet, that just arrived, is dropped.

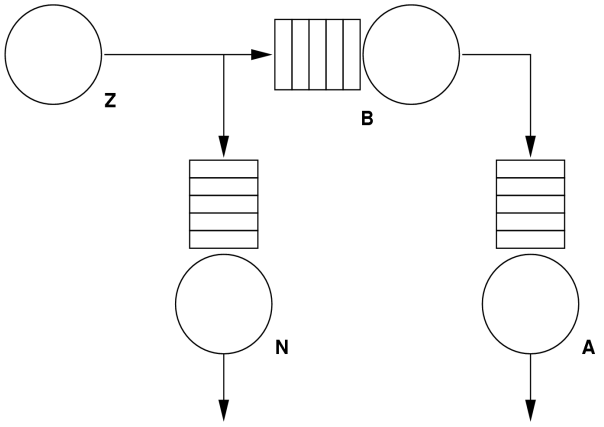

Fig. 1. A Markovian queuing model of the tail-drop mechanism
Rys. 1. Markowski model kolejkowy mechanizmu *tail-drop*

A Markovian queuing model for such a mechanism is presented in the Figure 1. It consists of: a source **Z** (which sends packets with variable rates), a service station **B** (which corresponds to a router's buffer) and two auxiliary service stations **A** and **N** (which correspond to confirmations and rejections returning to the source **Z**, respectively).

The rate of the source **Z** is not constant but it increases (not above a given maximum) as confirmations leave the station **A** and it decreases (not below a given minimum) as rejections leave the station **N** (both events represent reaching the source by the information about the packet's fate).

The states of such a model can be written as vectors of numbers. In our example it would be as four numbers (l, b, a, n), where l is an integer between 1 and l_{max} and it means the current relative intensity of the source **Z** and b, a and n are integers between 0 and b_{max}, a_{max}, n_{max} (respectively) and they mean the number of packets waiting or being processed currently in the stations **B**, **A**, **N** (respectively).

3. Generating a distributed matrix for parallel processing

We briefly present the idea of an algorithm for generating the matrix **Q**. It is based on the fact that the transition rate matrix **Q** can be seen as a directed graph (the transition graph) and our algorithm is a parallelization of the well-known BFS (*Breadth-First Search*) algorithm [10] with a special division of the graph among computing nodes. The algorithm is a *master-slave* algorithm that means that we need one process (and one processor/machine) as a supervisor (the *master*) and some processes (each on its own processor/machine) as workers (the *slaves*).

3.1. Pools of states

A key concept for the distributed algorithm is a *pool of states*. All the possible states are divided into the separate sets (that is pools) – so that each state belongs exactly to one pool. Moreover, there is an exact correspondence: one pool – one slave. To limit the communication we need to design our division so as there are the smallest number of transitions between different pools – because each transition between states of different pools causes necessity of some additional communications between respective processors.

Figure 2 shows a sample division of a 15 x 15 matrix **Q** among four slaves according to a division of the states set into four pools. In Figure 2, $\mathbf{S}_i$ specifies a slave with the number i – which can run on a separate processor (or machine) or as ith thread on a single processor.

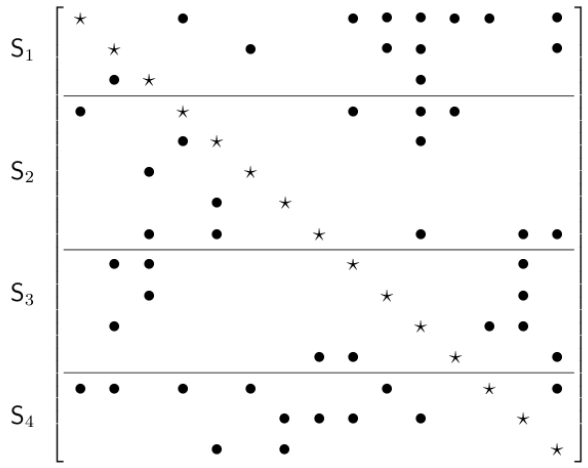

Fig. 2. A (row-striped) division of a sample matrix **Q** (15 × 15) among four slaves. The pools are uneven, but their sizes (and numbers of positive values) are close to each other. The matrix is sparse, so bullets denote only positive values, stars are negative values (obtained so that the sum of each row is zero) and zero elements (not stored) are not indicated

Rys. 2. Podział (wierszowy) przykładowej macierzy **Q** (o rozmiarze 15 × 15) pomiędzy cztery procesy *slave*. Pule nie są równe, ale ich rozmiary (i liczby wartości dodatnich) są zbliżone. Macierz jest rzadka, więc koła oznaczają tylko wartości dodatnie, gwiazdki – ujemne (otrzymane tak, by suma każdego wiersza się zerowała), natomiast elementy zerowe (nieprzechowywane) nie są zaznaczane

In this example the load of data for each processor is presented in Table 1.

Table 1

Slaves' load of data

slave	number of rows	number of non-zeros
1	3	13
2	5	13
3	4	11
4	3	12

3.2. Parallel algorithm for generating matrix Q

The matrix **Q** (N x N) can be represented as a directed graph $G = (W, K)$, where W is a set of vertices and K is a set of (directed) edges. The vertices (that is elements of W) can be identified with the states of the model and all the edges are labeled with transition rates between respective states.

For obtaining the matrix **Q** a modified BFS algorithm [10] is used. BFS allows to travel through (and thus: find) all vertices (that is: states) and edges (that is: transitions) with their labels (that is: transition rates) of a connected graph (that is: a transition rate matrix).

Adapting the algorithm to work in a distributed environment lies in running p slave processes and one master process. Each slave process is responsible for generating its own part of the matrix **Q**, occasionally communicating with other slaves. The master process does not take direct part in generating **Q**, but controls activities of the slaves – in particular

checking whether the slaves finished their job. The details of the algorithm are described in [6] (and other papers).

4. Numerical tests

This section describes numerical results that demonstrate the applicability, scalability and capacity of our technique. We analysis size of pools an number of communications for considered model of congestion control.

4.1. Computational environment

Experiments were carried out in a cluster environment (part of CLUSTERIX – a cluster of Linux machines consisting of more than 800 machines distributed all over Poland and connected with a fast optical network; one of the fastest European distributed supercomputers [12]) dedicated for computing. We used up to 9 machines (one master and eight slaves), each with two 64-bit processors Intel Itanium 2 1.4 GHz and 4 GB RAM. We tried to use the cluster nodes when they were idle – in order to assure credible performance times.

4.2. Parameters of the model

We used – as a model for tests – the model of the tail-drop mechanism described in Section 2. We generated the matrix for the following values of the model parameters: $l_{max} = 15$, $b_{max} = 15$, $a_{max} = 25$, $n_{max} = 25$. They gave the matrix of the size $N = 161\,250$, and our matrix was not symmetric.

Table 2

The division into pools for p slaves (N_i – number of rows in the ith slave)

p	N_1	N_2	N_3	N_4	N_5	N_6	N_7	N_8
1	161 250							
2	80 604	80 646						
3	53 882	53 875	53 493					
4	40 297	40 318	40 307	40 328				
5	32 571	32 168	32 159	32 167	32 185			
6	26 738	27 132	27 123	27 144	26 743	26 370		
7	22 877	23 257	23 257	23 247	23 250	22 868	22 494	
8	20 159	20 156	20 140	20 144	20 138	20 162	20 167	20 184

To get a roughly even division into the pools we used the following rule: the state described by the vector (l, b, a, n) – see Section 2 – belongs to the pool number

$i = (b+a) \bmod p$ (and thus to the slave $\mathbf{S}_i$). Hence, the division into the pools (Table 2) was almost completely even, so there was a good load balance in our tests.

4.3. Results of the experiment

The run-time for p slaves (T_p), the speedup ($S_p = T_1/T_p$) and the efficiency ($E_p = S_p/p$) for our tests are presented in Figures 3, 4, 5.

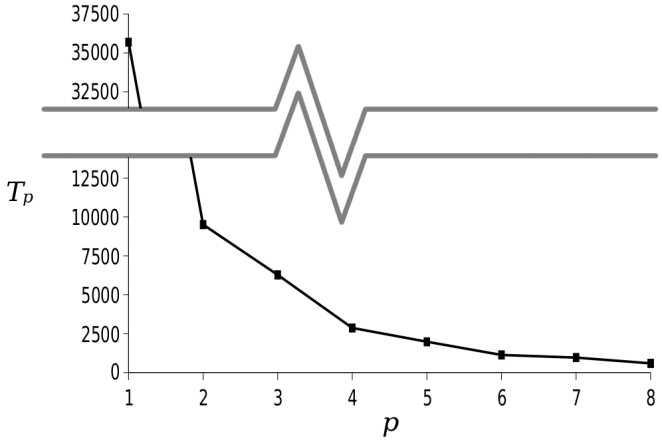

Fig. 3. The run-time T_p of the algorithm for p slaves
Rys. 3. Czas działania T_p algorytmu dla p procesów *slave*

It is worth noting that although the speedup and efficiency seems to be too high (against Amdahl's law), it is caused not only by employing more processors but also by employing more RAM and cache (which is very needed in our generation) and (what is perhaps also important) by division of used data structures between machines, what makes them much more effective.

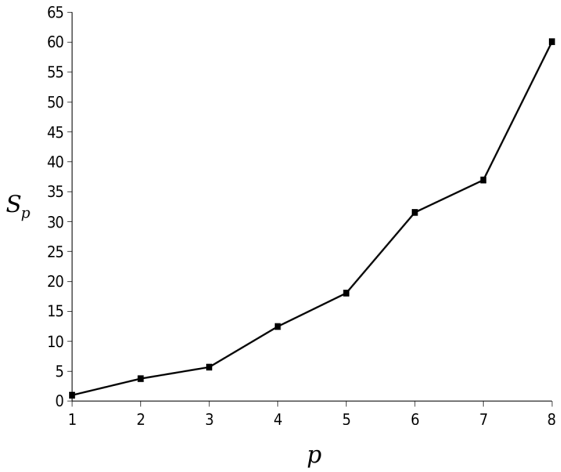

Fig. 4. The speedup S_p of the algorithm for p slaves
Rys. 4. Przyspieszenie S_p algorytmu dla p procesów *slave*

Amdahl's law is a rule about expected speedup of parallelized algorithm relative to its sequential version, under the assumption that the problem size remains the same when

parallelized. However, it does not take into account such phenomena as cache aggregation (and certainly cache is heavily used here) which can explain such a high speedup in our case.

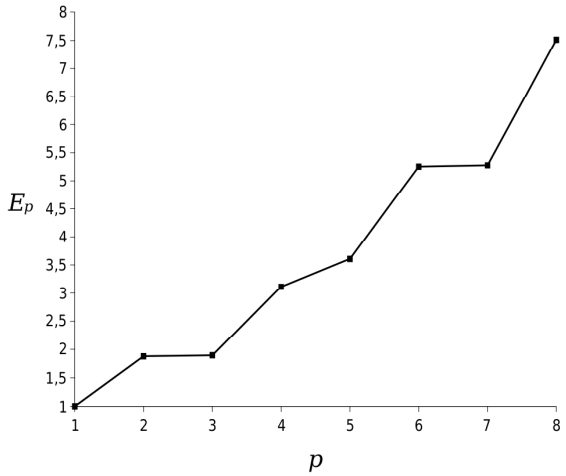

Fig. 5. The efficiency E_p of the algorithm for p slaves
Rys. 5. Efektywność E_p algorytmu dla p procesów slave

Table 3

The scalability of the algorithm

#	l_{max}	b_{max}	n_{max}	a_{max}	N	p	T_p [s]
1	5	10	10	10	6 455	1	4
	5	10	10	10	6 455	2	9
2	10	10	10	10	13 011	1	20
	10	10	10	10	13 011	3	84
	10	10	10	10	13 011	4	104
3	15	10	20	20	72 465	1	6 712
	15	10	20	20	72 465	3	619
	15	10	20	20	72 465	4	219
4	35	35	35	35	1 620 128	8	17 859
	35	35	35	35	1 620 128	17	13 233

Scalability of this application we can investigate in Table 3. For matrices of small sizes (as numbers 1 and 2) there is no scalability, but for large matrices the algorithm is scalable and the larger the matrix, the better the scalability.

5. Conclusions

We have developed a parallel (distributed) and scalable algorithm that computes the matrix **Q** needed for further investigations of the behavior of the Markovian models of various systems – as congestion control in networks. Możemy rozszerzyć to na dowolny model Markowa.

The numerical experiments on a parallel system have been carried out in order to as-sess the effectiveness and scalability of the distributed algorithm on a computer clus-ter. The numerical tests indicate that the good scalability is possible provided that the sufficient amount of work per processor is provided (for small sizes of the matrix the scalability is not as good as presented here). Parallelization based on the graph parti-tioning is usually effective and that was also in shown case.

Our results suggest an important area for future research – writing numerical algo-rithms (which find probability vectors) for huge matrices modeling Markov chains distributed in row-striped manner among many processor (as output by our algorithm).

Acknowledgments

This work was partially supported by Marie Curie-Sklodowska University in Lublin within the project *Równoległe algorytmy generacji i rozwiązywania mechanizmów kontroli przeciążenia w protokole TCP modelowanych przy użyciu łańcuchów Markow*a.

REFERENCES

1. Benzi M., Tùma M.: A parallel solver for large-scale Markov chains. Applied Numerical Mathematics 41, 2002, p. 135÷153.
2. Bradley J. T., Dingle N. J., Knottenbelt W. J., Wilson H. J.: Hypergraph based parallel computation of passage time densities in large semi-Markov processes. Proceedings of the Fourth International Conference on the Numerical Solution of Markov Chains (NSNC '03), Urbana-Champaing, IL, September 2003, p. 99÷120.
3. Buchholz P., Fischer M., Kemper P.: Distrbuted steady state analysis using Kronecker algebra. Proceedings of the Third International Conference on the Numerical Solution of Markov Chains (NSNC '99), Zaragoza, Spain, September 1999, p. 76÷95.
4. Bylina J.: A distributed approach to solve large Markov chains. Proceedings from EuroNGI Workshop: New Trends in Modeling, Quantitive Methods and Measurements. Jacek Skalmierski Computer Studio, Gliwice 2004, p. 145÷154.
5. Bylina J., Bylina B.: A Markovian model of the RED mechanism solved with a cluster of computers. Annales UMCS Informatica 5, 2006, p. 19÷27.

6. Bylina J., Bylina B.: Development of a distributed algorithm of matrix generation for Markovian models of congestion control and its performance analysis on a computer cluster. Contemporary Aspects of Computer Networks, t. 2, WKŁ Warszawa 2008, p. 251÷258.
7. Bylina J., Bylina B.: Distributed generation of Markov chains infinitesimal generators with the use of the low level network interface. Proceedings of 4th International Conference Aplimat 2005, part II, p. 257÷262.
8. Bylina J., Bylina B., Czachórski T.: Rozproszona aplikacja do rozwiązywania markowskich modeli mechanizmów sieciowych. Nowe technologie sieci komputerowych, t. 1, WKŁ, Warszawa 2006, p. 331÷338.
9. Dingle N. J., Harrison P. G., Knottenbelt W. J.: Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models. Journal of Parallel and Distributed Computing 64, 2004, p. 908÷920.
10. Knottenbelt W.: Distributed disk-based solution techniques for large Markov models. Proceedings of the Third International Conference on the Numerical Solution of Markov Chains (NSNC '99), Zaragoza, Spain, September 1999.
11. Ng Chee Hock: Queuing Modelling Fundamentals. Wiley, New York 1996.
12. National CLUSTER of LinuX Systems [@:] http://clusterix.pcz.pl/ .

Recenzent: Prof. dr hab. inż. Tadeusz Czachórski

Wpłynęło do Redakcji 25 września 2008 r.

Omówienie

Do przewidywania pracy mechanizmów stosowanych do kontroli zatłoczenia w sieciach komputerowych stosowane są między innymi modele Markowa. Modele te pozwalają na przykład na znalezienie średniego rozmiaru kolejki czy rozkładu długości kolejki w buforze. W praktyce implementacyjnej etap znajdowania wektora prawdopodobieństwa stanów w łańcuchu Markowa składa się z dwóch części: generacji macierzy **Q** i numerycznego rozwiązania równania (liniowego lub różniczkowego).

W tej pracy poruszana jest tylko kwestia związana z generacją macierzy **Q**. Z powodu bardzo dużej liczby stanów (a co za tym idzie dużego rozmiaru macierzy **Q**) w wielu rzeczywistych zastosowaniach wzrasta zainteresowanie rozwojem równoległych algorytmów do obliczania łańcuchów Markowa [1, 2, 3, 4, 7, 8, 9, 10]. W prezentowanych pracach

macierz **Q** jest najpierw generowana zwykle na jednym procesorze, a potem następuje jej "dystrybucja", to znaczy podział pomiędzy *p* procesorów. Dopiero następnym krokiem jest numeryczne rozwiązywanie równań, które odbywa się na *p* procesorach. Takie podejście powoduje, że liczony jest czas nie tylko generacji macierzy, ale także jej dystrybucji pomiędzy procesory.

W tej pracy zaprezentowane jest inne podejście, mianowicie każdy fragment macierzy **Q** jest generowany od razu w odpowiednim procesorze (rys. 2). Takie podejście zmniejsza komunikację między procesorami oraz umożliwia równomierne obciążenie pracy wszystkich procesorów, a także umożliwia generowanie macierzy o większym rozmiarze. W zaprezentowanym podejściu użyto techniki *master-slave* do zaprogramowania równoległości na klaster komputerowy. Dodatkowo, kluczowym zagadnieniem jest tutaj pojęcie pul stanów. Zbiór wszystkich stanów łańcucha Markowa jest podzielony na *p* rozłącznych podzbiorów, w miarę równolicznych, suma tych podzbiorów stanowi zbiór wszystkich stanów. Taki podzbiór nazywany jest pulą. W pracy jest analizowana wydajności algorytmu generacji rozproszonej macierzy opisującej model Markowa kontroli zatłoczenia (rys. 1). Badana jest skalowalność (rys. 4), efektywność (rys. 5) i czas wykonania (rys. 3) algorytmu oraz podział łańcucha Markowa opisującego model kontroli zatłoczenia na pule stanów.

Addresses

Jarosław BYLINA: Maria Curie-Skłodowska University, Institute of Mathematics, Plac Marii Curie-Skłodowskiej 5, 20-031 Lublin, Poland, jmbylina@hektor.umcs.lublin.pl
Beata BYLINA: Maria Curie-Skłodowska University, Institute of Mathematics, Plac Marii Curie-Skłodowskiej 5, 20-031 Lublin, Poland, beatas@hektor.umcs.lublin.pl