Annales UMCS Informatica AI 1 (2003) 15-20

Annales UMCS
Informatica
Lublin-Polonia
Sectio AI

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

Distributed solving of Markov chains for computer network models

Jarosław Bylina*

Department of Computer Science, Institute of Mathematics, Maria Curie Skłodowska University, Pl. M. Curie-Skłodowskiej 1, 20-031 Lublin, Poland

Abstract

In this paper a distributed iterative GMRES algorithm for solving huge and sparse linear systems (that appear in the Markov chain analysis of queueing network models) is considered. It is implemented using the MPI standard on a collection of Linux machines and the emphasis is put upon the size of linear systems being solved and possibility of storing huge and sparse matrices as well as huge vectors on distributed systems.

1. Introduction

Queuing network models have been widely used for analysis of real computer networks and many other kinds of networks (as comunication networks, for example).

For queuing networks satsifying some conditions [1], there exist efficient algorithms (called *product-form* algorithms) that can compute the network parameters. In some cases various methods that compute approximate solutions (without sufficient mathematical justification and defined error bounds) can be used. However, the application range of such algorithms is very limited.

In general, it is necessary to adopt a strictly numerical approach to queueing networks. It is always possible to obtain a Markov chain for any queuing network – we can approximate arbitrarily closely any probability distribution with phase-type representation [2]. Additionally, we can include features such as priority queueing, blocking etc. in the Markov chain representation.

Although, in general, when a compilcated network behaviour is to be represented by a Markov chain then the size of the state space becomes huge very fast and there are problems with space and time complexity of algorithms for solving such huge linear systems.

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

2. Model specification

We start with the description of the queuing network model [3, 4]. Such a model consists of *customers* moving among a set of *service stations*. The customers (also called *clients, jobs* can be divided into *classes*, but within a class all customers are identical.

A service station consists of *queues* where customers wait until they are served and *servers*. The queues may have *infinite capacity* (that is sufficiently large to hold any number of customers) or *finite capacity*. In one station there may be any number of identical servers.

The time distribution used most commonly for service distribution is the exponential distribution or - when the latter is insufficient - Coxian distribution (with which we can approximate any distribution arbitrarily closely).

Each station has its own *scheduling discipline* which defines the order in which clients are served (for example FIFO, LIFO, service in random order etc.).

Clients travel from a station to a station according to *routing probabilities* which determines chances for a customer to go from the given station after service to another. The transitions are instantaneous (but we can add a service station on the path of the clients as a delay mechanism).

3. Markov chain analysis of queuing networks

To represent behaviour of a queueing network as a Markov chain first we have to choose a *state space representation*. A state can be represented as a vector whose components described completely the state of each of the elements of the queuing network. For example a state of a station with one queue (storing only one class of the customers) and one server with the exponential service time can be described with only one integer – the number of customers in the queue. But for the station with a server with a Coxian service distribution we need two numbers - one for the number of customers and the other for the state (phase) of the server. Further, for example, we may require more paremeters if there may be more classes of customers in the queue and so on.

Next we have to enumerate all potential transitions among states and define for them the transition rates $\left(q_{ij}\right)$. That way we generate *transition rate matrix* (or *infinitesimal generator matrix*, $Q=\left(q_{ij}\right)$ given by

$$q_{ij}\left(t\right)=\lim_{\Delta t\to 0}\frac{p_{ij}\left(t,t+\Delta t\right)}{\Delta t} \quad \text{for } i\neq j, \tag{1}$$

$$q_{ii}\left(t\right)=-\sum_{j\neq i}q_{ij}\left(t\right), \tag{2}$$

where $p_{ij}\left(t_1,t_2\right)$ is the probability that the transition occurs from the state *i* to the state *j* between the moments t_1 and t_2.

The last step in our analysis is to compute probability vector ($p(t)$, see below) of the Markov chain which can be used to describe effectivenes of the network and its elements.

The components $p_i(t)$ of the (horizontal) vector $p(t)$ are the probabilities that the system is in the state *i* at time *t*. To find them we are to solve the equation

$$\frac{dp(t)}{dt} = p(t)Q(t), \tag{3}$$

that becomes

$$\frac{dp(t)}{dt} = p(t)Q, \tag{4}$$

when the transition rates are independent of time (that is when the Markov chain is *homogeneous* what we can often assume). The solution is given by

$$p(t) = e^{Qt} = I + \sum_{n=1}^{\infty} \frac{Q^n t^n}{n!}, \tag{5}$$

that is rather hard to compute [5].

However, we often deal with *steady-states* probability distribution, when there exists

$$p = \lim_{t\to\infty} p(t), \tag{6}$$

and then the rate of change of $p(t)$ at steady-state is zero, so

$$\frac{dp(t)}{dt} = 0, \tag{7}$$

and we are to solve a simpler equation

$$pQ = 0,$$

(now p is written as independent of time). It is worth noticing that the matrix Q is singular, so there exists a nonzero solution and if Q is of rank $n-1$ (this case is the most interesting for us) there exists exactly one nonzero solution satisfying an additional equation, namely

$$\sum_i p_i = 1. \tag{9}$$

4. Iterative GMRES algorithm

There are many ways to solve such an equation. The first class of methods are direct methods which are variants of the Gaussian elimination algorithms. Applying them to our problem requires some caution, because of singularity of the matrix *Q*. These methods can make the matrix denser which may use up the computer memory (for huge matrices) and slows down performance (changing a

zero element to a nonzero element in a sparse matrix is time-consuming because of specific storage schemes for sparse matrix).

Other classes include iterative (like the power method, Jacobi method,Gauss-Seidel, SOR etc.) and projection methods one of wich (namely: iterative GMRES algorithm) we implemented.

Generalized minimum residual algorithm (GMRES) begins by constructing an orthonormal basis for a Krylov subspace (of dimension m much lesser than the size of Q) from the normalized residual u_1 produced from an initial approximation x_0:

$$r_0 = -Q^T x_0 , \tag{10}$$

$$b = \|r_0\|_2 , \tag{11}$$

$$u_1 = \frac{r_0}{b} . \tag{12}$$

Constructing an orthonormal basis for a Krylov subspace is done by the Arnoldi process (for $j = 1,...,m$)

$$w = Q^T u_j , \tag{13}$$

$$\text{for } i = 1,...,j: \quad h_{ij} = u_i^T w \quad w = w - h_{ij} u_i ,$$

$$h_{j+1,j} = \|w\|_2 ,$$

$$u_{j+1} = \frac{w}{h_{j+1,j}} .$$

The next step is to solve the least square problem by finding the vector $y = (y_1,...,y_m)^T$ that minimizes the function

$$\|b\, e_1 - Hy\|_2$$

where

$$e_1 = (1,0,...,0)^T \quad \text{is of size } m+1,$$

H is the $m+1 \times m$ marix and

$$H = \begin{cases} h_{ij} & \text{for } i \le j+1, \\ 0 & \text{for } i > j+1, \end{cases}$$

and to compute the new approximate solution

$$x = x_0 + \sum_{i+1}^{m} u_i y_i . \tag{14}$$

In *iterative GMRES* we start over with $x_o = x$, if computed approximation is insufficient.

5. Storage of sparse matrices

If we want to use the iterative GMRES algorithm to solve Markov chains representing behaviour of a queuing network we should expect the matrix (and of course vectors) of huge sizes.

First, we must consider storage schemes for the sparse matrix. To store such a matrix in an efficient manner we use three one-dimensional arrays (after [4] and others). First of them (with real items) is used to hold the nonzero elements of the matrix. The elements are stored by rows of Q^T not Q, because we need rows of Q^T to multiply it by a vector in our algorithm), but the elements need not to be in order within a row (in other words: it is only necessary for all elements of row i to be before all elements of row $i + 1$).

Two other arrays are integer arrays. The first integer array contains the column index of each nonzero element. The second integer array is much smaller than the former two arrays - it holds only one number for each column: starting position of the column in the former arrays.

6. Implementation

In our distributed implementation of iterative GMRES we used MPI (*Message-Passing Interface*, [6-7]) standard that allows writing distributed programs relatively easy. Program was written in C, compiled with gcc under Linux and tested on a collection of PC stations (*nodes*) connected with 10Mb Ethernet.

To achieve the best size performance we decide to divide evenly among nodes the matrix Q^T (n/k consecutive rows for each node, n is the size of the matrix, k is the number of nodes) and vectors r_0, x_0, w and u_i, $i = 1,...,m+1$ (n/k consecutive items of each vector for each node).

That is how operations of GMRES are held in our distributed implementation: All operations on vectors (as scaling, adding, multiplying, computing their norm) are computed locally by a node on the node's part of the vector(s). Then the partial result is exchanged with others nodes (only if it is necessary - as for computing the norm of a vector or the scalar product of two vectors).

To multiply the matrix Q^T by a vector we have to gather all components of the vector at the node (so we need one additional full-sized vector in each node) *before* multiplying but we do not need to exchange the elements of the product, because each node holds 'its own' part of the vector after multiplying.

7. Performance consideration

We focused on the best utilisation of the node memory. Let us see how much memory a node needs to perform computations. Let n be the size of the matrix

Q^T, nz be the number of nonzero elements of the matrix Q^T and k be the number of nodes. Let f be the size of a float number representation and i be the size of an integer number representation (in memory units, eg. bytes)

Each node requires $(nz \cdot f + nz \cdot i + n \cdot i)/k$ bytes for storing its part of the matrix Q^T, $(n \cdot f)/k$ bytes for its part of each vector r_0, x_0, w and u_i, $i = 1,...,m+1$ and $n \cdot f$ bytes for the additional full-sized vector.

8. Conclusion

We managed to create a program, which can find solution of a Markov chain of arbitrary size of state space (for example with $k = 20$ nodes, 128MB physical RAM each, we can solve problems with $n = 5 \cdot 10^6$ and $nz = 75 \cdot 10^6$) – if we have at our disposal a suitable number of nodes connected with fast communication media and protocols.

Unfortunately, the time performance was unsatisfactory but there are many points in which we are going to correct performance – both in algorithm (for example by using fast numerical subroutines – as BLAS etc. – for computations on vectors) and in connection of the nodes (by changing the network topology etc.).

References

[1] Chandy K.M., *The analysis and solution of general queueing networks*, Proceedings of the Sixth Princeton Conference of Information Sciences and Systems, Princeton, NJ, (1972) 224.
[2] Neuts M.F., *Matrix Geometric Solutions in Stochastic Models - An Algorithmic Approach*, Johns Hopkins University Press, Baltimore, (1981).
[3] Czachórski T., *Modele kolejkowe w ocenie efektywności sieci i systemów*, Gliwice (1999), in Polish.
[4] Stewart W.J., *Introduction to the Numerical Solution of Markov Chain*, Princeton University Press, Princeton, NJ, (1994).
[5] Moler C., van Loan C., *Nineteen dubious ways to compute the exponential of a matrix*, SIAM Review, 20 (1978) 801.
[6] Pacheco P.S., *A User's Guide to MPI*, University of San Francisco, (1998).
[7] Bylina B., „Komunikacja w MPI", *Informatyka Stosowana S2/01*, V Lubelskie Forum Informatyczne, Lublin, (2001) 31, in Polish