12th GI/ITG CONFERENCE ON MEASURING, MODELLING AND EVALUATION OF COMPUTER AND COMMUNICATION SYSTEMS

3rd POLISH-GERMAN TELETRAFIC SYMPOSIUM

Distributed Evaluation of stochastic Petri nets

Alexander Bell
Department of Applied Mathematics
University of Twente, P.O. Box 217, 7500AE, Enschede, Netherlands
a.bell@math.utwente.nl

Abstract
In this paper we present results on the distributed performance evaluation and model checking of systems specified by stochastic Petri nets. The approaches discussed rely on an explicit state-space generation and target at the usage of clusters of workstations. We present results for systems with several hundreds of millions of states. For the case studies addressed, the distributed algorithms scale very well.

Keywords
Petri nets, Markov chains, performance evaluation, model checking, linear systems, cluster computing

1 Introduction

Stochastic Petri nets (SPNs) are widely used as a high-level modelling formalism in performance evaluation to specify complex systems. Two important criteria for the quality of such systems are efficiency and correctness. We address both of them. The efficiency of a system is evaluated by model based performance evaluation, in our case of an SPN, whereas the correctness of a system is validated by model checking, where we verify that properties specified in the computational tree logic (CTL) [5], a temporal logic, hold in the SPN model.

When evaluating realistic systems, one usually encounters (at least) two problems: the size of the state-space of the system being modelled is prohibitive, an issue known as the state-space explosion problem, and the time required for both the performance evaluation steps and the model checking of even simple properties is very large. We simply accept the largeness of the system's state-spaces. The usage of multiprocessor systems, especially clusters of workstations, which offer large amounts of aggregated main memory and computational power at a distinguished price-performance ratio, helps us to consider models that cannot be evaluated serially and to shorten computation times considerably.

The rest of this paper is organised as follows. In Section 2 we present a brief introduction of the topics discussed in the thesis in conjunction with the main ideas of the used approaches. Subsequently we present serial and parallel results for specific case studies in Section 3. Finally Section 4 concludes this paper.

2 Overview

2.1 Performance evaluation

The performance evaluation of a system can be structured into the following steps, which are also depicted in Figure 1:

1. **Modelling** describes the work of transforming a real world system into a mathematical model. We do not address this job.

2. **State-space and reachability graph generation** is the task to map the high-level description onto a continuous-time Markov chain (CTMC). This CTMC is defined by its state-space S of size $n = |S|$ and its generator matrix Q. (We assume that the state-space of the underlying Markov chain is finite.) Essentially, this problem is solved by a graph search which explicitly enumerates the reachable states. For details on this step see [6].

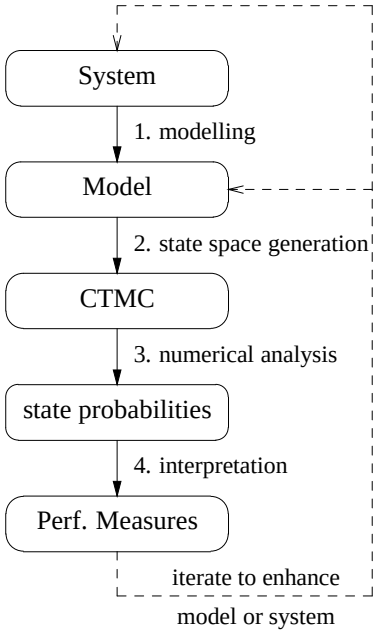


Figure 1: Performance evaluation steps

3. **Numerical analysis of the CTMC** deals with the calculation of transient and steady-state probability distributions. Mathematically, these two probability distributions are computed as the solutions of linear equation and differential equation systems. Note that both systems are equation systems with as many unknowns as the number of states of the examined CTMC. For an elaborate discussion, including a review of the achieved accuracy, of this step see [1, 3].

4. **Measures of interest** can be calculated from the transient or steady-state probability distributions using some simple vector calculations. This step is no challenge, neither serially nor in parallel.

Of course, in practice it often does not suffice to go through these four steps once, but one has to do several iterations of this performance evaluation cycle, where results from earlier models may influence the detail of the model, the choice of some parameters, or even the system as shown by the dotted arrows on the right hand side of

Figure 1. On the other hand, a model may be too detailed, resulting in a state-space for which a numerical analysis is not possible at all. In that case some details have to be omitted.

2.2 Model Checking

Model checking is a powerful method to verify the correctness of a system, where the system is modelled by a transition system, called the *implementation* of the system. Additionally, one considers the *specification* of the system, which is formulated in some temporal logic. An implementation of a system is called correct, if it meets its specification.

We used Petri nets to describe the implementation of a system and CTL to specify system properties. As our approach requires the state-space of the system to be available in main memory, we face the same problem as during the performance evaluation cycle discussed above, namely the largeness of the state-spaces. In order to model check realistic system we developed serial as well as parallel algorithms to perform this task.

3 Results

In this section we present results obtained from the PARSECS (**Par**allel **s**tate-space **e**xplorer and Markov **c**hain **s**olver) tool developed in the context of [3]. The serial measurements were taken on a dual 1 GHz Pentium III system (using only one processor) with 2 GB of RAM. The parallel results were obtained on a cluster of 26 Linux workstations, called nodes. Each node was equipped with two 500MHz Intel Pentium III processors, and 512MB main memory. The workstations were connected via switched fast Ethernet (100Mbps). Whenever we do not use all 52 processors, we use only one processor per node.

Beside absolute runtimes (wall-clock time) we measure the quality of a parallel program in the terms of **speedup** $S(N)$, the ratio of the one processor runtime and the time required by the parallelised program on an N processor system, and **efficiency** $E(N) := S(N)/N$. Then computing these values we used a cluster node to measure the serial runtime.

3.1 State-space generation

In Figure 2 we present the required runtimes for the serial and distributed state-space generation of a complex GSPN, the so-called **flexible manufacturing system (FMS)** [4]. This model can be parameterised with the number of pallets k circulating through it to create state-spaces of different sizes. Serially we were able to generate

state-spaces of up with up to 216 million states in 10.5 hours, while the distributed generation of a model consisting of 724 million states was possible in less than 3 hours on the cluster. In both cases the amount of available main memory was the limiting factor.

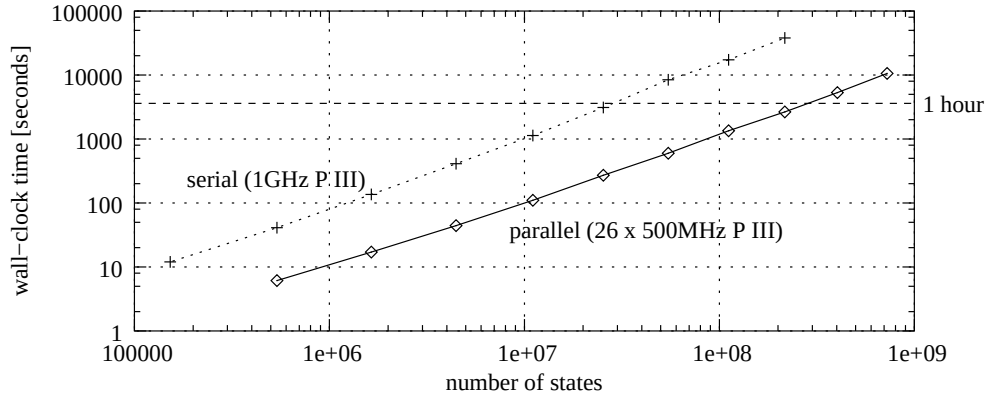


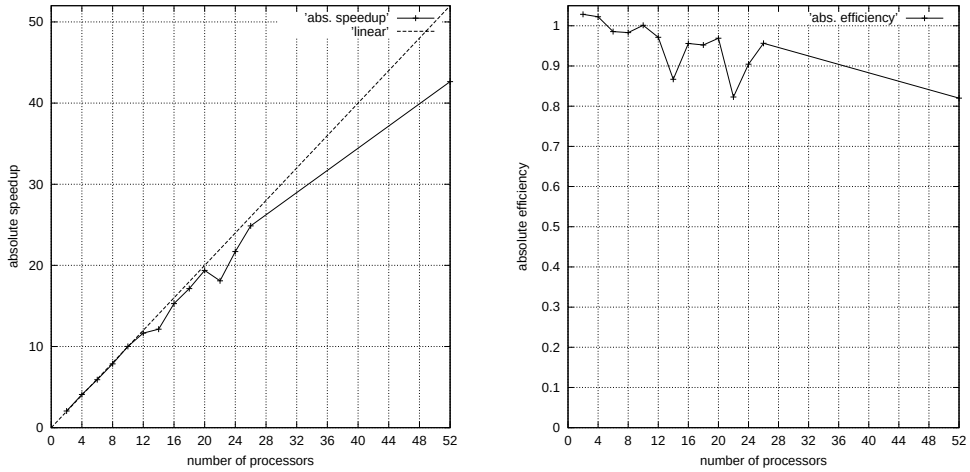
Figure 2: Wall-clock times for the serial and distributed state-space generation

While the figure above compares computers of different speeds, we compare the achieved absolute speedups to the *ideal* linear speedup in Figure 3(a). Figure 3(b) presents the corresponding absolute efficiencies. Two comments have to be made: First, for two and four processors one recognises that the efficiencies are larger than 1.0 which corresponds to a so-called superlinear speedup. Secondly one observes that even when using 2 processors per cluster node (52 processors) we achieve a noticeable efficiency of 0.82.

3.2 Numerical Solution

We could compute the steady-state solution for all models that we were able to generate, but the required time gets intolerable for large models. Using the serial version of our tool the solution time exceeds one day for the FMS model with 54 million states and it took 12.5 days to solve the largest case comprising of 216 million states. This case could be solved in less than three days using the cluster of workstations. In the largest case addressed (724 million states) the computation of the solution also required slightly over 12 days.

Achievable speedups depend heavily on the used numerical solver, the net structure and the number of processors used. Using 26 processors typical efficiencies range between 0.2 and 0.4. The usage of two processors per node led to nearly no reduction of computation times. For an extensive discussion of this topic and detailed results see [3].



(a) absolute speedup (b) absolute efficiency

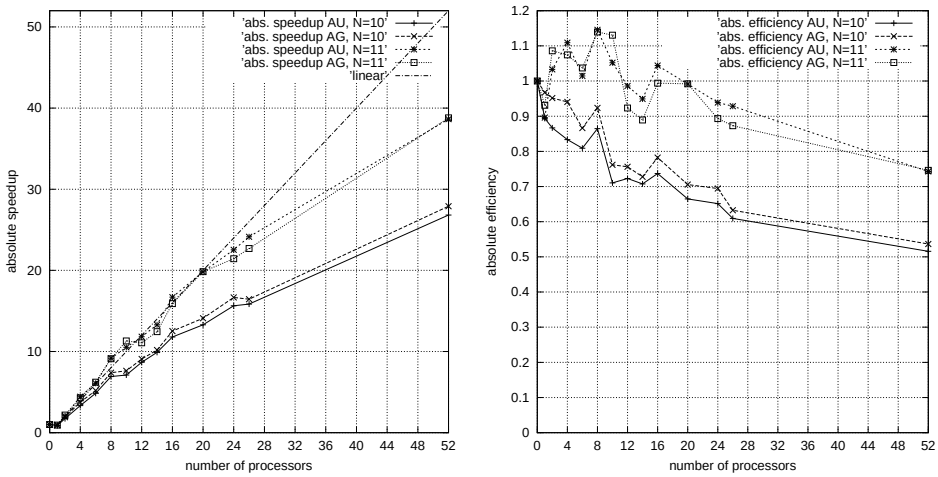
Figure 3: Results for the distributed generation (54 million states)

3.3 Model checking

As a model checking case study we selected the well-known dining-philosopher problem. In Figure 4 we show the absolute speedups and efficiencies for 10 and 11 philosophers obtained when checking for-all-until (AU) and for-all-globally formulas, which we consider the most difficult CTL formulas. Notice that the speedups and efficiencies for the AG test are slightly higher than for the AU test, even though the AG test involves a "more complicated" hierarchical CTL expression. The reason for this is that the AG test involves a number of subexpressions (like negations) that can be performed in parallel without communications, hence, of the overall longer computation time, a larger fraction can be parallelised perfectly. Details on this topic can be found in [2].

4 Conclusions

From the topics addressed state-space generation as well as CTL model checking can be done efficiently using a cluster of workstations. Efficiencies exceed 0.6 for all considered case studies. The main bottleneck in the SPN performance evaluation process is the numerical solution of the underlying CTMC. For this step we observed efficiencies as low as 0.2 using 26 processors.



(a) absolute speedups (b) absolute efficiencies

Figure 4: Absolute speedups/effi ciencies for th dining philosophers model

References

[1] A. Bell and B.R. Haverkort. Serial and parallel out-of-core solution of linear systems arising from generalised stochastic Petri net models. In A. Tentner, editor, *Proceedings High Performance Computing Symposium — HPC 2001*, pages 242–247. Society for Computer Simulation, 2001.

[2] A. Bell and B.R. Haverkort. Sequential and distributed model checking of petri net specifi cations. In L. Brim and O. Grumberg, editors, *Electronic Notes in Theoretical Computer Science*, volume 68. Elsevier, 2002.

[3] Alexander Bell. *Distributed Evaluation of stochastic Petri nets*. PhD thesis, RWTH Aachen, 2004. to be published.

[4] G. Ciardo and K.S. Trivedi. A decomposition approach for stochastic reward net models. *Performance Evaluation*, 18(3):37–59, 1993.

[5] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verifi cation of fi nite-state concurrent systems using temporal logic specifi cations. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[6] B.R. Haverkort, A. Bell, and H. Bohnenkamp. On the effi cient sequential and distributed generation of very large Markov chains from stochastic Petri nets. In *Proceedings of the 8th International Workshop on Petri Nets and Performance Models*, pages 12–21. IEEE Computer Society Press, 1999.