PVeStA: A Parallel Statistical Model Checking and Quantitative Analysis Tool

Musab AlTurki and José Meseguer

University of Illinois at Urbana-Champaign,
Urbana, IL 61801, USA

Abstract. Statistical model checking is an attractive formal analysis method for probabilistic systems such as, for example, cyber-physical systems which are often probabilistic in nature. This paper is about drastically increasing the scalability of statistical model checking, and making such scalability of analysis available to tools like Maude, where probabilistic systems can be specified at a high level as probabilistic rewrite theories. It presents PVeStA, an extension and parallelization of the VeStA statistical model checking tool [10]. PVeStA supports statistical model checking of probabilistic real-time systems specified as either: (i) discrete or continuous Markov Chains; or (ii) probabilistic rewrite theories in Maude. Furthermore, the properties that it can model check can be expressed in either: (i) *PCTL*/*CSL*, or (ii) the QuaTEx quantitative temporal logic. As our experiments show, the performance gains obtained from parallelization can be very high.

1 Introduction

Statistical model checking (see, e.g., [9,11]) is an attractive formal analysis method for probabilisitic systems. Although the properties model checked can only be ensured up to a user-specified level of statistical confidence (as opposed to the *absolute* guarantees provided by standard probabilistic model checkers), the approximate nature of the formal analysis is compensated for by its better scalability, the fact that the models to be analyzed can often be known only approximately, and the interest in analyzing quantitative properties for which an approximate result within known bounds is quite acceptable.

There are many systems for which this kind of statistical model checking analysis can be very useful. For example, distributed real-time systems, including so-called cyber-physical systems, are often probabilistic in nature, both because they often use probabilistic algorithms, and due to the uncertain, stochastic nature of the environments with which they interact. Furthermore, *quality of service properties* may be as important as traditional boolean-valued properties such as safety properties. For example, in a secure communications system, *availability* of vital information may be as important as its *secrecy*, but availability may be utterly lost due to a denial of service (DoS) attack with no loss of secrecy. Suppose that such a system is hardened against DoS attacks. How should one formally analyze the effectiveness of such a hardening? What is needed is not a Boolean-valued yes/no answer, but a *quantitative* one in terms of the expected *latency* of messages under certain assumptions about the attacker and the network. Quantitative information may include probabilities $p \in [0, 1]$, but need not be reducible to probabilities. For this reason, it is important to support statistical model checking not only of standard probabilistic temporal logics such as *PCTL*/*CSL*, but also of *quantitative temporal logics* like QuaTEx [1], where the result of evaluating a temporal formula on a path is a real number. This of course includes the case of probabilities, as values $p \in [0, 1]$, and even of standard truth values, as values in $\{0, 1\}$, as special cases.

This paper is about drastically increasing the scalability of statistical model checking, and also about making such scalability of analysis available to tools like Maude, where probabilistic systems can be specified at a high level as *probabilistic rewrite theories* [1], which are theories in rewriting logic [7] that may contain, in addition to regular rewrite rules, *probabilistic rewrite rules* modeling probabilistic transitions of such systems. The paper presents PVeStA, an extension and parallelization of the VeStA statistical model checking tool [10]. PVeStA supports statistical model checking of probabilistic real-time systems specified as either: (i) discrete or continuous Markov Chains; or (ii) probabilistic rewrite theories in Maude. Furthermore, the properties that it can model check can be expressed in either: (i) *PCTL*/*CSL*, or (ii) QuaTEx. As our experiments show, since statistical model checking is based on Monte-Carlo simulations, which are naturally parallelizable, the performance gains can be very high. In summary, the main contribution of this work is to parallelize the model checking algorithms and the VeStA tool itself, so that a wide range of statistical model checking analyses can be performed with high efficiency on probabilistic models such as Markov chains and probabilistic rewrite theories.

2 Efficient Parallel Statistical Analysis Algorithms

Sen et. al. [9] described an algorithm $\mathcal{A}$ based on simple hypothesis testing for statistical model checking of formulas in both: (1) *Probabilistic CTL* (*PCTL*) [5], which extends standard *CTL* by associating probability measures to computation paths of a probabilistic system and qualifying the temporal logic formulas with probability bounds, and (2) *Continuous Stochastic Logic* (*CSL*) [3,4], which further extends *PCTL* by continuous timing and qualifying temporal logic operators by time bounds. Given a probabilistic model $\mathcal{M}$, a *PCTL*/*CSL* formula $\mathcal{P}_{\bowtie p}(\varphi)$, with φ a state or path formula[1], and error bounds α and β, the algorithm $\mathcal{A}$ checks satisfiability of the formula by setting up a statistical hypothesis testing experiment such that its Type I and Type II errors are bounded, respectively, by α and β. The test is based on the sample mean of n random samples of φ computed over n Monte-Carlo simulations of the model. The algorithm uses standard statistical methods to precompute the total number n of samples needed to achieve the desired test strength (see [9] for more details).

[1] We restrict our attention to non-nested probabilistic formulas here, although the algorithm of [9] can handle nested formulas as well.

To be able to express not just probabilities of satisfaction of temporal logic formulas but also *quantitative properties* such as, for example, the expected latency of a probabilistic communication protocol, *PCTL* and *CSL* have been generalized to a logic of *Quantitative Temporal Expressions* (QuaTEx) in [1], in which state formulas and path formulas are generalized to user-definable, *real-valued state expressions* and *path expressions*. In [1], Agha et. al. proposed a statistical quantitative analysis algorithm $\mathcal{Q}$ for estimating the expectation of a temporal expression in QuaTEx. Given a probabilistic model $\mathcal{M}$, an expectation QuaTEx formula of the form $\mathbf{E}[\mathit{Exp}]$, with *Exp* a QuaTEx state or path expression, and bounds α and δ, the algorithm $\mathcal{Q}$ approximates the value of $\mathbf{E}[\mathit{Exp}]$ within a $(1-\alpha)100\%$ confidence interval, with size at most δ, by generating a large enough number n of random sample values $x_1, x_2, \ldots, x_n$ of *Exp* computed from n independent Monte Carlo simulations of $\mathcal{M}$. The value returned by the algorithm as the estimator for $\mathbf{E}[\mathit{Exp}]$ is the sample mean $\bar{x} = \frac{\Sigma_{i\in[1,n]} x_i}{n}$. To guarantee the quality and size requirements of the confidence interval (given respectively by α and δ) for $\bar{x}$, the number n of sample values must be large enough. In general, the more accurate the estimator, the larger the number of samples required. To generate enough samples, the algorithm $\mathcal{Q}$ uses student's t-distribution to compute a $(1-\alpha)100\%$ confidence interval by iteratively generating them in batches of N samples each (with $N > 5$). Once the size of the computed interval falls below the threshold δ, $\mathcal{Q}$ halts and the sample mean $\bar{x}$ is returned (more details can be found in [1]).

In this work, we develop parallel versions $\mathcal{A}^p$ and $\mathcal{Q}^p$ of both algorithms in which the task of computing a set of n sample values for a state or path formula in *CSL* or QuaTEx is done in parallel by performing n Monte Carlo simulations in parallel. Both parallel algorithms make no assumptions about the underlying parallel architecture. For *CSL*, $\mathcal{A}^p$ assumes non-nested probabilistic formulas.

The algorithms take as input a list of available computing resources R on which the task of generating random samples is mapped. This task is first distributed as evenly as possible by determining the number of simulations m_i to be performed by each available computing resource R_i in R. In $\mathcal{A}^p$, since the total number of samples n is precomputed, m_i is simply either $\lfloor n/|R|\rfloor$ or $\lfloor n/|R|\rfloor + 1$. In $\mathcal{Q}^p$, m_i is computed as a positive integer multiple of $|R|$ and the *load factor* k, which is a parameter to $\mathcal{Q}^p$ that can be used to increase the number of simulations performed by each resource in a round. Given a verification task, the load factor k can be tuned to optimize performance, especially for lightweight simulations when the desired statistical confidence is high, as we will see in Section 4. Once m_i is determined, each resource R_i performs m_i discrete-event simulations of the model $\mathcal{M}$ and returns a list of m_i random samples. Once the samples from all resources are collected, $\mathcal{A}^p$ and $\mathcal{Q}^p$ proceed as their sequential counterparts.

3 Implementation of PVeStA

We have implemented a client-server prototype, PVeStA, of both parallel algorithms $\mathcal{A}^p$ and $\mathcal{Q}^p$, in Java, based on the Java implementation of the original algorithms in VeStA [10]. The tool, which is available for download online at `http://www.cs.illinois.edu/~alturki/pvesta`, consists of two command-line-based executable programs: (1) a client program `pvesta-client`, which implements the sequential parts of the algorithms performing simple hypothesis testing for *PCTL*/*CSL* formulas and confidence interval computations for QuaTEx expressions, and (2) a server program `pvesta-server`, which implements the role of a resource R_i that computes random samples by performing discrete-event simulations of a given model expressed as a Markov chain or as a probabilistic rewrite theory. Figure 1 presents a schematic diagram of the structure and interactions of the client and server parts of the tool.

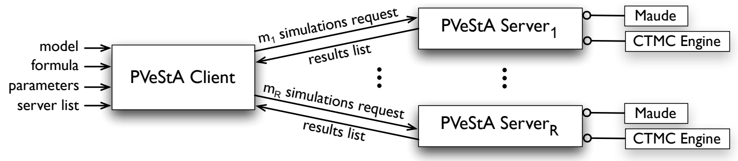


Fig. 1. Components and interactions of PVeStA

The client program first reads a list of servers R that are available for performing simulations. It then creates, using Java's managed concurrency library, a thread pool of $|R|$ *callable* computation threads, which are Java threads that implement the `Callable` interface by specifying a `run` method to be called when the thread is invoked. Each thread, which will manage simulation requests and responses with a particular server in R, is supplied with a pseudo-random seed to be used by its corresponding server to guarantee statistical independence of the simulations. The thread pool is then submitted to an *executor* object, which invokes all the threads in the pool, commencing communication with the servers in R. Upon receiving the simulations request, each PVeStA server performs the requested number of simulations using either Maude (for models expressed as probabilistic rewrite theories) or the built-in Continuous-Time Markov Chain (CTMC) engine (for CTMC models) and produces a list of sample results. The client collects all samples in an array of $|R|$ `Future` Java class objects, from which the results are extracted and then used in performing the appropriate sequential computations. For confidence interval computations, the client may need to repeat this process until enough samples are collected.

4 Experimental Evaluation

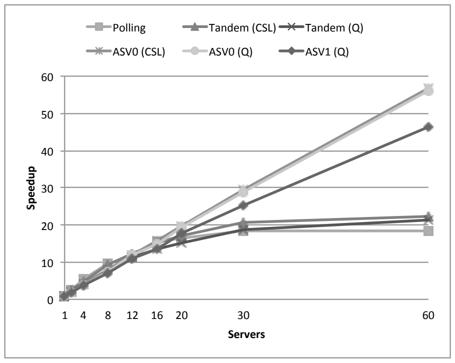
We have conducted two sets of experiments with PVeStA to evaluate the performance gains of parallelization using two different parallel architectures: (1) a high-performance computing (HPC) architecture, in which simulation tasks are distributed over different nodes in a PC cluster, and (2) a multi-core architecture, in which simulations are distributed over different processing cores within a single node. The HPC benchmarks were executed on a PC cluster consisting of 256 nodes, each of which has two (single-core) AMD Opteron 2.2GHz CPUs with 2GB of RAM. The second set of experiments was performed on a server machine having two quad-core 2.66GHz Intel Xeon processors with 16GB of RAM.

We use two examples from [10]: (1) a simplified server polling system, *Polling*, and (2) a simple tandem queuing system, *Tandem*, both expressed as continuous-time Markov chains. In addition, we use two variants of a larger case study from [2], which provides a probabilistic model in rewriting logic of the Adaptive Selective Verification (ASV) protocol [6] for thwarting DoS attacks. The first variant, denoted ASV_0, assumes a reliable communication channel, a fixed attack rate, and no message transmission delays, while the second variant, ASV_1, is slightly more realistic as it assumes a lossy channel, a variable attack rate, and random delays. Benchmarking is performed by measuring the total time required (including any additional time required for file and network I/O, thread and object management, and so on) to verify a probabilistic *CSL* formula in *Polling*, *Tandem*, and ASV_0, or a QuaTEx expectation expression in *Tandem* (load factor, $k = 100$), ASV_0 ($k = 1$), and ASV_1 ($k = 1$). The results are summarized in Table 1.

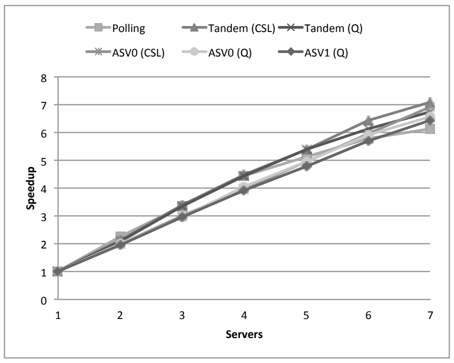
	Polling (CSL)	Tandem (CSL)	Tandem (Q)	ASV_0 (CSL)	ASV_0 (Q)	ASV_1 (Q)
Simulations	16,906	16,906	46,380	1051	706	1,308
Servers	HPC Cluster					
1	6.78	9.54	17.36	494.9	770.8	1,584.3
2	2.61	4.06	8.56	248.4	385.4	798.5
4	1.24	2.01	4.26	124.2	197.1	410.5
8	0.70	1.02	2.19	62.1	103.4	221.9
12	0.59	0.77	1.53	41.4	65.3	144.3
16	0.44	0.63	1.27	31.1	52.3	116.6
20	0.42	0.56	1.14	25.1	39.4	89.9
30	0.37	0.46	0.93	16.9	26.7	63.1
60	0.38	0.43	0.82	8.7	13.7	34.2
Servers	Multi-core Computer					
1	3.83	5.53	11.26	367.7	559.7	1,167.9
2	1.70	2.60	5.43	184.5	281.1	589.5
3	1.15	1.62	3.36	122.9	189.4	396.5
4	0.86	1.24	2.53	92.3	138.7	298.3
5	0.74	1.03	2.09	74.2	113.1	243.0
6	0.66	0.86	1.84	61.8	94.5	204.5
7	0.62	0.78	1.66	53.1	85.1	181.2

Table 1. The (average) times in seconds taken by PVeStA to complete six verification tasks using a PC cluster and a multi-core computer

As the table clearly shows, performance gains as a result of parallelization can be substantial. For example, for ASV_1, a verification task that would normally require about 27 minutes, can be completed in about 34 seconds on an HPC cluster using 60 nodes, and a 20-minute task can be done in just above 3 minutes on a multi-core machine using seven cores in parallel. In practice, several factors influence the speedups achieved by PVeStA, including the complexity of the model and the formula, and the statistical strength of the result. Figure 2(a) plots the speedups achieved against the number of servers used for HPC experiments in Table 1. We note that while performance scales almost linearly with the number of servers used for ASV_0 and ASV_1, the speedups for both *Polling* and *Tandem* begin to decelerate beyond 20 servers. This is primarily because the models *Polling* and *Tandem* are so simple that, as the number of servers increases, the time needed to generate random samples begins to be dominated by other computations in the tool. For the *Tandem*-Q experiment, which requires a fairly high statistical confidence, and thus a higher number of random samples, achievable speedups are greatly influenced by the chosen load factor k. In general, for such simple models, a higher value of k (and thus a higher number of simulations performed by a server in each round) translates into reduced processing and communication overhead and increased efficiency. For example, speedup tripled when using $k = 100$ compared with $k = 1$ for *Tandem*-Q with 60 servers. Of course, excessively high values of k result in an unnecessarily excessive number of simulations and degrade performance. Appropriate values of k can be determined by experimentation using the above ideas as guidelines. Figure 2(b), which plots speedups on a multi-core architecture, shows a similar pattern to Figure 2(a).



(a) HPC cluster



(b) Multi-core computer

Fig. 2. The speedup using multiple PVeStA servers

5 Conclusion and Future Work

We have briefly presented parallelized algorithms for statistical model checking and quantitative analysis and described their implementations in a client-server extension of VeStA, called PVeStA. Experimental evaluation on two different parallel architectures demonstrated the expected performance gains, especially for complex probabilistic models, for which the process of computing random samples can be computationally expensive. Future work include further refining the tool to improve performance for simpler models when high parallelization factors are assumed, and extending the parallel algorithms to support parallel statistical model checking of nested *PCTL*/*CSL* formulas using ideas from [9,8].

Acknowledgments. We thank Koushik Sen, Mahesh Viswanathan, and Gul Agha for the original work on VeStA, and Gul Agha and Koushik Sen for their work on PMaude and QuaTEx, which provide the basis on which PVeStA has been built. This work has been partially supported by NSF grants CNS 08-34709, CNS 07-16638, and CCF 09-05584, King Fahd University of Petroleum and Minerals, and King Abdullah University of Science and Technology.

References

1. Agha, G., Meseguer, J., Sen, K.: PMaude: Rewrite-based specification language for probabilistic object systems. Electronic Notes in Theoretical Computer Science 153(2), 213–239 (2006)
2. AlTurki, M., Meseguer, J., Gunter, C.A.: Probabilistic modeling and analysis of DoS protection for the ASV protocol. Electron. Notes Theor. Comput. Sci. 234, 3–18 (2009)
3. Aziz, A., Singhal, V., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: It usually works: The temporal logic of stochastic systems. In: P. Wolper (ed.) 7th International Conference On Computer Aided Verification. vol. 939, pp. 155–165. Springer Verlag, Liege, Belgium (1995)
4. Baier, C., Katoen, J.P., Hermanns, H.: Approximative symbolic model checking of continuous-time markov chains. In: Baeten, J., Mauw, S. (eds.) CONCUR'99 Concurrency Theory, Lecture Notes in Computer Science, vol. 1664, pp. 781–781. Springer Berlin / Heidelberg (1999)
5. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (09 1994)
6. Khanna, S., Venkatesh, S.S., Fatemieh, O., Khan, F., Gunter, C.A.: Adaptive selective verification. In: IEEE Conference on Computer Communications (INFOCOM '08). IEEE, Phoenix, AZ (April 2008)
7. Meseguer, J.: Conditional rewriting logic as a unified model of concurrency. Theor. Comput. Sci. 96(1), 73–155 (1992)
8. Sen, K., Viswanathan, M., Agha, G.: Statistical model checking of black-box probabilistic systems. Computer Aided Verification pp. 399–401 (2004)
9. Sen, K., Viswanathan, M., Agha, G.: On statistical model checking of stochastic systems. In: Computer Aided Verification (CAV 2005). LNCS, vol. 3576. Springer (2005)
10. Sen, K., Viswanathan, M., Agha, G.A.: VESTA: A statistical model-checker and analyzer for probabilistic systems. In: Second International Conference on the Quantitative Evaluation of Systems (QEST). pp. 251–252 (2005)
11. Younes, H.L.S., Simmons, R.G.: Statistical probabilistic model checking with a focus on time-bounded properties. Inf. Comput. 204(9), 1368–1409 (2006)