QoS Verification and Model Tuning @ Runtime

Antonio Filieri
DEI-DeepSE Group - Politecnico di Milano
via Golgi, 42 - Milan, Italy
filieri@elet.polimi.it

1. INTRODUCTION

Unpredictable changes continuously affect software systems and may have a severe impact on their quality of service, potentially jeopardizing the system's ability to meet the desired requirements. Changes may occur in critical components of the system, clients' operational profiles, requirements, or deployment environments. As a consequence, software engineers are increasingly required to design software as a *(self-) adaptive system*, which automatically detects and reacts to changes.

In order to detect significant changes in the execution environment, effective monitoring procedures are not enough since their outcome can seldom provide a direct support for reasoning and verification on the state of the system and its changes. The adoption of software models and model checking techniques at run time may support automatic reasoning about such changes, detect harmful configurations, and potentially enable appropriate (self-)reactions. However, traditional model checking techniques and tools may not be simply applied as they are at run time, since they may not meet the constraints imposed by on-the-fly analysis, in terms of execution time and memory occupation. The key idea to deal with verification complexity as proposed in this research consists of splitting the problem in two steps: 1) precomputing a set of closed formulae corresponding to desired properties and depending on relevant system's variables, and then 2) quickly evaluate such formulas every time a variation is detected.

This continuous verification of QoS requirements can support continuous adaptation of the software system. The term continuous here subsumes that reactions should be completed before a new variation invalidates their utility. A special, though large, class of systems behaves depending on a finite set of parameters, e.g. possible configurations, routing options, third party components selection and so on. Many control-theory based approaches have been studied to manipulate control parametersin order to reach or keep desired goals, but this continuous verification is an extremely hard task since software is usually very complex to formalize as a dynamic system, and even when it is possible, it usually shows a very complex, often non-linear, structure, which control-theory can hardly deal with. The claim of this research is that by exploiting software models at runtime, control can be made feasible and efficient. The key idea is to measure and control the system through models capturing its relevant aspects.

The goal of this research is to study new solutions for runtime efficient verification of non-functional properties and to exploit them for self-tuning.

FSE '11 Szeged, Hungary

2. RUNTIME VERIFICATION

The goal on this topic is to define a set of methodologies allowing for *time-efficient verification* of non-functional properties of software systems.

The focus of this research is on system models where the parameters that describe runtime variability are explicitly represented. Furthermore, it is assumed that phenomena reflected as changes are measurable by monitors and/or runtime predictors as values of models' parameters. The property of interest depends on all or part of those variable parameters and thus its truth must be re-verified every time a relevant parameter changes.

2.1 State of The Art

A number of methods have been proposed to support reasoning on non-functional properties of software based on models that are analyzed at run-time by relying on monitor data (e.g.[9, 19, 20]). Unfortunately most of them can be computationally expensive, which implies that detection of requirements violation may require a time frame that is incompatible with an effective reaction.

Concerning reliability, the research was up to now focused on properties formally expressed by means of Probabilistic Computational Tree Logic (PCTL) [16] assertions checked against Discrete Time Markov Chains (DTMC) [18]. The most popular way to verify the truth of PCTL assertions on DTMC models is by Probabilistic Model-Checking [1]. Model-Checking may be unsuitable for self-adaptive systems because it may require minutes or more to evaluate properties over large models, thus hindering planning and reconfiguration capabilities that must respond to tighter time bounds. In [5] Daws proposed a procedure to first convert the DTMC into a finite automaton from which it is possible to obtain a corresponding regular expression. This expression can be evaluated to produce a mathematical formula, which represents any arbitrary reachability property. Daws' approach is restricted to formulae without nested probabilistic operators and the outcoming regular expression grows

quickly with the number of states composing the DTMC ($O(n^{log(n)})$). In [15] Hahn et al. propose a refinement of the approach presented in [5] for reachability formulae, which combines state space reduction techniques and early evaluation of the regular expression in order to improve actual execution times when only a few variable parameters appear in the model. The improvement in [15] requires n^3 arithmetic operations among polynomials, performing better than [5] in most practical cases, although still leading to an expression whose length is $O(n^{log(n)})$ in the worse case.

In [14] the opposite approach is followed. Instead of trying to produce a closed formula corresponding to a reachability property, the goal is to find the sets of parameter values for which a formula is true. Such an approach essentially proceeds through an iterative refinement of an initial partition of parameter values spaces. The performance of the algorithm is quite formula-dependent, and for complex formulae it seems to take longer than [15] and [12]. Still [14] is correct and complete with respect to reachability formulae (i.e. concerning the probability of reaching a certain set of execution states) and is still under improvement.

In [17], an incremental approach for quantitative verification of Markov Decision Processes (MDPs) is proposed. Given that (D)MDPs are a superclass of DTMCs, the approach is of interest for this research. It introduces an incremental verification method, which allows for reuse of previous partial results, by exploiting the presence of Strongly Connected Components (SCCs). The approach in [17] reduces model-checking time, on average, of one order of magnitude on popular large case studies. Such a reduction might already be sufficient for a number of self-adaptive systems.

2.2 Current Achievements

The initial achievements on this research are going to appear in [12], and is focused on reliability properties. In this section the main concepts summarized.

DTMCs are defined as state-transition systems augmented with probabilities. *States* represent possible configurations of the system. *Transitions* among states occur at discrete time and have an associated probability. DTMCs are discrete stochastic processes with the Markov property, according to which the probability distribution of future states depend only upon the current state.

Formally, a (labeled) DTMC is tuple (S, S_0, P, L) where

- S is a finite set of states
- $S_0 \subseteq S$ is a set of initial states
- $P : S \times S \to [0, 1]$ is a stochastic matrix ($\sum_{s' \in S} P(s, s') = 1 \ \forall s \in S$). An element $P(s_i, s_j)$ represents the probability that the next state of the process will be s_j given that the current state is s_i.
- $L : S \to 2^{AP}$ is a labeling function which assigns to each state the set of *Atomic Propositions* which are true in that state.

This definition is here implicitly extended by also allowing transitions to be labeled with variables (in the range 0..1) instead of constants. A state $s \in S$ is said to be an *absorbing state* if $P(s, s) = 1$. If a DTMC contains at least one absorbing state, the DTMC itself is said to be a *absorbing*.

In an absorbing DTMC with r absorbing states and t transient states, rows and columns of the transition matrix P can be reordered such that P is in the following *canonical form*:

$$\mathbf{P} = \begin{pmatrix} Q & R \\ 0 & I \end{pmatrix}$$

where I is an r by r identity matrix, 0 is an r by t zero matrix, R is a nonzero t by r matrix and Q is a t by t matrix.

In the simplest model for reliability analysis, the DTMC will have two absorbing states, representing the correct accomplishment of the task and the task's failure, respectively. The use of absorbing states is commonly extended to modeling different failure conditions. For example, different failure states may be associated with the invocation of different external services. Notice that, for a sufficiently long execution time, every run of an absorbing DTMC is going to be absorbed in one of the absorbing states [18].

Due to the lack of space, PCTL is not recalled here. The interested reader can refer to [16, 1]. The most important properties of PTCL are *reachability properties*, which predicate on the probability of reaching a certain desired state. It can be shown that all the other properties can be reduced to reachability ones on properly manipulated versions of the DTMC they are being checked on [1].

Contribution. The goal is to precompute, once for all, parametric closed formulae corresponding to PCTL properties over given DTMCs; closed formulae can then be efficiently evaluated at runtime, when parameters' values will be available from monitors. As shown in [18], the probability of reaching an absorbing state s_j given that the execution of the DTMC started in s_i corresponds to the entry b_{ij} of the matrix $B = N * R$, where $N = (I - Q)^{-1}$. In an analogous way, it can be proved that the probability of passing through a state s_j before being absorbed (and given that the execution started in the transient state s_i) corresponds to the value $f_{ij} = n_{ij}/n_{jj}$.

The computation of b_{ij} and f_{ij} in the field of polynomials multivariate by parameters could be, in general, computationally expensive. But, this operation has to be done once for all. Additionally, it can be heavily parallelized as well known from matrix algorithms. Finally, there are special situations in which complexity can be reduced. One of these situations appears when the number of variable components c (i.e. components parametric outgoing transitions) is small and the matrix describing the DTMC is sparse, as very frequently happens in practice.

Let $W = I - Q$. The elements of its inverse N are:

$$n_{ij} = \frac{1}{det(W)} \cdot \alpha_{ji}(W)$$

where $\alpha_{ji}(W)$ is the cofactor of the element w_{ji}. Thus:

$$b_{ik} = \sum_{x \in 0..t-1} n_{ix} \cdot r_{xj} = \frac{1}{det(W)} \sum_{x \in 0..t-1} \alpha_{xi}(W) \cdot r_{xj}$$

Computing b_{ik} requires the computation of t determinants of square matrices with size $t - 1$. Let τ be the average number of outgoing transitions from each state ($\tau << n$ by assumption). Each of the determinants can be computed by means of Laplace expansion. Precisely, by expanding first the c rows representing the variable states (each has τ symbolic terms), we need to compute at most τ^c determinants and then linearly combine them. Each submatrix of size $t - c$ does not contain any variable symbol, by construction, thus its determinant can be computed with $(t - c)^3$ operations among constant numbers (LU-decomposition), thus much faster than the corresponding ones among polynomials. The final complexity is thus $\tau^c \cdot (t - c)^3 \sim \tau^c \cdot t^3$, which significantly reduces the original complexity and makes the design-time pre-computation of *reachability properties* feasible in a reasonable time, even for large values of t.

Similar considerations lead to the definition of $f_{ij} = n_{ij}/n_{jj} = \alpha_{ji}(W)/\alpha_{jj}(W)$ and to the estimation of their complexity.

The approach can be generalized, significantly increasing pre-computation complexity but still keeping the option of high-parallelism, to cover the entire PCTL [12]. To the best of the author's knowledge, none of the other approaches to parametric model-checking of PCTL are able expressions with nested temporal operators, as this contribution can do.

The validity of this approach has been proved via simulation and comparison with other model-checking tools [12].

2.3 Future Directions

The approach can be improved at different levels. One concerns the realization of a more efficient implementation by exploiting state-of-the-art mathematical algorithms for matrix manipulation. A distributed version could also be interesting in order to verify the scalability of the approach in case of very large systems.

Other levels of improvement and extension consist in: 1) using PCTL*, which is more expressive than PCTL and allows the expression of more complex reliability requirements, and 2) the extension to different models, e.g. CTMCs for performance and Reward models, in order to support basic cost metrics. The integration of the procedure within an established model-checker would instead lead to a comprehensive tool for DTMC-based reliability analysis.

The identification of significant benchmarks in order to compare this approach with others in the field is also a needed step toward a deeper understanding of strengths and pitfalls of each one.

3. PARAMETER TUNING

Self-adaptive software systems interact with their environment. A number of them embed control parameters which allow to "tune" the behavior of the system in different ways. Example of those systems are:

- routing algorithms for networks
- graphical user interfaces that adapt to the user
- caching strategies for OS memory management
- load-balancing procedures

The goal is to identify appropriate tuning strategies suitable for the adaptation of software systems at runtime, once the need for adaptation is discovered, as explained in the previous section. The novelty of the author's approach consists in observing and controlling an model of the software, kept alive by proper methodologies [9]. Verifying QoS properties on and controlling a proper abstraction instead of the software itself could make the complexity of software manageable, at least to some reasonable extent.

3.1 State of The Art

In recent time, Control Theory has captured a large interest from the community of Software Engineering (SE) interested in self-management as a mean to meet QoS requirements [6]. There were a number of attempts in the field of web servers, data centers, storage systems. Nevertheless Control Theory is still not a widely present practice in SE. This is largely due to the fact that developing accurate system models for software is in fact hard. Moreover, strong mathematical skills are needed in order to deal with complex non-linear dynamics o real systems [8, 6, 21]. These difficulties usually lead to the design of controllers focused on particular operating regions or conditions. Such "local" controllers can than be combined in order to be able to switch from one to another as the operating region changes.

3.2 Current Achievements

In the current stage two directions have been explored. The simplest one is a reliability-driven load-balancer; the second is a general methodology for optimal control of software systems through their DTMC representation.

Reliability-driven Load-balancing. The research issue in this case is how to design a load-balancer aware of the reliability of possible providers as well as of users' requirements. The goal for the load-balancer (LB) is to keep the desired reliability, avoiding higher failure rates, that would turn in contract violation, and higher reliability, that would imply higher costs and would reduce the market value of higher reliability solution. Another strong requirement for the LB concerns its performance: it must be fast and its computational overhead negligible.

The solution can be an adaptive feedback control with the following features:

- Linear hierarchical control: the LB exposes the same interface of the service to be provided, thus it is itself an instance of the service. Such a structure allows for hierarchical load-balancing in a natural way: each LB has only 2 possible choices, each of them can in turn be lower level load-balancer. Keeping the hierarchy as balanced as possible, requires to make $log_2(n)$ choices in order to select the proper provider. Each of these choices is as fast as rolling a dice: each load balancer knows with which fraction of the incoming requests has to be routed toward each of the two available alternatives and can behave consequently.
- PI control + Auto-Tuning: LB behavior, determined by the probability of choosing the first or second alternative, is controlled by a Proportional Integral (PI) controller [8]. Desired reliability to be provided to the users is the set-point of the PI. Variation in the nominal reliability of providers (revealed via monitoring) and variation in reliability requirements are considered "disturbs", in the sense that they are able to alter *the*unpredictably ability of the LB to meet its requirements. Controller's performance, in terms of *disturb rejection*, *spike reduction*, *convergence time to set-point* and *overshoots* are determined by two parameters, namely the proportional and the integral coefficients. These two values are hard to guess for a non expert. As part of this work an automatic tuning procedure has been design for the controller. Notice that this case concerns the tuning of the controller, not the system (which will be tuned by the LB).
- As close as possible to the goal: in real life it could happen that, due to temporary or permanent external conditions, goals become unreachable. In this case the expected behavior of the LB is to take the system as close as possible to its requirements, and to signal the unfeasibility of them to whom it may concern. These features are currently provided by this pilot project.

Controlling software through DTMC abstractions. In many practical cases, the actual behavior of a system can be modeled as a stochastic process. This can be due either to an intrinsically random behavior or, more often, to

the dependency of the deterministic procedure on external uncertainties, e.g. user decisions, communication failures, load conditions, or non-determinism of distributed systems.

Many of these behaviors can be suitably modeled by DTMCs [2]. Some transitions of the DTMC will represent observable (misurable) *disturbs*, such as monitored reliability of external services or the current usage profile expressed as a probabilistic characterization of users' choices. Other transitions will represent *control inputs*, that is, tunable configuration parameters of the system, for example the selection of a specific user interface or a certain set of advertisement banners. The goal of the controller is to continuously correct the value of control inputs in order to make the system meet its requirements, compensating the effects of disturbs. In general the number of control inputs may be different from the number of the goals, and this setting suffers an increased mathematical complexity of the controller [8]. Also, there are not automatic techniques to produce a strictly linear controller, in general. Leaving out details, the core of the problem is that possibly infinite assignments of the control inputs can make the controlled system reach its goals. This indeterminism requires appropriate techniques to be managed, since the controller must provide one single directive.

The solutions proposed in this research make use of optimal-control. In practice, a cost function is introduced as index of the quality of an assignment above the others. Costs may depend on any reasonable factor, e.g. service cost per invocation or average overhead. In case it does not make sense in the specific domain to talk about cost of the configuration, any convex, possibly linear, function of the control input can be provided. Minimizing this function would anyway make the assignment unique and the control feasible.

The entire solution is implemented in Wolfram Mathematica and shown to be effective and general.

3.3 Future Directions

Transposition of Control Theory principles into Software Engineering is becoming increasingly popular. The idea of controlling models of the software system and delegating the actuation to a synchronization layer seems promising.

The next step on the Load-Balancer topic is the extension to multiple QoS metrics and thus the adoption of model more complex than DTMCs. Also a support for some cost metrics could be interesting. Concerning the tuning methodology, the current effort is toward controlling a general CTMC model, thus including both reliability, performance and eventually rewards. There is some preliminary result that defined a first thread of exploration.

Both of the topics can benefit from the application of more advanced control techniques. First of all the adoption of some prediction model that would give to the controller a longer time to take decisions, thus possibly allowing for the realization of more effective strategies that are also more expensive to be found.

Another application field under exam concerns the control of virtual machines farms. Appropriate resource allocation strategies could reduce costs while still ensuring required QoS. An adaptive controller can do the job: based on preliminary results, it should be possible to produce adaptive controllers that allocate as much as needed, with minimal overshoots.

4. ACKNOWLEDGMENTS

Author's advisors is Prof. Carlo Ghezzi (carlo.ghezzi@polimi.it), DeepSE Group @ DEI, Politecnico di Milano.
This research has been partially funded by the European Commission, Programme IDEAS- ERC, Project 227977-SMScom.
Other publications of the author include [10, 4, 7, 3, 12, 13, 11]. He is also co-author of the Dagstuhl proceedings book "Model Driven Quality Prediction" (Springer ed.)

5. REFERENCES

[1] C. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[2] R. C. Cheung. A user-oriented software reliability model. *IEEE TSE*, 6(2):118–125, 1980.

[3] A. Ciancone, A. Filieri, M. L. Drago, R. Mirandola, and V. Grassi. Klapersuite: an integrated model-driven environment for non-functional requirements analysis of component-based systems. In *TOOLS 2011 - to appear*. 2011.

[4] A. Ciancone, A. Filieri, and R. Mirandola. Mantra: Towards model transformation testing. In *QUATIC 2010*, pages 97–105. IEEE, 2010.

[5] C. Daws. Symbolic and parametric model checking of discrete-time markov chains. *ICTAC*, pages 280–294, 2005.

[6] Y. Diao, J. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, and D. Phung. Self-managing systems: a control theory foundation. In *ECBS '05*, pages 441 – 448, 2005.

[7] S. Distefano, A. Filieri, C. Ghezzi, and R. Mirandola. A compositional method for reliability analysis of workflows affected by multiple failure modes. In *CBSE 2011 - to appear*. 2011.

[8] R. Dorf and R. Bishop. *Modern control systems*. Prentice Hall, 2008.

[9] I. Epifani, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Model evolution by run-time parameter adaptation. In *ICSE*, 2009.

[10] A. Filieri, C. Ghezzi, V. Grassi, and R. Mirandola. Reliability analysis of component-based systems with multiple failure modes. In *CBSE 2010*, volume 6092, pages 1–20. 2010.

[11] A. Filieri, C. Ghezzi, R. Mirandola, and G. Tamburrelli. Conquering complexity via seamless integration of design-time and run-time verification, to appear.

[12] A. Filieri, C. Ghezzi, and G. Tamburrelli. Run-time efficient probabilistic model checking. In *ICSE - to appear*, 2011.

[13] A. Filieri, C. Ghezzi, and G. Tamburrelli. A formal approach to adaptive software: Continuous assurance of non-functional requirements. *Formal Aspects of Computing*, under revision.

[14] E. Hahn, T. Han, A. Mereacre, and L. Zhang. Synthesis for pctl in parametric markov decision processes. *NFM - to appear*, 2011.

[15] E. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric markov models. *Model Checking Software*, pages 88–106, 2009.

[16] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[17] M. Kwiatkowska, D. Parker, and H. Qu. Incremental quantitative verification for markov decision processes. In *IEEE/IFIP DSN-PDS*. IEEE CS Press, 2011. To appear.

[18] S. Ross. *Stochastic Processes*. Wiley New York, 1996.

[19] J. Zhang and B. H. C. Cheng. Model-based development of dynamically adaptive software. In *ICSE*. ACM, 2006.

[20] T. Zheng, M. Woodside, and M. Litoiu. Performance model estimation and tracking using optimal filters. *IEEE TSE*, 34(3):391–406, 2008.

[21] X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, P. Padala, and K. Shin. What does control theory bring to systems research? *SIGOPS Oper. Syst. Rev.*, 43:62–69, January 2009.