Continual Verification of Non-Functional Properties in Cloud-Based Systems[*]

Invited Paper

Radu Calinescu, Kenneth Johnson, Yasmin Rafiq, Simos Gerasimou, Gabriel Costa Silva and Stanimir N. Pehlivanov

Department of Computer Science, University of York, UK

Abstract. Cloud-based systems are used to deliver business-critical and safety-critical services in domains ranging from e-commerce and e-government to finance and healthcare. Many of these systems must comply with strict non-functional requirements while evolving in order to adapt to changing workloads and environments. To achieve this compliance, formal techniques traditionally employed to verify the non-functional properties of critical systems at design time must also be used during their operation. We describe how a formal technique called runtime quantitative verification can be used to verify cloud-based systems continually.

1 Introduction

Few new technologies have been embraced with as much enthusiasm as cloud computing. Users, providers and policy makers have set out to exploit the advantages of the new technology and to encourage its adoption. Their success stories abound, and the range of services delivered by cloud-based systems is growing at a fast pace. Business-critical e-commerce and e-government services, and safety-critical healthcare services are increasingly part of this range.

Using cloud-based systems to deliver critical services is a double-edged sword. On the one hand, the ability of cloud to dynamically scale resource allocations in line with changing workloads makes it particularly suited for running such systems. On the other hand, its reliance on third-party hardware, software and networking can lead to violations of the strict non-functional requirements (NFRs) associated with critical services. Taking advantage of the former characteristic of cloud without being adversely affected by the latter represents a great challenge, as the traditional approaches to verifying NFRs are often ineffective in a cloud setting. NFR verification approaches such as model checking, design by contract and quality assurance were devised for off-line use during the design or verification and validation stages of system development. As a result, they have high overheads and operate with models and non-functional properties (NFPs) that in the case of cloud-based systems evolve continually as the workloads, allocated resources and requirements of these systems change over time.

In this survey paper, we describe how a formal technique called *runtime quantitative verification* [1] can be used (a) to verify the NFPs of evolving cloud-based systems continually; and (b) to guide this evolution towards configurations that are guaranteed to satisfy the system NFRs.

[*] This work was partly supported by the UK EPSRC grant EP/H042644/1.

2 Runtime Quantitative Verification

Quantitative verification is a mathematically based technique for analysing the correctness, performance and reliability NFPs of systems that exhibit stochastic behaviour [14]. To analyse the NFPs of a system, the technique uses finite-state Markov models comprising states that correspond to different system configurations, and edges associated with the transitions that are possible between these states. Depending on the type of the analysed NFPs, the edges are annotated with transition probabilities or transition rates; and the model states and transitions may additionally be labelled with costs/rewards. Given a model and an NFP specified formally in temporal logic extended with probabilities and costs/rewards, *probabilistic model checkers* that implement the technique analyse the model exhaustively in order to evaluate the property.

Like most formal verification techniques, quantitative verification is traditionally used in off-line settings, to evaluate the performance-cost tradeoffs of alternative system designs, or to analyse NFR compliance for existing systems. In the latter case, systems in violation of their NFRs undergo off-line maintenance. This approach does not meet the demands of emerging application scenarios in which systems need to be continually verified as they adapt autonomously, whenever a need for change is detected while they operate [6]. To address this need for continual verification, the runtime variant of the approach depicted in Fig. 1 was introduced in [10, 11] and further refined in [1, 4, 9, 12].

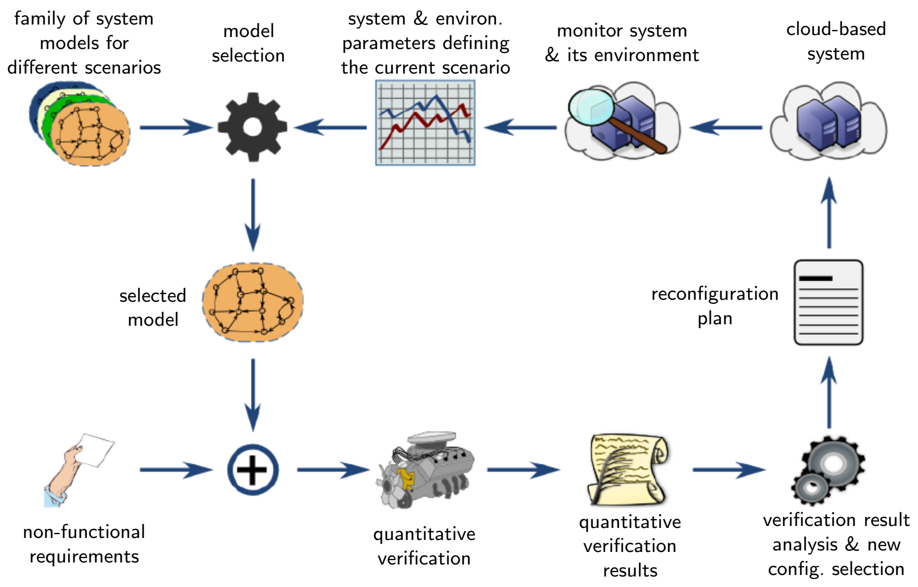
family of system models for different scenarios
model selection
system & environ. parameters defining the current scenario
monitor system & its environment
cloud-based system
selected model
reconfiguration plan
non-functional requirements
quantitative verification
quantitative verification results
verification result analysis & new config. selection

Fig. 1. Continual verification of a cloud-based system—Monitoring is used to identify the scenario the system operates in, and to select a model whose quantitative verification enables the detection or, sometimes, prediction of NFR violations. The subsequent synthesis and execution of a provably correct reconfiguration plan help the system reinstate or maintain compliance with NFRs such as response time, availability and cost.

3 Continual Verification of Cloud-Deployed Services

Continual verification can be used to manage the reliability of *multi-tier software services* deployed on cloud infrastructure owned by the service provider. As we show in [8, 13], quantitative verification at runtime enables service administrators to quantify the impact of planned and unexpected changes on the reliability of their services, as the approach provides precise answers to questions such as:

Q1 What is the maximum probability of a service becoming unavailable within a one-month time period?
Q2 How will the probability of failure for a service be affected if one of its database instances is switched off to reflect a decrease in service workload?
Q3 How many additional VMs should be used to run a service component when moving it to VMs placed on servers with fewer memory blocks?

Service administrators (or their automated resource management scripts) can then respond with remedial action if the service fails to comply with its NFRs, e.g., by discarding the planned removal of a component instance.

Compositional NFR Verification Standard verification uses a monolithic model constructed from the parallel composition of models of all system components, in order to capture interleaving and synchronised behaviour between components. Despite significant improvements in the efficiency of quantitative verification techniques and tools, this often results in high overheads that make the runtime use of the technique unfeasible because the system may change again before its latest change has been fully analysed. This problem is addressed by *compositional NFR verification*, a formal technique that replaces the analysis of monolithic system models with a sequence of verification steps carried out component-wise on local NFPs. The ordering of the steps is determined according to component dependencies within the analysed system, and ultimately helps infer global system NFPs. Compositional verification reduces the NFP analysis overheads significantly, and in [8] we show that it can be used for continual NFP verification in certain scenarios associated with cloud-based systems.

Incremental NFR Verification Cloud-deployed services operate in a highly dynamic environment where service components are frequently added, removed and scaled according to demand, or might fail unexpectedly. While compositional verification broadens the range of systems to which NFR verification can be applied effectively, it cannot cope with such rapid changes. Our INVEST *incremental verification* framework from [13] addresses this limitation by identifying and executing a minimal sequence of reverification steps after each change in a component-based system.

Practical Use Using our continual NFR verification of cloud-deployed services requires the construction of accurate models of the analysed system and the specification of its NFRs in probabilistic temporal logic. As most practitioners lack the formal methods expertise required to carry out this task, we developed domain-specific languages that administrators of cloud resources can use to specify the initial structure of their multi-tier cloud-deployed services and the changes under which their NFPs should be analysed [16].

4 Continual Verification of Service-Based Systems

Continual NFP verification can also be exploited by cloud-service users, to drive the dynamic selection of service-based system (SBS) components. Built through the integration of third-party services deployed on cloud datacentres and accessed over the Internet, SBSs are increasingly used in business-critical and safety-critical applications in which they must comply with strict NFRs. Self-adaptive SBSs achieve NFR compliance by selecting the *concrete services* for their operations dynamically, from sets of functionally equivalent third-party services with different levels of performance, reliability and cost.

Continually verified SBSs The self-adaptive SBS architecture and development methodology we introduced in [2] and extended in [1, 7] carry out this concrete service selection using continual NFR verification. Given an SBS workflow comprising $n \geq 1$ operations to be performed by third-party services, our SBS architecture uses *intelligent proxies* to interface the workflow with sets of remote service such that the i-th SBS operation can be carried out by $m_i \geqslant 1$ functionally equivalent services. The main role of the intelligent proxies is to ensure that each execution of an SBS operation is carried out through the invocation of a concrete service selected as described below. The proxy overseeing the i-th SBS operation is initialised with a sequence of "promised" service-level agreements $sla_{ij} = (p^0_{ij}, c_{ij}), 1 \leqslant j \leqslant m_i$, where $p^0_{ij} \in [0, 1]$ and $c_{i,j} > 0$ represent the provider-supplied probability of success and the cost for an invocation of service $s_{i,j}$, respectively. The intelligent proxies are also responsible for notifying a *model updater* about each service invocation and its outcome. The model updater starts from a developer-supplied initial Markov model of the SBS workflow, and uses the on-line learning techniques from [3, 5] to adjust the transition probabilities of the model in line with these proxy notifications. Finally, the up-to-date Markov model is used by an *autonomic manager* that performs runtime quantitative verification to select the service combination used by the intelligent proxies so that the SBS satisfies its NFRs with minimal cost at all times.

Efficient NFR Verification The runtime quantitative verification within our framework [3, 5] is carried out by an embedded version of the PRISM probabilistic model checker [15]. This version of PRISM can handle the "on the fly" analysis necessary for a range of small-sized SBS workflows. The compositional and incremental verification approaches described in the previous section can be used to reduce the overheads of continual NFP verification in order to extend its applicability to larger SBS workflows.

Practical Use For a continually verified SBS architecture to be practical, developers should be able to build instances of it without requiring special training in formal methods. To address this need, our work from [3, 5] introduced a tool-supported SBS development framework. In this framework, many of the components and artefacts described above are either generated automatically (e.g., starting from an annotated UML activity diagram of the SBS workflow and from the WSDL specifications of its concrete services) or workflow-independent and therefore provided as reusable middleware.

5 Conclusion

A key advantage of cloud-based systems is their ability to evolve, e.g., by scaling their resources up and down in response to changes in workload. When such evolving systems deliver business-critical or safety-critical services, their non-functional properties (NFPs) must be verified continually. We summarised recent research into using quantitative model checking to support continual NFP verification in cloud-based systems. This novel approach to NFP verification can be exploited by both providers and users of cloud-deployed services, and has been employed successfully in several case studies, e.g. [1, 5, 7, 8, 13]. Nevertheless, significant research is still needed to extend its applicability to new scenarios, to improve its scalability, and to identify and address its limitations.

References

1. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Comm. ACM 55(9), 69–77 (Sept 2012)
2. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimization in service-based systems. IEEE Trans. Softw. Eng. 37, 387–409 (2011)
3. Calinescu, R., Johnson, K., Rafiq, Y.: Using observation ageing to improve Markovian model learning in QoS engineering. In: ICPE 2011. pp. 505–510
4. Calinescu, R., Kikuchi, S., Kwiatkowska, M.: Formal methods for the development and verification of autonomic IT systems. In: Formal and Practical Aspects of Autonomic Computing and Networking. pp. 1–37. IGI Global (2010)
5. Calinescu, R., Rafiq, Y.: Using intelligent proxies to develop self-adaptive service-based systems. In: TASE 2013. pp. 131–134
6. Calinescu, R.: Emerging techniques for the engineering of self-adaptive high-integrity software. In: Assurances for Self-Adaptive Systems. Springer (2013)
7. Calinescu, R., Johnson, K., Rafiq, Y.: Developing self-verifying service-based systems. In: ASE 2013. To appear
8. Calinescu, R., Kikuchi, S., Johnson, K.: Compositional reverification of probabilistic safety properties for large-scale complex IT systems. In: Large-Scale Complex IT Systems. LNCS, vol. 7539, pp. 303–329. Springer (2012)
9. Calinescu, R., Kwiatkowska, M.: CADS*: Computer-aided development of self-* systems. In: FASE 2009. pp. 421–424. Springer
10. Calinescu, R., Kwiatkowska, M.Z.: Using quantitative analysis to implement autonomic IT systems. In: ICSE 2009. pp. 100–110
11. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time adaptation. In: ICSE 2009. pp. 111–121
12. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: ICSE 2011. pp. 341–350
13. Johnson, K., Calinescu, R., Kikuchi, S.: An incremental verification framework for component-based software systems. In: CBSE 2013. pp. 33–42
14. Kwiatkowska, M.: Quantitative verification: Models, techniques and tools. In: ESEC/FSE 2007. pp. 449–458. ACM Press
15. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: CAV'11. LNCS, vol. 6806, pp. 585–591. Springer (2011)
16. Pehlivanov, S.: Modelling of Multi-Tier Applications Deployed on Cloud Computing Infrastructure. Master's thesis, University of York (September 2013)