Continual Verification of Non-Functional Properties in Cloud-Based Systems

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

Department of Computer Science
University of York
Deramore Lane, York YO10 5GH, UK
{radu.calinescu,kenneth.johnson,yr534,sg778,gcs507,snp502}@york.ac.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. Our paper describes how a formal technique called runtime quantitative verification can be used to verify evolving cloud-based systems continually. We survey two approaches to continually verifying the non-functional properties of cloud-based systems, and discuss the main research challenges addressed during their development.

1 Introduction

Few new technologies have been embraced with as much enthusiasm as cloud computing. Users, providers and policy makers have all set to exploiting the advantages of the new technology and encouraging 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 critical-system 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* 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. Runtime quantitative verification belongs to a growing set of principled tools for model-driven quality-of-service (QoS) engineering in cloud-based systems. Other recent additions to this toolset include QoS engineering techniques underpinned by the formal analysis of Petri Nets [36], queueing performance models [22, 33, 35], finite state transition models [5, 26], and first-order logic specifications [10, 27].

The paper is organised as follows. In Section 2, we introduce runtime quantitative verification, and present the generic steps undertaken by a cloud-based system that uses the technique to achieve continual NFP verification. We then describe two approaches to using continual verification in cloud-based systems managed by cloud providers and by cloud users, respectively. The first approach, described in Section 3, uses runtime quantitative verification to re-evaluate reliability NFPs of multi-tier cloud-deployed services after every change in the cloud infrastructure used to run these services. The second approach, presented in Section 4, involves the dynamic, QoS-driven selection of the cloud services used to carry out the operations of service-based systems. The two sections highlight the research challenges associated with using continual NFP verification in cloud-based systems, and discuss some of our work to address these challenges. Section 5 concludes the paper with a brief summary.

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 [18, 24, 29, 30]. To analyse the NFPs of a system, the technique uses finite state-transition 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; additionally, the model states and transitions may be labelled with costs/rewards. The types of models with probability-annotated transitions include discrete-time Markov chains and Markov decision processes, while the edges of continuous-time Markov chains are annotated with transition rates.

Given one of these models and a non-functional system property specified formally in temporal logic extended with probabilities and costs/rewards, the technique analyses the model exhaustively in order to evaluate the property. Examples of properties that can be established using the technique include the probability that a fault occurs within a specified time period, the expected response time of a service under a given workload, and the expected energy usage

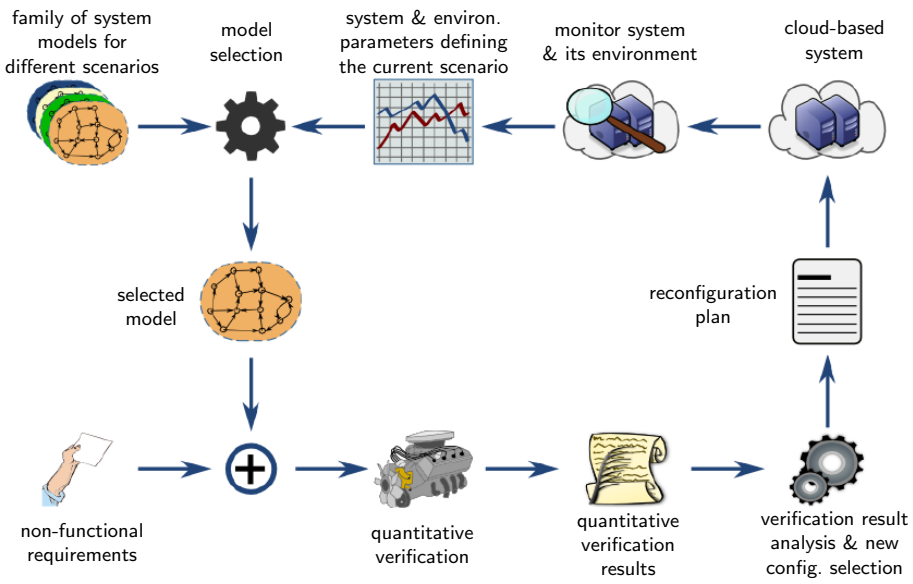

Fig. 1. Continual NFP verification of a cloud-based system

of a cloud-based system. Quantitative verification is typically performed entirely automatically by tools termed *probabilistic model checkers*. Widely used probabilistic model checkers include PRISM [31], MRMC [23] and Ymer [38].

Like most formal verification techniques, quantitative verification is traditionally used in off-line settings, e.g., to evaluate the performance-cost tradeoffs of alternative system designs, and to establish if existing systems comply with their non-functional requirements. In the latter case, systems in violation of their NFRs undergo off-line maintenance, and are eventually replaced with suitably modified system versions. This approach does not meet the demands of emerging application scenarios in which systems need to be continually verified as they adapt autonomously, as soon as a need for change is detected while they operate [7, 8]. To address this need for continual verification, a runtime variant of the technique was introduced in [13, 14] and further refined in [1, 5, 12, 16, 17].

Fig. 1 illustrates the use of runtime quantitative verification for the continual verification and reconfiguration of a cloud-based system. The system and its environment are monitored continually, and relevant changes are identified and quantified using fast on-line learning techniques. The resulting measurements enable the identification of the scenario that the system operates in, and the selection of a suitable model from a family of system models associated with different such scenarios. As an example, Bayesian learning techniques are used in [3, 6, 14] to monitor the changing probabilities of successful service invocation for a service-based system, and thus to determine the transition probabilities of a parametric Markovian chain that models the system.

The model selected through the monitoring process described above is then analysed using quantitative verification, to identify and/or predict violations of

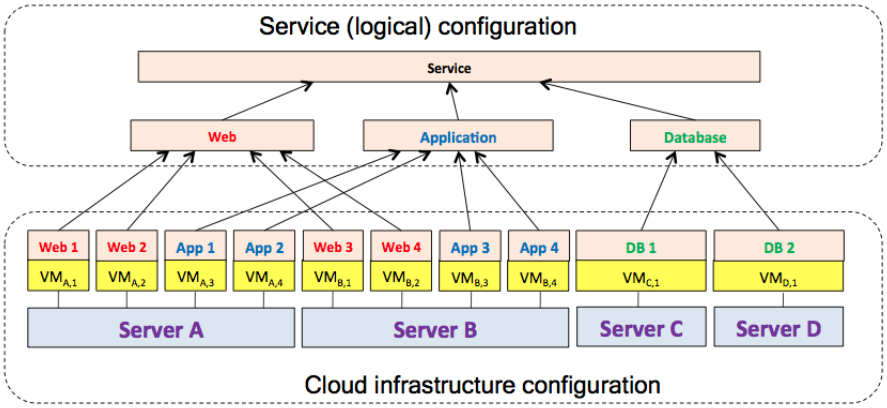

Fig. 2. Three-tier architecture of a cloud-deployed service

NFRs requirements such as response time, availability and cost. When NFR violations are identified or predicted, the results of the analysis support the synthesis of a reconfiguration plan. This plan comprises adaptation steps whose execution ensures that the system will continue to satisfy its NFRs despite the changes identified in the monitoring step.

Using quantitative verification to implement a *monitor-analyse-plan-execute* autonomic computing control loop [25] provides irrefutable guarantees about the compliance of the selected reconfiguration plans with the NFRs of the cloud-based system. Typical exploitations of this important capability in cloud-based systems are described in the next two sections.

3 Continual Verification of Cloud-Deployed Services

The approach described in this section was originally proposed in [11, 21], and involves the continual verification of the reliability of *multi-tier software services* deployed on cloud infrastructure owned by the service provider. A multi-tier software service is a collection of independent components, each delivering a specific functionality of the service, and we are interested in continually assessing the probability that the service becomes unavailable as it changes over time.

Figure 2 depicts the deployment of a service whose architecture comprises three *functions*: web, application and database. Several instances of each function run in different virtual machines (VMs) that are deployed across four physical servers that make up the cloud infrastructure.

Services deployed on cloud infrastructure take advantage of its elastic nature, scaling rapidly to meet demands by adding new servers, virtual machines and function instances. These types of changes are governed by policies and are often carried out programatically in response to fluctuations of the service workload. Other changes to the service happen unexpectedly. Examples of un-

expected changes include hardware failure of physical servers or failure of one or more function instances arising from software faults. By applying quantitative verification at runtime, cloud datacentre administrators can gain valuable insight into the impact that each change has on the service reliability by obtaining precise answers to questions such as

Q1 What is the maximum probability of a service failing over a one-month time period?[1]
Q2 How will the probability of failure for my 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 the instances of a service function when these instances are moved to VMs placed on physical servers with fewer/less reliable memory blocks?

Cloud datacentre administrators (or their automated resource management scripts) then have the option of reacting with remedial action if the service fails to comply with its non-functional requirements. They can also discard planned changes (e.g., a removal of a function instance) whose implementation would violate these requirements.

3.1 Modelling Cloud-Deployed Services

In order to apply continual NFP verification to this application domain, each component of the cloud-deployed service is associated with a probabilistic automaton (PA) modelling its operation over a specified time interval, such as one year. Queries relating to service NFPs like those in Q1–Q3 are formulated in a variant of temporal logic and checked against component models of the service. The primary obstacle to providing continuous verification to these services is their size complexity. For example, to verify the service depicted in Figure 2, we require models that formalise component behaviour for:

– The four physical servers—The four server PAs (A, B, C and D) comprise states for server configurations corresponding to the different quantities of memory blocks, hard disks and CPU units that can be operational on these servers. The states are labelled with atomic propositions expressing the numbers of operational memory blocks/disks/CPUs, and in the initial state all components installed on a server are operational. Additional states model a fault detection mechanism present on the server, and the VM migration triggered by this mechanism when the numbers of operational memory blocks, disks or CPUs on the server reach a dangerously low level. State transitions are labeled with the probabilities of individual server components (including the fault detection and VM migration) failing or operating correctly over the analysed time period.

[1] A service is deemed to have failed if all instances of one of its functions are unavailable.

– The web, application and database functions—Two PAs ($webapp_A$, $webapp_B$) model the web and application function instances deployed on servers A and B, and two further PAs (db_C, db_D) model the database function instances on servers C and D. The states of these PAs correspond to the operational status of each function instance, with atomic propositions expressing the number of operational instances in each state. All instances are operational in the initial state, and disjoint *sets of state transitions* are used to model how the status of function instances changes for different evolution paths of the servers that run the VMs containing these instances. As an example, the status of a function instance will change in different ways when the fault detection mechanism on its server operates correctly compared to the scenario in which this mechanism fails. This nondeterminism is solved by composing the function instance and server models, as explained below.
– The multi-tier service—A single service PA model *service* is used, comprising states that correspond to the operational status of each of the functions web, application and database.

Standard NFP verification requires the monolithic model

$$M = A \parallel B \parallel C \parallel D \parallel webapp_A \parallel webapp_B \parallel db_C \parallel db_D \parallel service \qquad (1)$$

to be constructed from the parallel composition of all component models in order to analyse interleaving and synchronised behaviour between components. Despite significant progress in improving the efficiency of quantitative verification tools and techniques, this often results in a high overhead or even an intractable verification task, even for small systems. Extending these tools and techniques with the ability to support continual NFP verification represents a major challenge, and in the next part of this section we summarise recent advances in addressing this challenge.

3.2 Compositional Verification

To avoid problems with state explosion in large monolithic models, compositional verification techniques have been developed to take advantage of the modular structure of cloud-deployed services. These techniques perform verification tasks component-wise, under *assumptions* that encode component NFPs and restrict the possible behaviour of other system components.

In our framework, an assumption $\langle \mathcal{A} \rangle_{\geqslant v}$ is a *probabilistic safety property* comprising a discrete finite automata (DFA) $\mathcal{A}$ whose alphabet is formed from action labels of the component models and a rational probability bound v. The finite words accepted by $\mathcal{A}$ express sequences of component actions associated with prefixes of paths that satisfy an NFP of that component. For example, the regular expressions

– $\mathcal{A}_1 = \text{serverCwarn}^+$ and $\mathcal{A}_2 = \text{serverCfail (serverCwarn} \mid \text{serverCfail)}^*$ specify sequences of actions corresponding to the fault detection mechanism of the physical server labelled C triggering a warning, and when server C fails without warning, respectively; and

– $\mathcal{A}_3 = \text{database\_fail}\text{C}^+$ specifies sequences of actions that correspond to the database function instance on server C failing.

The *probabilistic assume-guarantee* paradigm described in [32] replaces an analysis of the monolithic model constructed in eq. (1) with a sequence of quantitative 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 infer global system NFPs. For example, to determine the reliability of the database function on server C from our example of a multi-tier service, we perform the following steps:

1. The probabilistic safety properties $\langle \mathcal{A}_1 \rangle_{\geqslant v_1}$ and $\langle \mathcal{A}_2 \rangle_{\geqslant v_2}$ associated with server C are verified using standard quantitative verification to obtain the minimum probabilities v_1 and v_2 of not reaching the accepting states of DFAs $\mathcal{A}_1$ and $\mathcal{A}_2$, respectively. In symbols

$$\langle \mathit{true} \rangle \ C \langle \mathcal{A}_1 \rangle_{\geqslant v_1} \text{ and } \langle \mathit{true} \rangle \ C \langle \mathcal{A}_2 \rangle_{\geqslant v_2},$$

   where $\langle \mathit{true} \rangle$ is the property that holds under all circumstances: no assumptions are made on the behaviour of server C.
2. The probabilistic safety property $\langle \mathcal{A}_3 \rangle_{\geqslant v_3}$ associated with component model db_C is then verified using multi-objective model checking [15] to obtain the minimum probability v_3 of not reaching the accepting state of $\mathcal{A}_3$, under the assumptions $\langle \mathcal{A}_1 \rangle_{\geqslant v_1}$ and $\langle \mathcal{A}_2 \rangle_{\geqslant v_2}$ from the previous step. In symbols

$$\langle \mathcal{A}_1, \mathcal{A}_2 \rangle_{\geqslant v_1, v_2} \ db_C \ \langle \mathcal{A}_3 \rangle_{\geqslant v_3}.$$

3. We conclude that the parallel composition of C and db_C satisfies the probabilistic safety property $\langle \mathcal{A}_3 \rangle_{\geqslant v_3}$ under all circumstances:

$$\langle \mathit{true} \rangle \ C \parallel db_C \ \langle \mathcal{A}_3 \rangle_{\geqslant v_3}.$$

These steps describe the probabilistic assume-guarantee rule introduced in [32] and written in standard proof-theoretic sequent notation as

$$\frac{\langle \mathit{true} \rangle C \langle \mathcal{A}_1, \mathcal{A}_2 \rangle_{\geqslant v_1, v_2} \qquad \langle \mathcal{A}_1, \mathcal{A}_2 \rangle_{\geqslant v_1, v_2} db_C \langle \mathcal{A}_3 \rangle_{\geqslant v_3}}{\langle \mathit{true} \rangle C \parallel db_C \langle \mathcal{A}_3 \rangle_{\geqslant v_3}}. \tag{2}$$

In this way, the verification of the NFP $\langle \mathcal{A}_3 \rangle_{\geqslant v_3}$ on the model $C \parallel db_C$ can be decomposed into two separate steps involving the analysis of much smaller models. Indeed, the likelihood of the database function db_C failing depends upon the reliability of the physical server C it is deployed on.

Interdependencies between the hardware, software and other components of a cloud-based system can be derived from its architecture and expressed symbolically as a *dependency tree*. As an example, the dependency tree for our cloud-deployed service is depicted in Figure 3. By associating component models with probabilistic safety properties, probabilistic assume-guarantee rules of the form (2) are applied to models in the order specified by a depth-first traversal of this

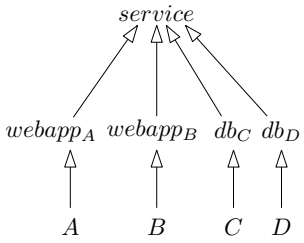

Fig. 3. The tree structure expressing dependencies between components, derived from the three-tier architecture of the cloud-deployed service in Figure 2.

dependency tree. The resulting *compositional verification task* (i.e., sequence of basic verification steps) for our cloud-deployed service is given by

$$cvt = \langle A, webapp_A, B, webapp_B, C, db_C, D, db_D, service \rangle. \tag{3}$$

Each verification step of *cvt* is performed using the results of some or all of the previous steps as assumptions.

Compositional quantitative verification reduces the NFP analysis overheads significantly, and in [11] we show that it can be used for continual NFP verification in certain scenarios associated with cloud-based systems.

3.3 Incremental Verification

Services deployed on the cloud operate in a dynamic environment where rapid changes to its structure are commonplace. Service components are added, removed or modified to scale according to demand or might fail unexpectedly. While compositional verification broadens the range of systems to which formal verification techniques can be applied effectively, service changes require reverification and make it difficult to maintain up to date verification results.

Incremental verification techniques [11, 21] are used to identify and execute a minimal sequence of reverification steps after a change in a component-based system. We describe two change scenarios typically encountered by an administrator of a cloud-deployed service with the dependency tree in Figure 3, and present at a high level the reverification steps performed by incremental verification to the changes in these scenarios. These scenarios are described in detail in [21], and assume that the service has been completely verified by a compositional verification task such as (3).

Scenario 1 In response to an increase in service workload, the administrator deploys a new instance of the database function on the physical server modelled by C. This results in an updated component model $db_C{}'$ modelling two database instances. The sequence of reverification steps that may need to be carried out is

$(db', service)$,

since the *service* model uses assumptions obtained from the verification of db', according to the dependencies in Figure 3. It is often the case that changes in a component result in improved reliability and satisfy local NFPs that allow the reverification to terminate early. In this scenario, the reliability of the database functions on C is improved with the additional instance and the reverification process halts without reverifying the *service* component model.

Scenario 2 Suppose that the physical server represented by the component model A has an unexpected failure in one of its hard disks, causing a degradation in reliability. This change results in an updated component model A' that reflects a reduction in the number of operation hard disks. The sequence of reverification steps that may need to be carried out is

$$(A', webapp_A, service),$$

since $webapp_A$ uses the reliability assumptions of A'. Furthermore, because the reverification of $webapp_A$ is required, the *service* model must also be included in the reverification sequence, so that the impact on the reliability NFP of the entire service may be determined.

A range of experiments were carried out in [21] to evaluate the effectiveness of incremental verification. The continual NFP verification scenarios explored in these experiments consisted of a number of service changes, starting from a configuration identical to the cloud-deployed service described here. They included the comparison of incremental verification and compositional verification for services with: (i) increasingly larger numbers of web and application instances, adding one instance at a time; (ii) increasingly larger numbers of database instances; and (iii) failed hard disks in a server causing a degradation in the server reliability for increasingly larger service sizes, obtained through adding between one and 20 web, application and database instances. The performance results of these experiments showed that incremental verification can provide significant reductions in the time required for the continual verification of cloud-deployed service NFPs after changes.

3.4 Practical Aspects

Using our continual NFP 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, enabling the adoption of the approach in mainstream cloud administration practice is a significant challenge.

To address this challenge, we developed two domain-specific languages (DSLs) that administrators of cloud resources can use to specify the initial structure of their multi-tier cloud-deployed services and a range of changes under which their reliability NFPs should be analysed, respectively [34]. The first DSL was implemented as an Eclipse plugin, using the EuGENia [28] front-end for the Eclipse

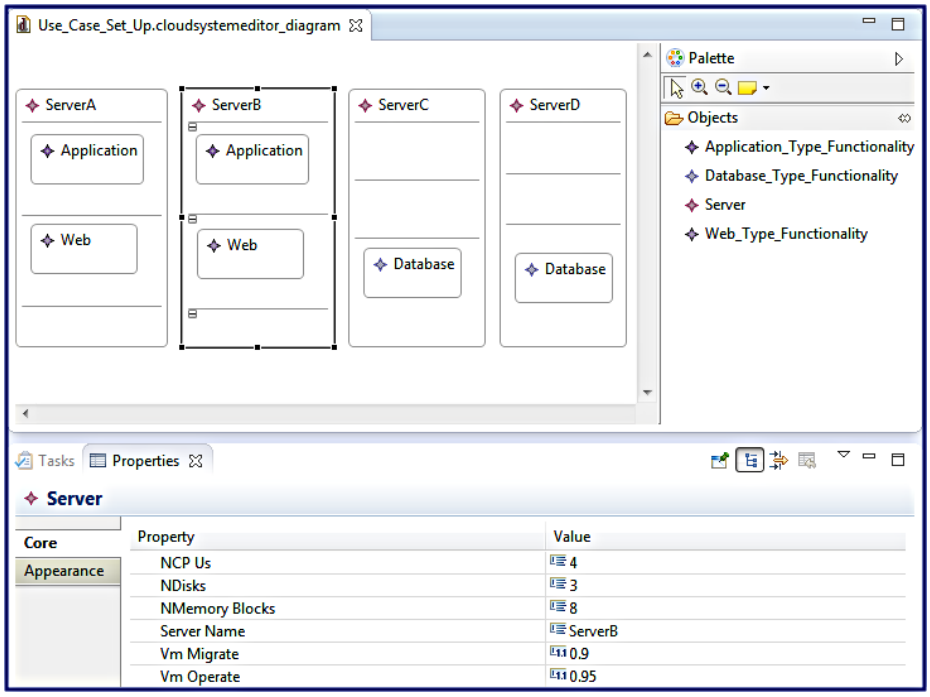

Fig. 4. Graphical DSL supporting the modelling of cloud-deployed services for continual NFP verification, used to model the three-tier service from Fig. 2.

Graphical Modelling Framework (GMF) to assemble the graphical service architecture modeller in Fig. 4. The integration of this graphical DSL with our incremental verification tool from [21] offers cloud administrators a reliability NFP verification framework that hides the complexity of the formal verification techniques underpinning its operation.

The second domain-specific language, implemented as a Java Swing application synchronised with the graphical DSL in Fig. 4, allows cloud administrators to specify system changes and analyse reliability NFPs for a range of "what if" scenarios. As illustrated in Fig. 5, the descriptions of these scenarios are assembled through the selection of keywords and phrases from a controlled vocabulary, along the lines of our related work from [2] and the results from [20] that it is based on.

4 Continual Verification of Service-Based Systems

This section presents an approach to using continual NFP verification for the benefit of cloud-service users. In this approach, runtime quantitative verification is used to drive the dynamic selection of the components of service-based systems (SBSs). Built through the integration of third-party services deployed on

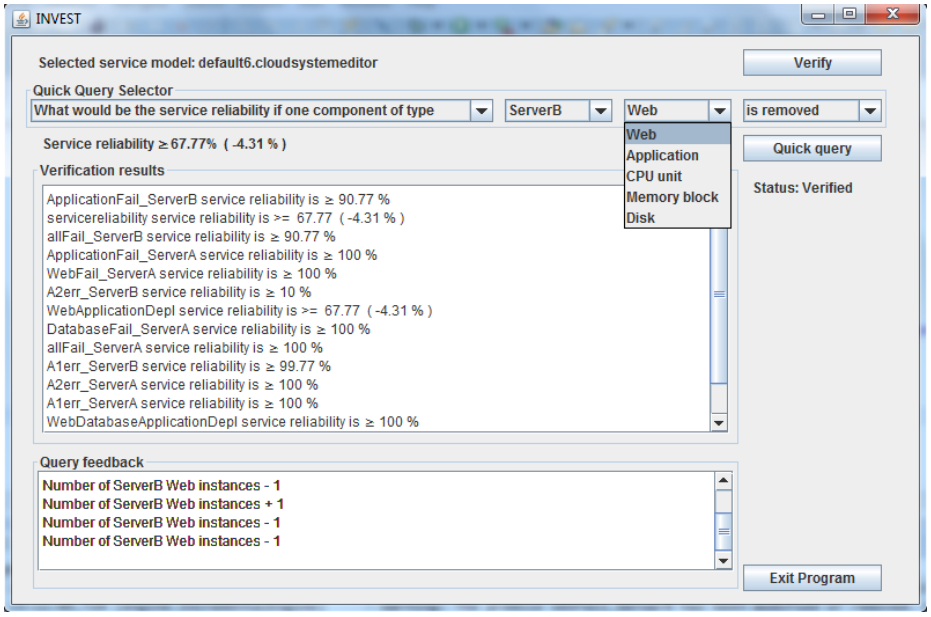

Fig. 5. DSL for the specification of cloud-deployed service changes/"what if" scenarios for which cloud administrators want to explore NFR-compliance.

cloud datacentres and accessed over the Internet, SBSs are increasingly used in business-critical and safety-critical applications where they must comply with strict non-functional requirements. Self-adaptive SBSs achieve this 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. Fig. 6 depicts our self-adaptive SBS architecture, in which this concrete service selection is underpinned by runtime quantitative verification.

Originally proposed in [2] and further extended in [1, 4, 9], the self-adaptive SBS architecture from Fig. 6 comprises $n \geqslant 1$ operations performed by remote third-party services. The $n \geq 1$ *intelligent proxies* in this architecture interface the SBS 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. Whenever an instance of the i-th proxy is created, it 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 n 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 Markovian model of the SBS workflow, and uses the online learn-

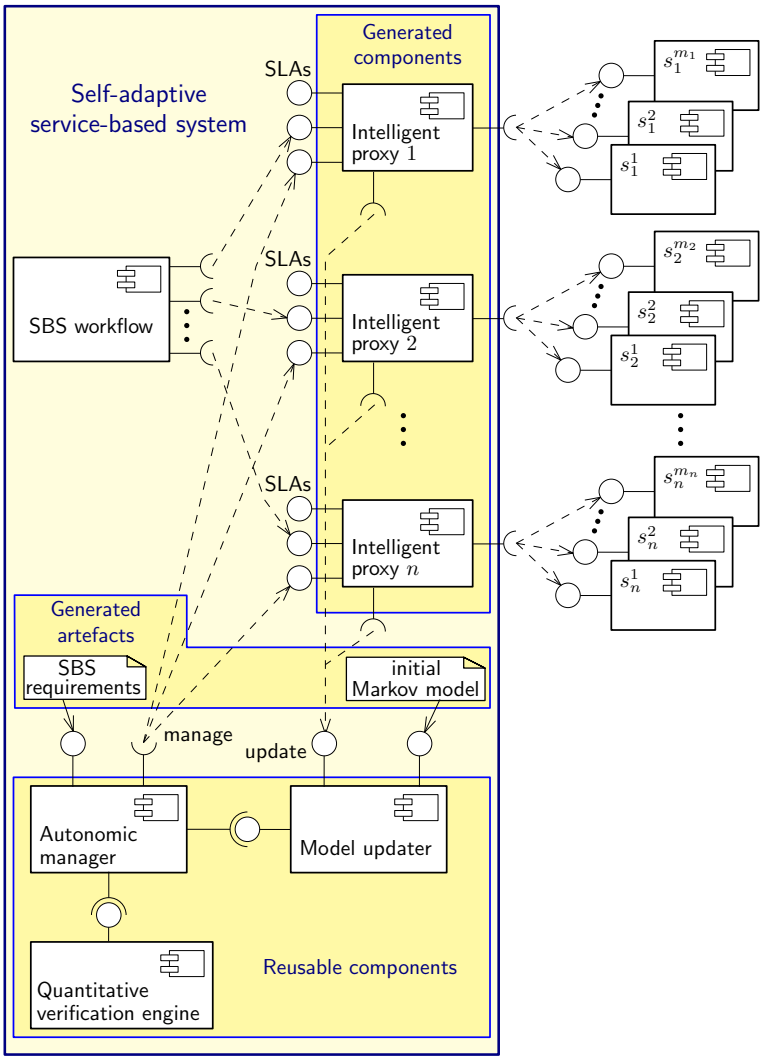

Fig. 6. Self-adaptive service-based system whose dynamic selection of cloud services is driven by continual NFP verification, adapted from [4, 9].

ing techniques from [3, 6] to adjust the transition probabilities of the model in line with these proxy notifications. Finally, the up-to-date Markovian model maintained by the model updater is used by an *autonomic manager* that performs runtime quantitative verification to select the service combination used by the n proxies so that it satisfies the SBS requirements with minimal cost at all times. Accordingly, the proxies, model updater and autonomic manager with its quantitative verification engine implement the continual NFP verification loop from Fig. 1.

As in the case of the continual NFP verification framework presented in Section 3, two key challenges must be addressed to ensure that the self-adaptive SBS architecture in Fig. 6 is practical and effective. First, SBS developers should be able to build instances of the architecture without requiring special training in formal methods. To address this first challenge, our work from [3, 6] 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.

The second challenge is related to the overheads associated with the quantitative verification of non-functional properties. The quantitative verification engine used by this framework is based on an embedded version of the PRISM probabilistic model checker [31], and the experiments in [3, 6] show that it can handle well the necessary "on the fly" analysis 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 workflows.

5 Conclusion

A key characteristic of cloud-based systems is their ability to evolve, e.g., by scaling their resources up and down in response to variations in workload. When such evolving systems are used to provide business-critical or safety-critical services, their non-functional properties (NFPs) must be verified continually.

We summarised the results of a research project that has explored ways in which a principled technique called runtime quantitative verification could be used to support continual NFP verification in cloud-based systems. In doing so, we described two continual verification approaches that can be employed by providers and users of cloud-deployed services, respectively. These approaches have been used successfully in a number of case studies, e.g. [1, 4, 6, 9, 11, 21]. Nevertheless, significant research is still needed to extend their applicability to new scenarios, to improve their scalability, and to identify and address their limitations. In particular, we are currently working on extending the applicability of our results to support NFP analysis when critical systems are ported between clouds [37] or deployed across multiple clouds [19].

Acknowledgements

This work was partly supported by the UK Engineering and Physical Sciences Research Council grant EP/H042644/1.

References

1. Calinescu, R., Ghezzi, C., Kwiatkowska, M., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Communications of the ACM 55(9), 69–77 (September 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: 2nd ACM/SPEC Intl. Conf. on Performance Engineering. pp. 505–510 (2011)
4. Calinescu, R., Johnson, K., Rafiq, Y.: Using continual verification to automate service selection in service-based systems. Tech. Rep. YCS-2013-484, Dept. of Computer Science, University of York (2013), http://www.cs.york.ac.uk/ftpdir/reports/2013/YCS/482/YCS-2013-484.pdf
5. Calinescu, R., Kikuchi, S., Kwiatkowska, M.: Formal methods for the development and verification of autonomic IT systems. In: Cong-Vinh, P. (ed.) Formal and Practical Aspects of Autonomic Computing and Networking: Specification, Development and Verification. pp. 1–37. IGI Global (2010)
6. Calinescu, R., Rafiq, Y.: Using intelligent proxies to develop self-adaptive service-based systems. In: 7th Intl. Symp. on Theoretical Aspects of Software Engineering. pp. 131–134 (2013)
7. Calinescu, R.: When the requirements for adaptation and high integrity meet. In: Proceedings of the 8th International Workshop on Assurances for Self-Adaptive Systems. pp. 1–4. ASAS '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2024436.2024438
8. Calinescu, R.: Emerging techniques for the engineering of self-adaptive high-integrity software. In: Camara, J., Lemos, R., Ghezzi, C., Lopes, A. (eds.) Assurances for Self-Adaptive Systems, Lecture Notes in Computer Science, vol. 7740, pp. 297–310. Springer Berlin Heidelberg (2013), http://dx.doi.org/10.1007/978-3-642-36249-1_11
9. Calinescu, R., Johnson, K., Rafiq, Y.: Developing self-verifying service-based systems. In: 28th Intl. IEEE/ACM Conference on Automated Software Engineering (2013), to appear
10. Calinescu, R., Kikuchi, S.: Formal methods @ runtime. In: Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems, Lecture Notes in Computer Science, vol. 6662, pp. 122–135. Springer (2011)
11. 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)
12. Calinescu, R., Kwiatkowska, M.: CADS*: Computer-aided development of self-* systems. In: Chechik, M., Wirsing, M. (eds.) Fundamental Approaches to Software Engineering (FASE 2009). Lecture Notes in Computer Science, vol. 5503, pp. 421–424. Springer (March 2009), http://qav.comlab.ox.ac.uk/papers/fase09.pdf
13. Calinescu, R., Kwiatkowska, M.Z.: Using quantitative analysis to implement autonomic IT systems. In: Proceedings of the 31st International Conference on Software Engineering, ICSE 2009. pp. 100–110. IEEE Computer Society (2009)
14. Epifani, I., Ghezzi, C., Mirandola, R., Tamburrelli, G.: Model evolution by run-time adaptation. In: Proc. 31st Intl. Conf. Software Engineering (ICSE'09). pp. 111–121 (2009)

15. Etessami, K., Kwiatkowska, M., Vardi, M., Yannakakis, M.: Multi-objective model checking of Markov decision processes. In: TACAS'07. pp. 50–65. Springer (2007)
16. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proceedings of the 33rd International Conference on Software Engineering. pp. 341–350. IEEE Computer Society (2011)
17. Filieri, A., Ghezzi, C., Tamburrelli, G.: A formal approach to adaptive software: continuous assurance of non-functional requirements. Formal Asp. Comput. 24(2), 163–186 (2012)
18. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) Formal Methods for Eternal Networked Software Systems (SFM'11). LNCS, vol. 6659, pp. 53–113. Springer (2011)
19. Global Inter-Cloud Technology Forum: Use cases and functional requirements for inter-cloud computing, GICTF White Paper (August 2010), http://www.gictf.jp/doc/GICTF_Whitepaper_20100809.pdf
20. Grunske, L.: Specification patterns for probabilistic quality properties. In: Robby (ed.) 30th International Conference on Software Engineering (ICSE 2008). pp. 31–40. ACM (2008)
21. Johnson, K., Calinescu, R., Kikuchi, S.: An incremental verification framework for component-based software systems. In: Proc. 16th Intl. ACM Sigsoft Symposium on Component-Based Software Engineering. pp. 33–42 (2013)
22. Jung, G., Hiltunen, M., Joshi, K., Schlichting, R., Pu, C.: Mistral: Dynamically managing power, performance, and adaptation cost in cloud infrastructures. In: Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on. pp. 62–73 (2010)
23. Katoen, J.P., Khattri, M., Zapreev, I.S.: A Markov reward model checker. In: Quantitative Evaluation of Systems. pp. 243–244. IEEE Computer Society, Los Alamitos (2005)
24. Katoen, J.P.: Model checking meets probability: A gentle introduction. In: Engineering Dependable Software Systems. pp. 1–29. NATO Science for Peace and Security Series - D, IOS Press (2013)
25. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. IEEE Computer Journal 36(1), 41–50 (January 2003)
26. Kikuchi, S., Aoki, T.: Evaluation of operational vulnerability in cloud service management using model checking. In: Service Oriented System Engineering (SOSE), 2013 IEEE 7th International Symposium on. pp. 37–48 (2013)
27. Kikuchi, S., Tsuchiya, S.: Configuration procedure synthesis for complex systems using model finder. In: Engineering of Complex Computer Systems (ICECCS), 2010 15th IEEE International Conference on. pp. 95–104 (2010)
28. Kolovos, D., Rose, L., Abid, S., Paige, R., Polack, F., Botterweck, G.: Taming emf and gmf using model transformation. In: Petriu, D., Rouquette, N., Haugen, . (eds.) Model Driven Engineering Languages and Systems, Lecture Notes in Computer Science, vol. 6394, pp. 211–225. Springer Berlin Heidelberg (2010), http://dx.doi.org/10.1007/978-3-642-16145-2_15
29. Kwiatkowska, M.: Quantitative verification: Models, techniques and tools. In: Proc. 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE). pp. 449–458. ACM Press (September 2007)
30. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (eds.) Formal Methods for the Design of Computer,

Communication and Software Systems: Performance Evaluation (SFM'07). LNCS (Tutorial Volume), vol. 4486, pp. 220–270. Springer (2007)
31. 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)
32. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: TACAS'10. pp. 23–37. Springer (2010)
33. Li, J., Chinneck, J., Woodside, M., Litoiu, M., Iszlai, G.: Performance model driven qos guarantees and optimization in clouds. In: Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. ICSE Workshop on. pp. 15–22 (2009)
34. Pehlivanov, S.: Modelling of Multi-Tier Applications Deployed on Cloud Computing Infrastructure. Master's thesis, University of York (September 2013)
35. Perez-Palacin, D., Calinescu, R., Merseguer, J.: log2cloud: log-based prediction of cost-performance trade-offs for cloud deployments. In: Shin, S.Y., Maldonado, J.C. (eds.) Proceedings of the 28th Annual ACM Symposium on Applied Computing. pp. 397–404. ACM (2013)
36. Perez-Palacin, D., Mirandola, R., Merseguer, J.: Qos and energy management with petri nets: A self-adaptive framework. J. Syst. Softw. 85(12), 2796–2811 (Dec 2012), http://dx.doi.org/10.1016/j.jss.2012.04.077
37. Silva, G.C., Rose, L.M., Calinescu, R.: Towards a model-driven solution to the vendor lock-in problem in cloud computing. In: 5th IEEE International Conference on Cloud Computing Technology and Science (2013), to appear
38. Younes, H.L.S.: Ymer: A statistical model checker. In: Etessami, K., et al. (eds.) Computer Aided Verification, LNCS, vol. 3576, pp. 429–433. Springer (2005)