Information and Software Technology xxx (xxxx) xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Formal Quality of Service assurances, ranking and verification of cloud deployment options with a probabilistic model checking method

Petar Kochovski[a,b], Pavel D. Drobintsev[b], Vlado Stankovski[a,*]

[a] Faculty of Civil and Geodetic Engineering, University of Ljubljana, Ljubljana, Slovenia
[b] Institute of Computer Science and Technology, Peter the Great St. Petersburg Polytechnic University, St. Petersburg, Russian Federation

ARTICLE INFO

Keywords:
Cloud
Fog
Edge
Software engineering
Decision-making
Equivalence classes
Probabilistic model checking

ABSTRACT

Context: Existing software workbenches allow for the deployment of cloud applications across a variety of Infrastructure-as-a-Service (IaaS) providers. The expected workload, Quality of Service (QoS) and Non-Functional Requirements (NFRs) must be considered before an appropriate infrastructure is selected. However, this decision-making process is complex and time-consuming. Moreover, the software engineer needs assurances that the selected infrastructure will lead to an adequate QoS of the application.

Objective: The goal is to develop a new method for selection of an optimal cloud deployment option, that is, an infrastructure and configuration for deployment and to verify that all hard and as many soft QoS requirements as possible will be met at runtime.

Method: A new Formal QoS Assurances Method (FoQoSAM), which relies on stochastic Markov models is introduced to facilitate an automated decision-making process. For a given workload, it uses QoS monitoring data and a user-related metric in order to automatically generate a probabilistic model. The probabilistic model takes the form of a finite automaton. It is further used to produce a rank list of cloud deployment options. As a result, any of the cloud deployment options can be verified by applying a probabilistic model checking approach.

Results: Testing was performed by ranking deployment options for two cloud applications, File Upload and Video-conferencing. The FoQoSAM method was compared to a baseline Analytic Hierarchy Process (AHP). The results show that the first ranked cloud deployment options satisfy all hard and at least one of the soft requirements for both methods, however, the FoQoSAM method always satisfies at least an additional QoS requirement compared to the baseline AHP method.

Conclusions: The proposed new FoQoSAM method is appropriate and can be used in decision-making when ranking and verifying cloud deployment options. Due to its practical utility it was integrated into the SWITCH workbench.

1. Introduction

With the emergence of the Internet of Things (IoT), the potential for various smart applications is radically increasing in practically all areas [1–3]. Microservices architecture [4] is an approach to software development in which a smart cloud application can be built as a suite of modular services. A single microservice is understood as a small, independently versioned and scalable customer-focused service with specific business goals, which communicates with clients and other services over standard protocols with well-defined interfaces. A microservice has a firm module boundary, so that it can be written in a programming language of the software engineer's choice. Microservices are usually deployed in Virtual Machines (VM) or containers [5], so that they can be seamlessly managed across the entire computing spectrum, starting from cloud data centres to Fog (e.g. routers, micro-servers) and Edge (e.g. battery-powered, automated cars) computing nodes [6,7].

Moreover, the development of various smart applications is supported with new approaches, methodologies and workbenches that promise radical improvements of the software life-cycle [8–10]. This is achieved through a clear separation of the application design, deployment and operational (runtime) phases. Advanced software engineering workbenches support the composition of applications from existing microservices, discovery and negotiations of computing resources with Infrastructure-as-a-Service (IaaS) cloud providers, Service Level Agreements, deployment and runtime orchestration in multi-cloud environments [11–15]. The software engineering process is cloud agnostic in the sense that a cloud application can be designed and developed before an IaaS offer is selected [16].

* Corresponding author.
E-mail addresses: vlado.stankovski@fgg.uni-lj.si, vlado@stankovski.net (V. Stankovski).

https://doi.org/10.1016/j.infsof.2019.01.003
Received 7 April 2018; Received in revised form 8 January 2019; Accepted 9 January 2019
Available online xxx
0950-5849/

Please cite this article as: P. Kochovski, P.D. Drobintsev and V. Stankovski, Formal Quality of Service assurances, ranking and verification of cloud deployment options with a probabilistic model checking method, Information and Software Technology, https://doi.org/10.1016/j.infsof.2019.01.003

This study focuses on the deployment phase, when the software engineer has to select an appropriate IaaS offer and deploy the application or one of its parts, such as a microservice. The Quality of Service (QoS) and Non-Functional requirements (NFRs) of smart applications usually vary greatly, which must be addressed in the deployment phase. For example, for a specific microservice and intended workload, it may be necessary to obtain a high-speed processor, a large amount of memory, low latency, high bandwidth, a specific geolocation for deployment, achieve low operational cost or similar. The problem of selecting an appropriate computational resource, that is, an infrastructure and configuration scheme where a microservice will be deployed and run is not at all unimportant. The more criteria used, the higher the complexity of the decision-making process [17,18]. It is therefore necessary to design new methods that can support the deployment of microservices, and which may be integrated into existing workbenches, such as Juju, Fabric8 or SWITCH [19].

Therefore, the goal of this work is to develop a new method that can be used by software engineers to automatically rank a list of cloud deployment options for their microservices. In addition, the goal is to include a verification step, thus providing assurances that all hard and as many soft QoS requirements as possible will be met at runtime. The hypothesis of this work contends that formal QoS assurances in the cloud application deployment phase can be provided with a new probabilistic decision-making method. Our approach relies on the theory and practice of stochastic Markov models [20,21]. Hence, the objectives of this study have been set out to develop:

- a Markov-based probabilistic decision-making approach, which offers the software engineer a set of optimal IaaS that would satisfy a specific QoS of the application,
- an equivalence classification approach of available cloud deployment options,
- a monitoring and context input method for the development of a probability model, and
- a design and implementation of the new method to be included in a software engineering tool.

The new Formal QoS Assurances Method (FoQoSAM) is presented in detail in this work. Section 2 presents related work and previous contributions from academia and the software engineering industry. Section 3 provides a thorough description of the new deployment methodology, which includes a formal definition of the equivalence classification, a description of QoS and NFR attributes and the actual probabilistic model. The developed components that form the architecture of the proposed model are described in Section 4. In addition, the results of the experimental tests of the described model are explained in Section 5. Section 6 discusses the experimental results and intended for work in the future. Finally, Section 7 concludes the paper and discusses future plans, including improvement of the developed model.

2. Related work

Existing studies have investigated the selection of optimal IaaS providers with respect to the expected QoS, for example, studies related to load balancing [22–25], resource management and allocation [26–29], resource provisioning [30–32] or service deployment and management systems [15,33].

As a result of the large array of different NFRs that can be taken into account, the reviewed studies proposed multi-objective approaches for various deployment scenarios on cloud infrastructures. For instance, Karim et al. [34], Garg et al. [35] and Gonçalves et al. [36] describe deterministic approaches based on AHP, which has been the most frequently used method for multi-criteria decision-making in the period 2000–2014 [37]. The AHP takes into account the users QoS requirements in order to rank and select cloud deployment options. However, the AHP can be computationally impractical in cases where it is used to compare a large number of parameters, as it will generate a large number of embedded decisions. For example, if AHP is used to rank 500 available cloud deployment options by comparing 20 non-functional attributes, it will generate a decision tree with 10,000 nodes. Zheng et al. [38] developed a framework for delivering optimal cloud service selection, which also ranks the services according to the QoS they must provide. However, in order to rank the cloud deployment options, their framework only utilizes past data usage based on two network-level metrics (throughput and response time). Guerrero et al. [39] presented an approach for container resource allocation, which is based on the Non-dominated Sort Genetic Algorithm (NSGA-II); however, possibilities to include or exclude of QoS attributes are limited. In contrast to the above methods, our new FoQoSAM method is designed to take into account the current deployment context of the application, such as user preferences for geographic availability and estimated workload.

Moreover, all of the methods described above are deterministic. They do not a provide probabilistic evaluation for the success of the QoS, which is important to achieve for critical business applications. Multiple studies of different fields use the Markov Decision Processes (MDP) to deliver decision-making results in non-deterministic cases, where unpredicted situations may arise. For instance, MDP has been used to facilitate an optimal treatment for patients [40] that has been tailored to individual patients, optimal charging of specific electrical vehicles [41] and similar. Moreover, it has been applied in the domain of artificial intelligence and reinforcement learning [42].

In this context, MDP is also suitable for application in the cloud-computing domain, to address its non-deterministic properties. For instance, Yang et al. [43] present an MDP-based method, whose purpose is to choose a deployment option that provides optimal performance to applications. In addition, Su et al. [44] also proposed an MDP-based scheduling mechanism, which supports a trade-off between three metrics (accuracy, data usage and computational overhead) by implementing an iterative decision-making approach. Nevertheless, to the best of our knowledge, the use of MDP to assure high QoS of a deployed software component in the context of containers has not yet been considered.

MDP also allows formal verification of the correctness for the placement of a deployment decision. Llerena et al. [45] developed a technique for analysing the effects of probabilistic values perturbations by verifying the reachability properties of MDP models. The following studies [46,47] have applied such techniques to the problem of the horizontal scaling of VMs. However, their computational complexity prevents the integration of such methods in mainstream software engineering practices and this problem has not been addressed adequately in the existing studies.

The present work complements the above efforts by focusing on aspects related to dependability, computational complexity and formal assurances, which are required for a method to find applications in mainstream software engineering. Our method aims at automating the process of cloud deployment option selection, which is (1) optimal given the specific requirements and usage context and (2) verified through a stochastic approach, which is new in the domain. The new method supports the use of various NFRs, continuous accumulation of metrics during operation, and includes a mechanism to deal with the computational complexity of the MDP model.

3. Decision-making process for microservice deployment

Our new method was developed to automatically rank all candidate deployment options by taking into account the NFRs, and the potentially unique usage context of the microservice. Additionally, we implemented a formal approach, which is used to provide assurances to the software engineer that the NFRs will be satisfied at runtime.

The process is illustrated in Fig. 1. Here, we proceed by explaining the individual steps involved.

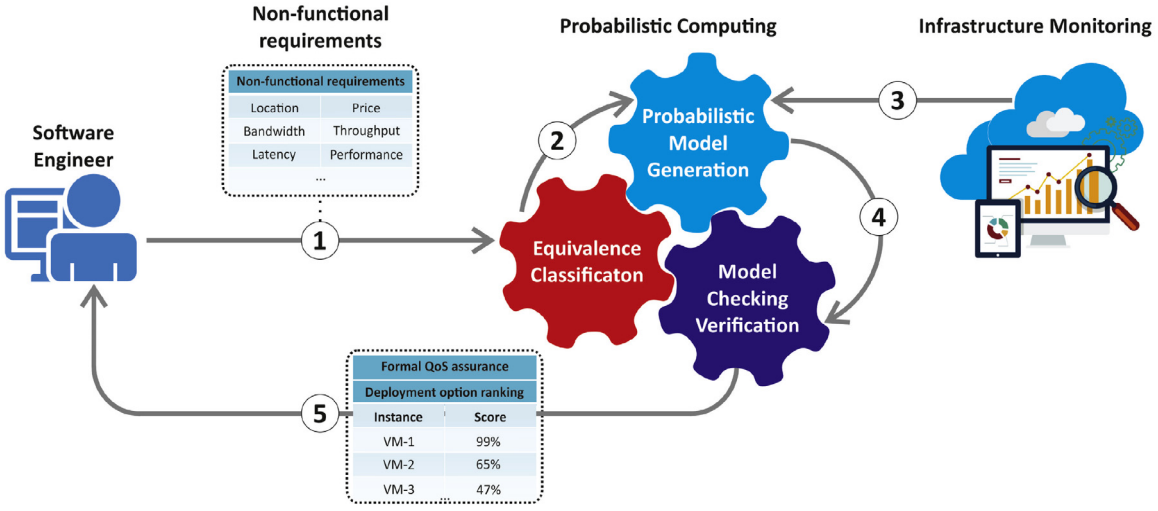

Fig. 1. Decision-making process for microservice deployment.

1. *Selection of NFRs:* In this step, the software engineer selects all important NFRs. Additionally, threshold values for any selected QoS metrics are defined.
   The software engineer can choose from a variety of QoS metrics (e.g. throughput, latency, performance, resource utilisation, cost and similar) and NFRs are categorically specified (e.g. specific geolocation). This selection can differ significantly a microservice to microservice. In this step, the software engineer must also decide, which NFRs represent hard constraints and must be satisfied continuously during execution, and which NFRs are desirable, but not mandatory (soft constraints). Once the hard and soft constraints are defined, they are used in two distinctive steps of the automated decision-making process. The hard constraints are used as input parameters for the equivalence classification (second step), while the soft constraints are used in the Probabilistic Model Generation and verification stage (third step).
2. *Initial model generation and equivalence classification:* The goal of this step is to reduce the number of necessary computations when using the FoQoSAM method. All multi-criteria decision-making processes have a high degree of computational complexity, since MDP is known as a P-complete problem [48]. Due to its nature, we have to reduce the list of available deployment options that are provided as inputs to the method.
   In the beginning, a finite automaton is generated by using the list of all available cloud deployment options (e.g. 1000 available cloud deployment options). Every state of this initial model represents one distinctive deployment option. Then we follow an automated process in which the deployment options are classified into equivalence classes. For example, a specific equivalence class may contain all deployment options that contain a minimum of 8 CPUs, or all deployment options for which the CPU utilisation is less than 80%. The resulting equivalence classes are enumerated and further used to generate the probabilistic model.
3. *Probabilistic model generation*: The goal of this step is to generate a probabilistic model, which is a stochastic model of randomly changing systems. It incorporates: (1) the potential outcomes of the system (i.e. the different deployment options within the equivalence class) and (2) the probabilities that are assigned to the potential outcomes.
   In order to build the probabilistic model, various QoS metrics that represent the past and present performance status of the cloud deployment options are taken into account. These are collected and stored in a time-series database by using a multi-level monitoring system, which is part of the SWITCH workbench. The Probabilistic Model Generation method receives all relevant metrics, such as the utilisation percentage of the cloud deployment options' resources, throughput, latency, packet loss and other metrics. If available, Quality of Experience (QoE) metrics gathered from previous executions may also be used.
   Only those NFRs that are part of the hard and soft constraints are used in the process. The method then calculates ranking scores for all deployment options and ranks them.
4. *Model checking verification:* Following the generation of the probabilistic model, the obtained result is verified by using a model checking approach. This step verifies to what extent the NFRs are satisfied by the available deployment options within each equivalence class.
   Formal criteria are used to verify the results provided by the probabilistic model. As an illustration, in this step the software engineer can verify that the NFRs will be satisfied with a 95% probability rate for the first ranked cloud deployment option. The calculated value is an output of the probabilistic model and represents a formal assurance.
5. *Ranking and deployment of cloud deployment options:* The first ranked deployment option is automatically selected and the microservice is deployed.
   This step proceeds under the assumption that the received formal assurance for the first ranked cloud deployment option, i.e. the one with the highest score is acceptable to the software engineer.

With the specifications of hard and soft constraints and the expected formal assurance before microservice deployment, the process is fully automated and does not require any additional interactions with the software engineer. The individual steps of this process are further elaborated in the following sub-sections.

3.1. Non-functional requirements

The new FoQoSAM method is designed to operate with a potentially large number of NFRs. NFRs are expressed by means of specific attributes that many have real or categorical values. The authors [49] conclude that attributes such as: geographical location, CPU utilisation, availability and response time should be addressed to achieve high QoS of Big Data applications. Another study [50] addresses network-level metrics, such as bandwidth, throughput, jitter, power efficiency and cost as important attributes for their IoT applications.

Based on an analysis of important metrics, and in order to illustrate our new method, we have made a selection of NFR attributes (metrics) to be used in the decision-making process (see Table 1).

Table 1
NFR attributes used for experimentation.

NFR	Description
Throughput (Gb/s)	The rate at which data is transferred between two endpoints, without losses. It measures the quantity of data (TCP/UDP traffic) that a given microservice can successfully transfer per unit of time.
Latency (ms)	Latency is a metric (a.k.a. attribute) that represents the time required for a packet to be transferred across the network. In our case it is measured as the round-trip time for the package to reach a microservice and return to the client.
Packet loss (%)	Packet loss is a metric measured at the connection-point between the client and the running microservice.
QoE	Quality of Experience (QoE) is a metric calculated by using end-user satisfaction ratings. End-users rate the quality of the received service with grades within the range 1–5, where 1 is the worst and 5 is the best quality perceived.
CPU utilisation (%)	This is a performance metric that represents the sum of work handled by the CPU. CPU utilisation varies according to the workload of the microservice.
Memory utilisation (%)	This is a performance metric, which represents the amount of vRAM used by the deployed application.
Cost ($/month)	This is the monthly cost of using a cloud infrastructure.

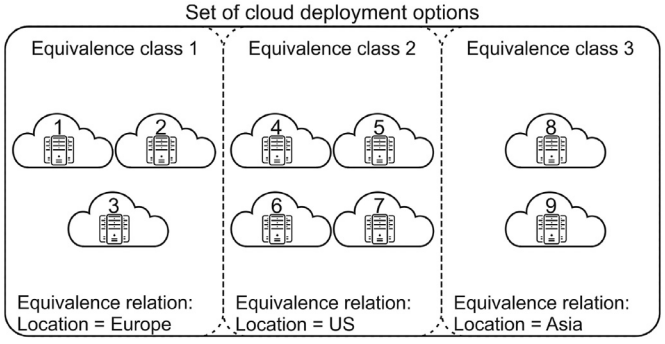

Fig. 2. Example classification of available cloud deployment options.

As many smart IoT-based applications are time-critical in nature, we also developed two time-critical microservices (File Upload and Video-conferencing) for experimentation. They are elaborated in detail in Section 5.1. The two microservices require flawless data transfer, without signal delays or packet loss. Hence, several network-level (e.g. throughput, response time, packet loss) and infrastructure-level attributes (e.g. cost, resource utilisation and QoE) are used in our experiments.

3.2. Forming NFR equivalence classes of cloud deployment options

Using an MDP-based approach to rank a potentially large number of cloud deployment options according to multiple constraints is a computationally intensive process. The number of necessary computations increases with the number of available deployment options. Here, we reduce the input list of available deployment options by forming equivalence classes based on the selected NFR constraints.

Equivalence classes are subsets of a large set of elements, which satisfy a precisely defined hard constraint that is also known as an equivalence relation. An equivalence relation is a strict mathematical definition that divides any set into subsets. Any equivalence system must satisfy the following rules: (1) the equivalence classes are disjoint; (2) all element pairs within the same equivalence class are equivalent to one another; (3) any two elements that do not belong to the same equivalence class are not equivalent to one another. Fig. 2 depicts equivalence classification within a set of cloud deployment options, where the location of the deployment options is used as an equivalence relation. As a result, the set is divided into three equivalence classes, which are composed of deployment options that are located in three regions: Europe, the USA and Asia.

In order to define equivalence classes formally, we first define two sets: a set of NFRs and a set of deployment options. The NFR set is a set of attributes' sets $NFR=\{\{nr_{00}, nr_{01}, \dots, nr_{0n}\}, \dots, \{nr_{m0}, nr_{m1}, \dots, nr_{mn}\}\}$, where *nr* represents a specific non-functional requirement (attribute), for example: $NFR = \{\{vCPU = 1, location = Europe, latency = 30\}, \{vCPU = 1, location = Europe, latency = 45\}, \{vCPU = 1, location = USA, latency = 200\}\}$. The set of cloud deployment options is defined as $I = \{inf_0, inf_1, \dots, inf_m\}$, where *inf* is a deployment option, for example, $I = \{g1-small-EU, n1-standard1-EU, n1-standard1-USA\}$.

The relationship between the elements of the NFR set and the elements of the deployment options' set is a bijective function: f: $NFR \rightarrow I$, where: $inf_m = f(nr_{m0}, nr_{m1}, \dots, nr_{mn})$. In other words, each attribute set from the *NFR* collection is mapped to exactly one deployment option from set *I*. This basically means that each deployment option is related to a unique set of NFRs. For instance, the sets presented in the previous paragraph can be associated to deployment options as follows: $f(vCPU = 1, location = Europe, latency = 30) = g1-small-EU$, $f(vCPU = 1, location = Europe, latency = 45) = n1-standard1-EU$, $f(vCPU = 1, location = USA, latency = 200) = n1-standard1-USA$.

The next step is to build an equivalence class. Let e be a desired deployment option that satisfies all hard constraints and $e \in I$. An equivalence class $[e]$ that is populated with deployment options equivalent to e, has to satisfy the following condition $[e] \subset I$. Such equivalence class is defined as: $[e] = \{inf \in I \mid inf \sim e\}$, where $inf \sim e$ is used to denote equivalence relation between two deployment options satisfying the condition $\forall i \exists j (nr_j^{inf} = nr_i^e)$. This means that within the scope of the equivalence class, the two deployment options e and *inf* will be equivalent if each attribute of e has a corresponding attribute of *inf* with equal value. For instance, an equivalence class of the sets presented in the paragraphs above that is based on $e = f(location = Europe)$ will be the following: $[e] = \{g1 - small - EU, n1 - standard1 - EU\}$.

Alongside the use of categorical values to form equivalence classes (such as the use of geographic location), we can also use threshold values to achieve the same, for example, by classifying all cloud deployment options that have CPU utilisation of less than 80% in the same class.

3.3. Decision-making method

Our new decision-making method relies on MDP, which is a stochastic process. It is a powerful mathematical framework for decision-making in dynamic environments, where the results are partly random and partly under the control of a decision maker [51]. Its non-deterministic nature allows the incorporation of multiple potential system behaviours in a single model. This characteristic allows the MDP in our case to perform multiple simulations from each state and aims at retrieving the optimal cloud deployment option in the model. MDP is suitable for use because it is able to: (1) autonomously select one of a number of possible events, situations or actions (e.g. better CPU utilisation or higher/lower QoS); (2) implement a utility function, which is calculated at each step of the process and generates a score for each cloud deployment option; (3) verify the behaviour of the model in a specific moment in the future, thus providing for reliable decision-making. An MDP's output is hard to accurately predict, because it depends on variable input parameters, such as network throughput, latency, resource utilisation and other NFRs.

MDP is defined as a tuple *M=(S, A, P, R, γ)*, where:

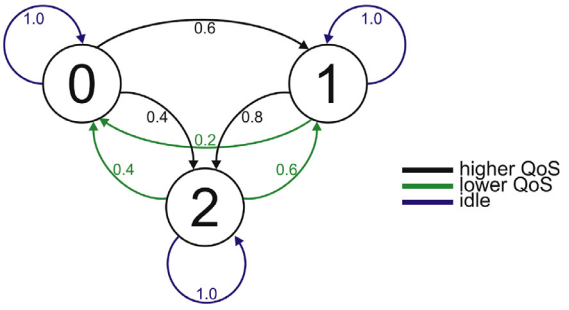

Fig. 3. Example of a probabilistic model composed of cloud deployment options within one equivalence class. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

- $S=\{S_0, ...,S_n\}$ is a finite set of states, which in our study it is a set of cloud deployment options;
- $A=\{a_0, ...,a_n\}$ is a finite set of actions, which in our study is shown as deployment of cloud application to deployment option with higher/lower QoS;
- $P=\{s_{t+1}=s' \mid s_t=s,\ a_t=a\}$ is the transition probability from state s at step t to state s' at the next step due to an action a;
- $R(s,s')$ is the expected reward received after transitioning from state s to state s', due to action a.
- γ is called a discount factor, and represents the difference in importance between current and future rewards. Its value is in the range 0–1.

The MDP model that was developed for this study follows the standard tuple from the definition above. Its design and implementation are thoroughly described in the following sections.

3.3.1. Probabilistic model

Probabilistic models can be fairly complex. In order to illustrate them, we provide a basic example. The probabilistic model is a finite automaton, which is built by following the MDP definition from the previous section. It is a necessary component used to generate the deployment options ranking results. Such models are built separately for every microservice, and can differ significantly due to the variability of the chosen input NFRs.

Each transition in the probabilistic model is a probabilistic choice over multiple next states. The goal of this study is to provide a ranking list of available cloud deployment options, hence, each state of the probabilistic model represents a different cloud deployment option. At runtime, the values of the NFR attributes can change, which is represented by probabilities. For example, a transition in the automaton may reach one deployment option with a 60% probability rate and another with a 40% probability rate, whereas at another moment in time these probabilities can be different.

The transitions between states occur as the result of different (and in some cases unpredicted) actions in the model, which imparts the non-deterministic behaviour of the model. The model used for illustration (see Fig. 3) implements two actions: a *deployment action*, which is responsible for selecting a deployment option and an *idle* action, which is activated if the current deployment option has the optimal set of NFR attributes (i.e. their values) requested by the software engineer.

In the probabilistic model, every action has a corresponding transition. Because the equivalence class might contain many deployment options that have different NFR (and QoS) attribute values, the deployment action could be shown as *higher QoS* for deploying an application to a deployment option with higher QoS, or *lower QoS* for selecting a deployment option with lower QoS. Every action results in a compatible transition between the states in the probabilistic model. For instance, a state that represents a deployment option with lower QoS will have valid transitions to all states that represent deployment options with a higher QoS. At the same time the same state may also have to transition to itself and to those states with lower QoS. Even though there are multiple transitions from one state, the result that the probabilistic model generates depends on the transition probabilities and state rewards. All transitions are mapped to a different probability value due to differences in past QoS values. Moreover, every state in the model is associated with a reward value. Thus, whenever the model decides to transition from one state to another, it chooses the state that provides a higher reward.

The reward and probability values represent the required input data for the utility function, because they are necessary for calculating the ranking score for every deployment option in the model. A more detailed description on how the probabilities and the rewards are calculated is presented in the next section.

Fig. 3 presents an example of a simple probabilistic model that our MDP-based method generates. It is composed of three deployment options, where every state represents a different deployment option with different network and computing performance. In this example, state 0 represents the deployment option with the lowest QoS and state 2 represents the deployment option with the highest QoS. The different types of transitions are represented with different colours. The transitions *more QoS* are shown in red. They represent transitions from a deployment option with lower QoS to a deployment option with higher QoS. The *lower QoS* transitions are shown in green and are the opposite of *more QoS*. The *idle* action transitions are shown in blue and are used to represent moments when the current state satisfies the QoS requirements, or there is simply no deployment option with a higher/lower QoS than the current one.

3.3.2. Calculating the model probabilities and rewards

When designing the probabilistic model, the main task is to calculate the probabilities for each transition and the rewards for each state. This section describes how to estimate the probabilities and rewards in our probabilistic model.

Essentially the transition probabilities must satisfy the Markov rule, which states that the probability of the future states of the process depends only upon the present state, and not on the states that preceded it in the past. Hence, the probabilities that predict the future behaviour of the model are only dependent on the current state of the model. For instance, when calculating the transition probability between two cloud deployment options A and B, the transition probability does not depend on the probability values that were necessary to reach deployment option A. In order to develop this probabilistic model, it is necessary to: (1) select an initial state of the probabilistic model, (2) calculate the transition probabilities between the states in the model, (3) calculate the reward values that are related to each state in the model.

The transition probabilities are depicted as layers in a probabilistic decision tree, as shown in Fig. 4 and explained as follows.

(A) The first level in the decision tree is necessary to determine the initial state of the probabilistic model. The result of this probability estimation is necessary, because it provides an initial state for the MDP. The transitions of the initial action are equally probable, and they allow the probabilistic model to commence the MDP from any available deployment option within the equivalence class. Hence, this is calculated using the following equation $P_1 = \frac{1}{N_{tran}}$, where N_{tran} is the amount of transitions of the same action from one state.

(B) The second level in the decision tree is necessary to calculate the transition probabilities between the states within the model. These probability values are used to compare the deployment options NFRs (and QoS) between each other. It can determine whether the microservice would achieve better quality in another state that can be reached from the current one in one transition. The probability values of this level are calculated using the following equation $P_2 = \frac{N_{chosen}}{N_{listed}}$, where N_{chosen} is the number of times

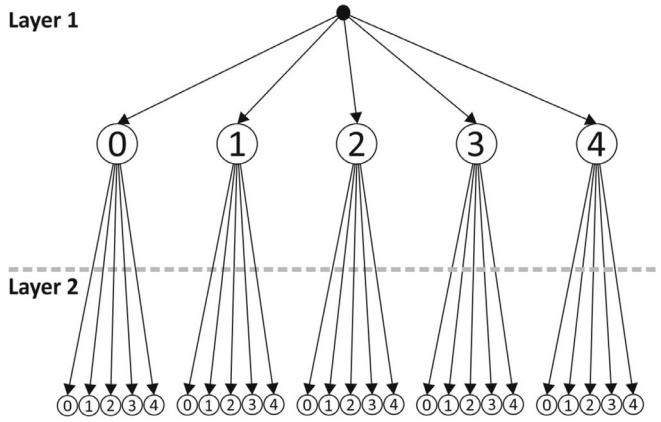

Fig. 4. Probabilistic decision tree.

a deployment option has been chosen for that specific type of software component and the N_{listed} is the number of times the deployment option has been listed in an equivalence class.

In general, if the probabilistic model has n amount of states, the transition probabilities can be estimated by using the following expression: $P_{ij} = \frac{P(P2_{ij} \mid P1_{ij}) \cdot P1_{ij}}{\sum_{k=1}^{n} P1_{ij} P2_{ij}}$, where $P(P2_{ij} \mid P1_{ij})$ is the probability that $P2_{ij}$ occurs, given that $P1_{ij}$ has occurred.

After estimating the probability values for each transition, it is necessary to estimate the reward values for reaching the states within the model. The reward values are scalar values, which are calculated according to Algorithm 1. They depend on the data acquired from the monitoring measurements and historical data from prior decisions. The algorithm inspects if the states of the model satisfy the soft constraints, which have been set by the software engineer. According to the algorithm, the more soft constraints are violated by a deployment option, the lower the reward for reaching the deployment option within the probabilistic model.

Algorithm 1 Algorithm for calculating the reward values in the probabilistic model.

1: ***Input parameters:*** *Network Throughput (NT), Network Latency (NL), Packet Loss (PL), CPU utilisation (CPU), Memory utilisation (MU), Quality of Experience (QoE), Cost(C), Network Throughput Threshold (NT_T),Network Latency Threshold (NL_T), Packet Loss Threshold (PL_T), CPU utilisation Threshold (CPU_T), Memory utilisation Threshold (MU_T), Quality of Experience Threshold (QoE_T), Cost Threshold (C_T), size - number of cloud deployment options*
2: ***Output parameters:*** *rewards*
3: $count \leftarrow 1$
4: **for** each i in $size$ **do**
5: **if** $NT < NT_T$ AND $NL > NL_T$ AND $PL > PL_T$ AND $CPU > CPU_T$ AND $MU > MU_T$ AND $QoE < QoE_T$ $C > C_T$ **then**
6: $rewards_i \leftarrow 0$
7: **else**
8: **if** NT $< NT_T$ **then** $count \leftarrow count + 1$
9: **if** NL $> NL_T$ **then** $count \leftarrow count + 1$
10: **if** PL $> PL_T$ **then** $count \leftarrow count + 1$
11: **if** CPU $> CPU_T$ **then** $count \leftarrow count + 1$
12: **if** MU $> MU_T$ **then** $count \leftarrow count + 1$
13: **if** QoE $< QoE_T$ **then** $count \leftarrow count + 1$
14: **if** C $> C_T$ **then** $count \leftarrow count + 1$
15: $rewards_i \leftarrow \frac{1}{count}$
16: **end if**
17: $count \leftarrow 1$
18: **end for**

To calculate the ranking score for each deployment option, the method implements utility function, which utilizes the transition probabilities and reward values as input parameters. The utility function has the following recursive form: $u(S) = r(S) + \gamma \max_a \sum_{S'} P(S'|a, S)u(S')$, where *r(S)* represents the reward value for reaching a state, *P(S′|a,S)* is the transition probability value for reaching state *S′* from state *S* due to action *a*, and *P(S′|a,S)u(S′)* represents the future, discounted rewards. The results that are generated by the utility function are presented in Section 5.

Although, the reward values in the current model are based on seven NFRs, the presented methodology is independent of the number and type of NFR attributes that can be used to prepare the rewards for each state. Thus, it can be extended with more attributes that could be implemented in the future.

3.4. Probabilistic model-checking

The probabilistic model that is generated by following this process has a finite number of states that allow the model's output and behaviour to be verified. The verification determines whether the system output conforms to the hard and soft NFR constraints given by the software engineer. We implemented an approach known as model checking that allows for automated analysis and verification of finite systems with random probabilistic behaviour.

In our case, the probabilistic model is verified by using a Probabilistic Computation Tree Logic (PCTL), which is a temporal logic that allows for the probabilistic quantification of system specifications. The primary function of PCTL is to verify whether the obligatory NFRs (hard constraints for NFRs) and maximum number of optional NFRs (soft constraints for NFRs) were satisfied.

Using PCTL the main verification criterion is as follows:

$P_{=1}[G(hard)\ \&\ F(soft)]$, where *hard* is a conjunction of all hard constraints:

$hard = hConstraint_1 \& hConstraint_2 \& \ldots \& hConstraint_m$; and *soft* is disjunction of all soft constraints:

$soft = sConstraint_1 | sConstraint_2 | \ldots | sConstraint_n$. The main verification criterion can be translated as follows:

- *Will the probability of satisfying all the hard constraints and at least one soft constraint within the equivalence class equal 100%?*

In order to narrow the verification, a maximisation criterion can be implemented and therefore the following description can be used:

$P_{=1}[G(hard)\ \&\ F[(soft)\ \&\ (nfrViolation \leq minViolations)]]$, where *nfrViolations* is the amount of NFRs that are not satisfied by the current cloud deployment option and *minViolations* is the minimum amount of NFR violations among all possible deployment options that are included in the model. In this case the verification criterion can be translated as follows:

- *Will the probability of satisfying all hard constraints and the maximum number of soft constraints within the equivalence class equal 100%?*

At the same time these PCTLs examine, whether the equivalence class is populated with deployment options that satisfy the hard constraints and whether the QoS probability score that was provided by the probabilistic model satisfies the soft constraints.

However, by implementing Model Checking Verification the system can be verified according to various criteria. In the following, we present three examples to indicate the wide scope of verification criteria that could be used by the software engineer to assure that the system satisfies the required NFR standards:

- *What is the probability that higher/lower QoS would be achieved, given that the microservice was redeployed from deployment option A to deployment option B?*
- *What is the probability of the microservice operational cost being less than 100$ if the QoE score is higher than 4?*

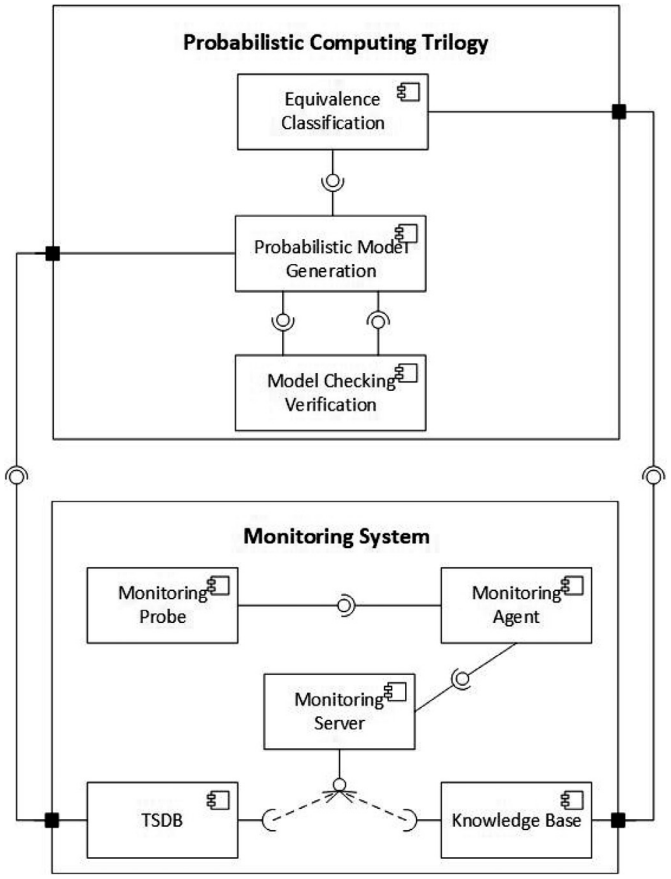

Fig. 5. System component diagram.

- *Will the probability of choosing the deployment option A instead of deployment option B be less than 40%?*

Using PCTL the above examples are presented with the following queries, respectively:

- $P_{=?}[F(higherQoS) \cup (chosenA \cup chosenB)]$;
  $P_{=?}[F(lowerQoS) \cup (chosenA \cup chosenB)]$;
- $P_{=?}[F(cost < 100\$ \,\&\, QoE > 4)]$;
- $P_{<0.4}[chosenA \cup chosenB]$.

4. System architecture

Based on the process specified in the previous section we developed an architecture and developed a system for formal QoS assurances. Our goal was to integrate the new method with an existing software engineering tool. In our architecture, we use the services of the SWITCH Workbench, which is an Interactive Development Environment for Cloud applications [52,53] that allows us to integrate this new method into the workbench.

The flow of interactions between the components of this architecture is presented in Fig. 5.

The system is composed of two main components - a Probabilistic Computing Trilogy (PCT) and a Monitoring System. PCT is a set of integrated components that execute the previously described methods for generation and verification of probabilistic models. It is designed to generate probabilistic models in a dynamic manner, using measurements and additional QoS- and NFR-related information.

The Monitoring System is composed of several sub-components. They are responsible for monitoring the system during run-time and collecting data that is necessary for the functioning of PCT. The first component, which initialises the work in PCT is the Equivalence Classification Component (ECC). This component allocates all of the available cloud deployment options and creates the initial model that is used as an input parameter for the equivalence classification. Then the equivalence classification method examines which deployment options satisfy the hard constraints and assigns them to equivalence classes. Hence, ECC can arrange a large number of available deployment options into equivalence classes.

For purposes of simplicity, the classification is based on service geolocation in the two developed microservices (use cases). Thus, two deployment options A and B will belong to the same equivalence class, if they are equivalent in terms of the same criteria, such as running within the same region. Finally, the ECC output (i.e. equivalence class) is sent to the Probabilistic Model Generation Component (PMGC).

PMGC prepares a finite probabilistic model, the size of which is equal to the size of the set that was provided by the ECC. In addition, PMGC calculates the values of the transition probabilities and the reward system. Both components, ECC and PMGC were developed using Java-based technologies and frameworks, such as Java Jersey for RESTful web services and Apache Maven for software management.

The Model Checking Verification (MCV) component is used to verify the probabilistic model generated by the PMGC. It allows for the probabilistic quantification of the described properties by implementing probabilistic computation tree logic (PCTL). In this way, MCV verifies the degree to which the cloud deployment options would satisfy the microservice NFRs under the specific constraints that were set up by the software engineer. Hence, this component provides assurances to the software engineer that the system short-lists the optimal set of deployment options for the microservice to run. This component utilized the PRISM model checker [54], which is a tool used to quantify and analyse probabilistic models.

The correctness of the results offered by the PCT component strongly relies on a Monitoring System that includes a Knowledge Base of various microservices properties. The Monitoring System is a set of monitoring components, such as probes for different QoS metrics, agents and a server, which dynamically collect metrics during the application's runtime. It has an important role in the system, because it is used to measure metrics (e.g. throughput, latency, CPU and memory utilisation), and this ensures that the necessary QoS requirements are satisfied at runtime by any required microservice adaptation. Usually the Monitoring System starts operating when a microservice container is deployed and started on a cloud deployment option. Jcatascopia [55] is used to implement the Monitoring System.

The Monitoring Agents are lightweight components that manage metrics collection from Virtual Machine and container instances. The main purpose of the Monitoring Agents is to collect the measured data, parse the data and register the Monitoring Probes. The Monitoring Probes are components created to collect low- and high-level metrics. Finally, the Monitoring Server component is used to collect the monitored data from the Monitoring Agent and forward it to a database, which in our case is a Time Series Database (TSDB).

The Time Series Database is a database that is used for storing QoS metrics. This database is optimised for rapid CRUD operations and does not require complex data queries. In our experiments we use the open-source Apache Cassandra.[1]

The Knowledge Base (KB) is developed by using the Jena Fuseki Open Source libraries. It is used to collect complex information expressed in the Resource Description Framework (RDF). This data is required by the ECC and the MGC as input data. The KB is used to collect information about the selected cloud deployment options, such as: geolocation, optimisation of application type, the calculated NFR and QoE scores. Through the use of SPARQL queries, the structured information from the KB is used when preparing the equivalence classes and populating them with the correct configurations.

[1] http://cassandra.apache.org/.

5. Experiments and results

The developed system was experimentally tested with two different microservices (use cases). These have high QoE requirements; for example, fast file uploads are needed for the first and high quality video-conferencing performance is needed for the second. Both microservices are time-critical in nature and are named File Upload and Jitsi Meet Video-conferencing. Both microservices are designed for very dynamic multi-instance event-driven usage, which allows the software components to be deployed on a different deployment option, used and terminated each time they have to be used by a client. This provides two excellent testing cases for our new method. For purposes of our experimentation, both applications are implemented as Docker containers,[2] which is an advanced technology for microservices virtualisation.

This section summarizes our experimental results related to the construction and use of two FoQoSAM models for both use cases. It also compares the results of the new MDP-based method with an AHP method as a baseline multi-criteria decision-making method.

5.1. Use cases

5.1.1. File upload

The goal of this use case is to determine an optimal IaaS for uploading a large file, where high QoS requirements (e.g. high throughput and low package loss) must be satisfied. Although file uploading operations represent a rather simple usage scenario, they are network intensive. The network throughput is therefore a very important requirement for this use case.

File Upload is a very basic software component, implemented as a service. It is developed as a Java Servlet-based Web application that processes the requests for uploading files on an HTTPS server. The implementation allows determining different NFR thresholds for each upload operation, since the container instance can be initiated on different cloud deployment options for each upload. For instance, if a software engineer that has initiated the container instance on one cloud deployment option does not receive the expected QoS due to the long upload time, the engineer can initiate the next uploading operation using another deployment option. Manual selection of deployment options is a time-consuming process and requires an advanced knowledge of cloud deployment options. By implementing the FoQoSAM method, the software engineer is automatically provided with an optimal deployment option that satisfies the QoS requirements.

For the File Upload microservice the following NFR attributes were implemented in the FoQoSAM method: service location, network latency, throughput, packet loss and QoE threshold. The service location was used as a hard constraint, whereas the other attributes were used as soft constraints.

5.1.2. WebRTC video-conferencing

The goal of the video-conferencing microservice is to determine an optimal placement deployment option for a high quality video-conference. In contrast to the File Upload, the video-conferencing microservice provides for communication between multiple parties with no interruptions or delays. Hence, different QoS are important, such as low network latency.

The microservice was developed based on Jitsi Meet open-source technology.[3] The client-side of the application is run on Web browsers that support the Real-Time Communication (RTC) protocol. This application is composed of four components: (1) Jitsi Videobridge manages the audio/video streams between participants; (2) Jicofo runs the video conferences; (3) Prosody is responsible for the exchange of signalling messages; (4) The Web server utilizes the WebRTC protocol to serve the application. For the current use case, these components were packed into a single Docker container image and represent a single microservice, although other arrangements are also possible. Also in this use case scenario, the microservice is deployed as a selected deployment option that is started and destroyed for each video-conferencing (usage) event.

We use the same NFR attributes as in the case of the File Upload microservice. However, due to the different nature of the software components, different threshold values are used as constraints. The threshold values for both use cases are shown in Section 5.3.

Table 2
Cloud deployment options used in the experiments.

Infrastructure	id	vCPU	RAM	Location
ARNES	0	1	4	Europe
g1-small	1	1	1.7	Asia
				Europe
				USA
n1-standard 1	2	1	3.75	Asia
				Europe
				USA
n1-standard 2	3	2	7	Asia
				Europe
				USA
n1-standard 4	4	4	15	Asia
				Europe
				USA
n1-standard 8	5	8	30	Asia
				Europe
				USA

5.2. Experimental testbed

The goal of the experimental evaluation is to show that the proposed method is fully functional, and that by utilizing multiple QoS constraints it can provide autonomous deployment of containerized cloud applications on optimal deployment options that satisfy the QoS requirements. The experiments were executed in Ljubljana, Slovenia, from where the two microservices were deployed. For testing purposes, let us assume that there were several deployment scenarios in which the software engineer had different workload requirements: 500 (Workload 1), 1000 (Workload 2) and 1500 (Workload 3) requests every five seconds. The workload was actually generated by using the *httperf* tool [56].

The software engineer(s) could deploy their microservices on one of 26 cloud deployment options that were hosted on Google Cloud Platform[4] and on the Academic and Research Network of Slovenia (ARNES).[5] Google Cloud Platform services are available at numerous locations in five regions around the world: North America, South America, Europe, Asia, and Australia. ARNES is the Academic and Research Network of Slovenia, which provides network services for research, educational and cultural purposes. As previously described, geolocation is one of the NFR attributes that is used for the experiments. Therefore, for purposes of experimental evaluation, our implemented microservices were run in five different locations: Ljubljana, Frankfurt, Iowa, Taiwan and Tokyo. The experimental evaluation was performed on standard VMI configurations with no additional performance optimisation. They are enumerated (id) and their properties listed in Table 2. Three people participated in the evaluation as software engineers.

In order to investigate the influence of geolocation on network performance, the microservices were deployed on all 26 deployment options. The results obtained for each microservice are summarized in Tables 3 and 4. As can be concluded from the results in the tables, network latency is heavily affected by the distance between the location of the engineers (also potentially users) and geolocation of the deploy-

[2] https://www.docker.com/
[3] https://jitsi.org/jitsi-meet/
[4] https://cloud.google.com/
[5] https://www.arnes.si/

Table 3
Average network performance at different geolocations of the video-conferencing microservice.

Location	Latency	Throughput	Packet loss
Ljubljana	9.03	0.039	0.00
Frankfurt	31.75	1.462	0.00
Iowa	331.78	1.531	0.00
Taiwan	624.98	1.821	0.00
Tokyo	570.56	1.710	0.00

Table 4
Average network performance on different geolocations of the File Upload microservice.

Location	Latency	Throughput	Packet loss
Ljubljana	6.87	0.038	0.00
Frankfurt	134.75	24.57	0.00
Iowa	216.05	25.23	0.00
Taiwan	710.77	25.89	0.00
Tokyo	655.87	26.80	0.00

ment option. Hence, the greater the geographic distance from Ljubljana, the higher the latency. Because of the significant impact of the geolocation on the QoE of both microservices, the deployment option location was selected as a hard constraint for the generation of the probabilistic model.

5.3. Decision-making results

The initial model was composed of 26 cloud deployment options. By using the equivalence classification method, the amount of deployment options was reduced to just six. Thus, the equivalence classification method significantly reduced the size of the probabilistic model. Although the computing complexity of the probabilistic model with 26 deployment options results in minimum computational overhead in practice, the method must check all the transitions between the states in the model. For instance, if there are transitions between all 26 states within the probabilistic model and they are all connected to each other, the method would have to check 676 transitions. Moreover, for a model with 260 states, the method would have to check 67,600 transitions. In some cases the number of transitions could dramatically increase to some thousands, which could have a significant impact on the computational complexity of the method. Hence, reducing the number of states in the model significantly reduces the computational time.

The preselected soft constraints that were used to generate the reward values for the probabilistic model were the following: for the File Upload microservice: latency<100 ms, throughput>4.0 Gb/s, QoE>4.0, CPU utilisation <50% and memory utilisation <70% and cost<100 $/month; for the video-conferencing microservice: latency<50 ms, throughput>0.1 Gb/s, QoE>4.0, CPU utilisation <50%, memory utilisation <70% and cost <100 $/month. The soft constraints were compared to the infrastructure-level and network-level monitoring measurements that were gathered by the NetData[6] monitoring tool; and prior usage data, which was collected during the testing phase of the cloud applications.

Figs. 6 and 7 illustrate the probabilistic models, which are composed of the cloud deployment options that satisfy the equivalence class criteria, and are based on the microservices for the Workload 1 (500 requests).

The results, presented in Tables 5 and 6, show that the new method provides a ranking mechanism for cloud deployment options according to the score calculated to achieve high QoS success. The results rely strongly on the reward system; therefore, deployment configura-

[6] https://github.com/firehol/netdata

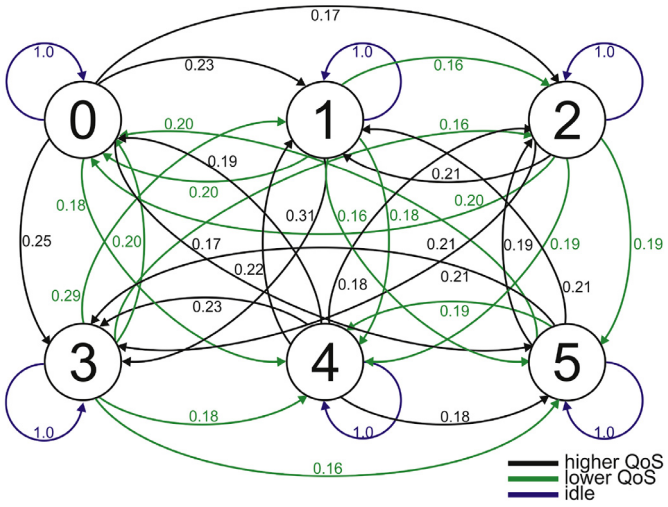

Fig. 6. Experimentally-derived probabilistic model for the File Upload microservice under Workload 1.

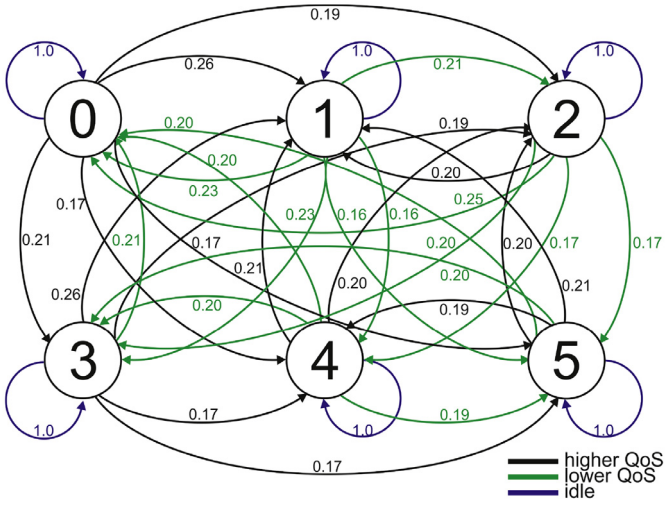

Fig. 7. Experimentally-derived probabilistic model for the video conferencing microservice under Workload 1.

Table 5
Decision-making results for the File Upload microservice.

id	Workload 500		Workload 1000		Workload 1500	
	Rank	Score	Rank	Score	Rank	Score
0	V	11.29	VI	11.77	V	11.94
1	II	11.74	II	12.84	II	13.13
2	II	11.69	IV	12.19	III	12.35
3	I	12.37	I	12.90	I	13.09
4	VI	11.27	III	11.81	VI	11.90
5	IV	11.69	V	12.06	IV	12.35

Table 6
Decision-making results for the Videoconferencing microservice.

id	Workload 500		Workload 1000		Workload 1500	
	Rank	Score	Rank	Score	Rank	Score
0	VI	13.83	VI	13.61	V	13.48
1	I	16.27	I	16.07	II	14.65
2	II	14.91	II	14.73	III	14.62
3	V	14.25	IV	13.67	I	16.05
4	III	14.87	V	14.08	VI	13.46
5	IV	14.19	III	14.69	IV	13.91

Table 7
Verification of the decision-making results for the File Upload microservice against the hard and soft constraints.

id	Workload 500		Workload 1000		Workload 1500	
	i[a]	ii[b]	i	ii	i	ii
0	+	-	+	-	+	-
1	+	-	+	+	+	+
2	+	-	+	-	+	-
3	+	+	+	+	+	+
4	+	-	+	-	+	-
5	+	-	+	-	+	-

[a] $i : Criterion1 = G(hardConstraints)$.
[b] $ii : Criterion2 = F[(soft)\&nfrViolation \leq minViolations]$.

Table 8
Verification of the decision-making results for the Videoconferencing microservice against the hard and soft constraints.

id	Workload 500		Workload 1000		Workload 1500	
	i[a]	ii[b]	i	ii	i	ii
0	+	-	+	-	+	-
1	+	+	+	+	+	-
2	+	-	-	-	+	-
3	+	-	-	-	+	+
4	+	-	-	-	+	-
5	+	-	-	-	+	-

[a] $i : Criterion1 = G(hardConstraints)$.
[b] $ii : Criterion2 = F[(soft)\&nfrViolation \leq minViolations]$.

tions with higher rewards usually have higher ranking scores. The different workloads obviously influence the QoS values of the deployment options. Consequently, the ranking results and scores generated by the FoQoSAM method differ with workloads as well. This shows that the user is offered an optimal deployment option with high QoS with respect to the monitoring measurements at the time of deployment (i.e. at the moment a decision for deployment is taken). For instance, g1-small provided the highest QoS for the File Upload microservice according to the requirements of Workload 500, while n1-standard provides the highest QoS according to Workload 1500 requirements. In addition, g1-small provides the highest QoS for the video-conferencing microservice according to Workload 1000 requirements, while n1-standard provides the highest QoS according to Workload 1500 requirements.

In order to prove the correctness of the generated results, the model was verified using the verification criteria described in Section 3.4. The goal of the verification was to check if: (1) the cloud deployment options that were taken into account by the method satisfy the hard constraints set by the software engineers; (2) the optimal cloud deployment option for deployment of the application satisfies the maximum amount of soft constraints. The results from the verification process are presented in Tables 7 and 8.

In general, the output results meet with our expectations, and show that the integrated constraints within the model influence the overall QoS score of the deployment options.

In the final stage of the experimental evaluation, we compare this new FoQoSAM method with the AHP baseline method (see Section 2). In order to generate the ranking results, AHP applied the same input parameters as FoQoSAM. For purposes of comparing them as pairs, AHP considered a set of evaluation criteria (i.e. NFRs) and a set of alternative options (i.e. cloud deployment options). As shown in Fig. 8, the hierarchy is structured in three layers. The first layer is the goal of the analysis, which is the selection of an optimal cloud deployment option. The second layer is populated by seven evaluation criteria that are used to compute the vector of criteria weights. The third layer is composed

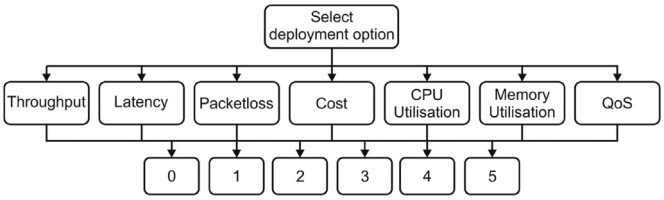

Fig. 8. AHP hierarchy of NFRs and available cloud deployment options.

of six cloud deployment alternatives that are to be ranked by AHP. In our case, the AHP was implemented in three consecutive steps.

- Computing the vector of criteria weights: At this step, a pairwise comparison matrix is composed, which is used to compute the criteria weights. In our case, the matrix had dimensions 7×7, because its size depends of the number of evaluation criteria considered. Because in the current case all of the criteria had equal importance, they all had equal importance weight. The vector of criteria weights was calculated by averaging the entries of each row from the comparison matrix.
- Computing the matrix of alternative scores: During this step, all of the alternatives are compared to each other with respect to the set of criteria. Because, each alternative is connected to each criterion, the method produces seven comparison matrices with dimensions 6×6. The method first populates a vector with the average values for the entries of each row of the comparison matrices. The resulting vectors are column vectors in the matrix of alternative scores. The resulting matrix of alternatives scores in our case had dimensions of 6×7.
- Ranking the alternatives: The ranking results are obtained as a vector, which is a derived by multiplying the vector of criteria weights and the matrix of alternative scores.

The results obtained from the AHP method are presented in Tables 9 and 10, where they are compared to the MDP results. In order to compare the results from both methods, we define the metric N as the number of QoS metrics, whose thresholds are violated. From the tables it can

Table 9
Comparison of method ranking results with respect to the number of QoS threshold violations for the File Upload microservice.

id	Workload 500			Workload 1000			Workload 1500		
	MDP[a]	AHP	N[b]	MDP	AHP	N	MDP	AHP	N
0	V	II	4	VI	IV	4	V	III	4
1	II	VI	3	II	VI	2	II	VI	2
2	III	IV	3	IV	I	3	III	V	3
3	I	V	2	I	III	2	I	I	2
4	VI	III	4	III	V	4	VI	IV	4
5	IV	I	3	V	II	3	IV	II	3

[a] *The MDP based FoQoSAM method;*
[b] *N = Number of metrics with threshold violations.*

Table 10
Comparison of method ranking results with respect to the number of QoS threshold violations for the Videoconferencing microservice.

id	Workload 500			Workload 1000			Workload 1500		
	MDP[a]	AHP	N[b]	MDP	AHP	N	MDP	AHP	N
0	VI	IV	4	VI	III	4	V	III	4
1	I	II	1	I	I	1	II	VI	2
2	II	III	2	II	II	2	III	II	2
3	V	VI	3	IV	VI	4	I	V	1
4	III	V	2	V	IV	3	VI	IV	3
5	IV	I	3	III	V	2	IV	I	4

[a] *The MDP based FoQoSAM method;*
[b] *N = Number of metrics with threshold violations.*

be seen that the two methods selected the same cloud deployment option as optimal for deployment on just two occasions, for the File Upload under Workload 1500 and for the Videoconferencing under Workload 1000.

In contrast to the AHP method, FoQoSAM always tends to select a deployment option that does not violate the QoS metrics thresholds. Therefore, the method ranks the deployment options in order of lowest number of QoS threshold violations to the highest number of QoS threshold violations. Our method generates its ranking list by executing multiple simulations from random initial states of the probabilistic model, thus exploring all possible alternatives before providing the ranking list. During our simulations, the FoQoSAM method checks whether the current state in the model delivers the optimal QoS and what action should be selected in order to reach such a state. In contrast, the AHP method derives its ranking by implementing a pairwise comparison of the deployment options attributes, without considering the user QoS requirements.

6. Discussion

In the course of our study, multiple infrastructure-level and network-level metrics were investigated to determine the most effective hard and soft constraints for equivalence classification. In the experiments, the geographical location is a hard constraint used to create deployment option equivalence classes. Tables 3 and 4 show that the location of the deployment option and the distance between the engineer and the deployment option significantly impact the network quality results. In our experiments, this specific NFR enabled us to reduce the number of deployment options for consideration by approximately 77%, thus reducing the time for computations. This of course depends on the types of deployment options and hard constraints that are used and in some situations it may not be possible to achieve such reductions. On the other hand, the equivalence classification method may be further improved. For instance, a taxonomy of application types could be used to prepare equivalence classes according to the type of microservice (e.g. database, time-critical service, logic calculations and so on), which may contribute toward more precise and optimal classification of cloud deployment options.

Since every application has different QoS requirements, the FoQoSAM method allows us to extend the NFRs according to the application's needs. The current system only allows the use of quantitative NFRs, while NFRs such as: safety, security and so on will be defined and integrated in our future work.

The output results, which are presented in Tables 5 and 6, demonstrate that the proposed probabilistic decision-making mechanism ranks the deployment options depending on the system QoS and the current measurement data. The optimal results rely on multi-level QoS metrics and data on past usage that is collected during runtime.

The different workloads have a direct impact on the network performance and on resource utilisation. Hence, the overall QoS is influenced and the user QoE ratings vary. As a result, the proposed model suggests different optimal deployment options for deploying the two different applications. A system like this can be further improved to be able to learn from information contained in the KB and the user's behaviour, so that it can provide more precise assurances to the software engineer.

7. Conclusions

QoS and NFR assurances are essential requirements of software engineers in the cloud application deployment phase. This study introduces a formal modelling method called FoQoSAM that can be used for this purpose and which is integrated in an existing software engineering tool, the SWITCH workbench. This paves the way for the use of such probabilistic methods in modern software engineering practices.

The developed methods take the necessary properties of the applications along with QoS and other NFR metrics, and information from IaaS providers, matches these properties and performs calculations in order to generate a ranked list of available deployment options. The ranked list of deployment options is accompanied by QoS assurances expressed as ranking scores.

The results of this process can therefore be very instrumental, which is fully automated. In many situations when the applications need to be deployed closer to the end-users, e.g. in Edge or Fog computing scenarios, automated decision-making is essential to address complexity issues and to avoid human errors.

This study presented two microservices with different QoS requirements, and an application of the FoQoSAM method. The results show that the new method is generic enough to model different microservices with different QoS requirements. The proposed FoQoSAM modelling approach is not limited to the NFR attributes considered in this study. It can be further expanded by implementing additional NFRs to effectively address the QoS requirements of any other microservice. Therefore, the use of our multi-level monitoring system is an essential part of obtaining data in order to achieve the fine grained results in the formal modelling process.

Following this study, we intend to enlarge the developed KB with information about the minimal QoS requirements of various application types. This can then be applied directly using our equivalence classification method. As such, the equivalence classification could organise the deployment options into classes, where the QoS requirements would be predefined according to the application type. As a result, software engineers would only need to select the application type and the decision-making mechanism would provide them with a calculated formal QoS assurance for the deployment option ranking results.

Acknowledgements

The research and development reported in this chapter have received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreements no. 643963 (SWITCH project: Software Workbench for Interactive, Time Critical and Highly self-adaptive Cloud applications), no. 644179 (ENTICE project: dEcentralized repositories for traNsparent and efficienT vIrtual maChine opErations), and no. 815141 (DECENTER: Decentralised technologies for orchestrated Cloud-to-Edge intelligence). Funding was also received from the Slovenian Research Agency under grant agreement no. BI-RU/16-18-043 (Internet of Things and cloud computing as support for the development of new smart approaches in the construction sector).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.infsof.2019.01.003.

References

[1] A. Zanella, N. Bui, A. Castellani, L. Vangelista, M. Zorzi, Internet of things for smart cities, IEEE Internet Things J. 1 (1) (2014) 22–32.
[2] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, M. Ayyash, Internet of things: a survey on enabling technologies, protocols, and applications, IEEE Commun. Surv. Tutorials 17 (4) (2015) 2347–2376.
[3] P. Kochovski, V. Stankovski, Supporting smart construction with dependable edge computing infrastructures and applications, Autom. Constr. 85 (2018) 182–192.
[4] J. Lewis, M. Fowler, Microservices: a definition of this new architectural term, Mars (2014).
[5] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: Performance Analysis of Systems and Software (ISPASS), 2015 IEEE International Symposium On, IEEE, 2015, pp. 171–172.
[6] H. Gupta, A. Vahid Dastjerdi, S.K. Ghosh, R. Buyya, Ifogsim: a toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments, Software 47 (9) (2017) 1275–1296.
[7] A.V. Dastjerdi, R. Buyya, Fog computing: helping the internet of things realize its potential, Computer 49 (8) (2016) 112–116.
[8] M. Tahir, F. Khan, M. Babar, F. Arif, F. Khan, Framework for better reusability in component based software engineering, J. Appl. Environ. Biol.Sci. (JAEBS) 6 (2016) 77–81.

[9] S. Kebir, I. Borne, D. Meslati, A genetic algorithm-based approach for automated refactoring of component-based software, Inf. Softw. Technol. 88 (2017) 17–36.
[10] L. Moonen, A.R. Yazdanshenas, Analyzing and visualizing information flow in heterogeneous component-based software systems, Inf. Softw. Technol. 77 (2016) 34–55.
[11] B. Sotomayor, R.S. Montero, I.M. Llorente, I. Foster, Virtual infrastructure management in private and hybrid clouds, IEEE Internet Comput. 13 (5) (2009).
[12] A. Tosatto, P. Ruiu, A. Attanasio, Container-based orchestration in cloud: state of the art and challenges, in: Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on, IEEE, 2015, pp. 70–75.
[13] J.L.L. Simarro, R. Moreno-Vozmediano, R.S. Montero, I.M. Llorente, Dynamic placement of virtual machines for cost optimization in multi-cloud environments, in: High Performance Computing and Simulation (HPCS), 2011 International Conference on, IEEE, 2011, pp. 1–7.
[14] S.K. Panda, P.K. Jana, Efficient task scheduling algorithms for heterogeneous multi-cloud environment, J. Supercomput. 71 (4) (2015) 1505–1533.
[15] U. Paščinski, J. Trnkoczy, V. Stankovski, M. Cigale, S. Gec, Qos-aware orchestration of network intensive software utilities within software defined data centres, J. Grid Comput. (2017) 1–28.
[16] V. Stankovski, D. Petcu, Developing a model driven approach for engineering applications based on mosaic, Cluster Comput. 17 (1) (2014) 101–110.
[17] P. Bonissone, Research issues in multi criteria decision making (mcdm): the impact of uncertainty in solution evaluation, in: Proceedings of the 12th International Conference on Processing and management of uncertainty in knowledge-based systems (IPMU), Málaga, Spain, 2008, pp. 1409–1416.
[18] D.M. Curry, C.H. Dagli, Computational complexity measures for many-objective optimization problems, Procedia Comput. Sci. 36 (2014) 185–191.
[19] P. Štefanič, D. Kimovski, G. Suciu Jr., V. Stankovski, Non-Functional Requirements Optimisation for Multi-tier Cloud Applications: An Early Warning System Case Study, IEEE, 2018.
[20] L. Rabiner, B. Juang, An introduction to hidden markov models, IEEE ASSP Mag. 3 (1) (1986) 4–16.
[21] H.M. Taylor, S. Karlin, An Introduction to Stochastic Modeling, Academic Press, 2014.
[22] J. Hu, J. Gu, G. Sun, T. Zhao, A scheduling strategy on load balancing of virtual machine resources in cloud computing environment, in: Parallel Architectures, Algorithms and Programming (PAAP), 2010 Third International Symposium on, IEEE, 2010, pp. 89–96.
[23] L.E. Li, T. Woo, Dynamic load balancing and scaling of allocated cloud resources in an enterprise network, 2011, US Patent App. 12/571,271.
[24] M. Randles, D. Lamb, A. Taleb-Bendiab, A comparative study into distributed load balancing algorithms for cloud computing, in: Advanced Information Networking and Applications Workshops (WAINA), 2010 IEEE 24th International Conference on, IEEE, 2010, pp. 551–556.
[25] Z. Chaczko, V. Mahadevan, S. Aslanzadeh, C. Mcdermid, Availability and load balancing in cloud computing, in: International Conference on Computer and Software Modeling, Singapore, 14, 2011.
[26] S.S. Manvi, G.K. Shyam, Resource management for infrastructure as a service (iaas) in cloud computing: a survey, J. Netw. Comput. Appl. 41 (2014) 424–440.
[27] B. Jennings, R. Stadler, Resource management in clouds: survey and research challenges, J. Netw. Syst. Manage. 23 (3) (2015) 567–619.
[28] N.C. Luong, P. Wang, D. Niyato, Y. Wen, Z. Han, Resource management in cloud networking using economic analysis and pricing models: a survey, IEEE Commun. Surv. Tutorials 19 (2) (2017) 954–1001.
[29] N. Jain, I. Menache, Resource management for cloud computing platforms, 2017, US Patent 9,595,054.
[30] S. Singh, I. Chana, Q-aware: quality of service based cloud resource provisioning, Comput. Electr. Eng. 47 (2015) 138–160.
[31] S. Chaisiri, B.-S. Lee, D. Niyato, Optimization of resource provisioning cost in cloud computing, IEEE Trans. Serv. Comput. 5 (2) (2012) 164–177.
[32] L. Zhang, Z. Li, C. Wu, Dynamic resource provisioning in cloud computing: arandomized auction approach, in: INFOCOM, 2014 Proceedings IEEE, IEEE, 2014, pp. 433–441.
[33] R. Mijumbi, J. Serrat, J.-L. Gorricho, S. Latre, M. Charalambides, D. Lopez, Management and orchestration challenges in network functions virtualization, IEEE Commun. Mag. 54 (1) (2016) 98–105.
[34] R. Karim, C. Ding, A. Miri, An end-to-end qos mapping approach for cloud service selection, in: Services (SERVICES), 2013 IEEE Ninth World Congress on, IEEE, 2013, pp. 341–348.
[35] S.K. Garg, S. Versteeg, R. Buyya, A framework for ranking of cloud computing services, Future Gener. Comput. Syst. 29 (4) (2013) 1012–1023.
[36] R. Gonçalves Junior, T. Rolim, A. Sampaio, N.C. Mendonça, A multi-criteria approach for assessing cloud deployment options based on non-functional requirements, in: Proceedings of the 30th Annual ACM Symposium on Applied Computing, ACM, 2015, pp. 1383–1389.
[37] A. Mardani, A. Jusoh, K. MD Nor, Z. Khalifah, N. Zakwan, A. Valipour, Multiple criteria decision-making techniques and their applications–a review of the literature from 2000 to 2014, Econ. Res.-Ekonomska Istraživanja 28 (1) (2015) 516–571.
[38] Z. Zheng, X. Wu, Y. Zhang, M.R. Lyu, J. Wang, Qos ranking prediction for cloud services, IEEE Trans. Parallel Distrib. Syst. 24 (6) (2013) 1213–1222.
[39] C. Guerrero, I. Lera, C. Juiz, Genetic algorithm for multi-objective optimization of container allocation in cloud architecture, J. Grid Comput. 16 (1) (2018) 113–135.
[40] C.C. Bennett, K. Hauser, Artificial intelligence framework for simulating clinical decision-making: a markov decision process approach, Artif. Intell. Med. 57 (1) (2013) 9–19.
[41] E.B. Iversen, J.M. Morales, H. Madsen, Optimal charging of an electric vehicle using a markov decision process, Appl. Energy 123 (2014) 1–12.
[42] A. Kolobov, Planning with markov decision processes: an ai perspective, Synth. Lect. Artif. Intell.Mach. Learn. 6 (1) (2012) 1–210.
[43] J. Yang, W. Lin, W. Dou, An adaptive service selection method for cross-cloud service composition, Concurrency Comput. 25 (18) (2013) 2435–2454.
[44] G. Su, T. Chen, Y. Feng, D.S. Rosenblum, P. Thiagarajan, An iterative decision-making scheme for markov decision processes and its application to self-adaptive systems, in: International Conference on Fundamental Approaches to Software Engineering, Springer, 2016, pp. 269–286.
[45] Y.R.S. Llerena, G. Su, D.S. Rosenblum, Probabilistic model checking of perturbed mdps with applications to cloud computing, in: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, 2017, pp. 454–464.
[46] D. Tsoumakos, I. Konstantinou, C. Boumpouka, S. Sioutas, N. Koziris, Automated, elastic resource provisioning for nosql clusters using tiramola, in: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on, IEEE, 2013, pp. 34–41.
[47] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, S. Sioutas, Dependable horizontal scaling based on probabilistic model checking, in: Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on, IEEE, 2015, pp. 31–40.
[48] M.L. Littman, T.L. Dean, L.P. Kaelbling, On the complexity of solving markov decision problems, in: Proceedings of the Eleventh conference on Uncertainty in artificial intelligence, Morgan Kaufmann Publishers Inc., 1995, pp. 394–402.
[49] R. Sandhu, S.K. Sood, Scheduling of big data applications on distributed cloud based on qos parameters, Cluster Comput. 18 (2) (2015) 817–828.
[50] L. Li, S. Li, S. Zhao, Qos-aware scheduling of services-oriented internet of things, IEEE Trans. Ind. Inf. 10 (2) (2014) 1497–1505.
[51] M.L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley & Sons, 2014.
[52] Z. Zhao, A. Taal, A. Jones, I. Taylor, V. Stankovski, I.G. Vega, F.J. Hidalgo, G. Suciu, A. Ulisses, P. Ferreira, et al., A software workbench for interactive, time critical and highly self-adaptive cloud applications (switch), in: Cluster, Cloud and Grid Computing (CCGrid), 2015 15th IEEE/ACM International Symposium on, IEEE, 2015, pp. 1181–1184.
[53] Z. Zhao, P. Martin, J. Wang, A. Taal, A. Jones, I. Taylor, V. Stankovski, I.G. Vega, G. Suciu, A. Ulisses, et al., Developing and operating time critical applications in clouds: the state of the art and the switch approach, Procedia Comput. Sci. 68 (2015) 17–28.
[54] M. Kwiatkowska, G. Norman, D. Parker, Prism 4.0: verification of probabilistic real-time systems, in: International Conference on Computer Aided Verification, Springer, 2011, pp. 585–591.
[55] D. Trihinas, G. Pallis, M.D. Dikaiakos, Jcatascopia: monitoring elastically adaptive applications in the cloud, in: Cluster, Cloud and Grid Computing (CCGrid), 2014 14th IEEE/ACM International Symposium on, IEEE, 2014, pp. 226–235.
[56] D. Mosberger, T. Jin, Httperfa tool for measuring web server performance, ACM SIGMETRICS Perform. Eval. Rev. 26 (3) (1998) 31–37.