This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

Certification-Based Cloud Adaptation

Claudio A. Ardagna, Rasool Asal, Ernesto Damiani, Theo Dimitrakos, Nabil El Ioini, Claus Pahl

Abstract—Performance and dependability levels of cloud-based computations are difficult to guarantee by-design due to segregation of visibility and control between applications, data owners, and cloud providers. Lack of predictability increases users' uncertainty about the service levels they will actually achieve. Cloud tenants compete for shared resources/services at all layers of the cloud stack, and pose heterogeneous and conflicting non-functional requirements over them. These requirements have implications for platform and infrastructure layers, which have to be configured to satisfy inter-tenants requirements. We argue that adaptation techniques can play a crucial role in providing a reliable cloud, supporting definite behavior of applications and stable quality of service. Existing adaptation techniques however are unsuitable for cloud use, since they mostly focus on single tenancy, performance requirements, and are based on unverifiable evidence, which is collected in an untrusted way. In this paper, we propose a multi-tenant, general-purpose adaptation technique for the cloud, based on evidence collected by means of a trustworthy certification process. We depart from traditional heavy and comprehensive certification processes, such as ISO/IEC 27017, and consider a flexible and lightweight certification process for the cloud. It is based on authentic evidence and provides accountable validation on the compliance of a cloud-based system. Our approach adapts the cloud at all layers to maintain stable non-functional properties in certificates over time, by continuously verifying certificate validity. We assess the performance and quality of our adaptation approach in a wide range of settings.

Index Terms—Cloud, Adaptive Systems, Certification, Non-functional properties, Trust

1 Introduction

Monolithic system design has become inconceivable for many of today's applications, due to the heavy effort required in the management of software components. Low scalability and tied dependencies are examples of limitations due to the monolithic approach [1]. To guarantee flexibility and scalability, software systems are increasingly designed and deployed in a cloud environment as composite services. One of the architectural advantages of cloud computing is that it provides a dynamic environment where multiple tenants share physical and virtual resources, while guaranteeing isolation among them [2]. This benefit, however, comes with some drawbacks: *i)* the higher the flexibility in terms of resource sharing provided to the tenants, the more the effort to manage non-functional aspects (e.g., security) and configurations; *ii)* resource planning and placement become more complex and less predictable [3]. Additionally, when dealing with shared resources, a special attention needs to be given to inter-tenant effects, because single-tenant actions may have an impact on the quality of service perceived by other tenants. Cloud services are affected by continuous context changes and (new and unexpected) cloud events, potentially creating a gap between the observed cloud service behavior and the expected one. These changes are known to introduce a level of uncertainty and limit predictability of cloud-based applications' behavior, when compared to on-premises IT systems [4].

Adaptation techniques and autonomic models have been proposed as foundations of a reliable cloud with consistent behavior [5], capable of reacting to context changes and events to preserve a stable quality of service for tenants. Current approaches [6]–[8] mainly target performance and availability properties, extending resource management techniques with scalability, elasticity, and reliability algorithms, without considering adaptation of other non-functional properties like the Confidentiality, Authenticity and Integrity (CIA) triad. This is clearly not acceptable in dynamic cloud environments, where automatic adaptation should support all major non-functional properties.

This paper presents a multi-tenant adaptation technique driven by certified non-functional properties. Our approach has the threefold advantage of: *i)* maintaining a stable quality of service based on trustworthy and verified evidence (Section 2.2), *ii)* providing a user-informed and multi-tenant adaptation by tuning cloud configurations over time (Sections 4 and 5), *iii)* taking into account interferences between conflicting requirements and different non-functional properties in certificates. Changes affecting the certified properties and triggering adaptation are precisely pinpointed and analyzed, using the VIKOR multi-criteria decision making method, to find and enforce the best local optimum adaptation that balances requirements of all tenants. A thorough evaluation of the proposed approach has been conducted on a wide range of simulated settings (Section 6).

We claim that our technique enhances the adaptation technique in [9] by providing a multi-tenant solution that adapts the system to find the best balance between requirements of all tenants, supports verification of any non-functional properties, and manages interferences between conflicting tenant requirements.

- *C.A. Ardagna and E. Damiani are with the Dipartimento di Informatica, Università degli Studi di Milano, Milano, Italy. R. Asal and E. Damiani are with Etisalat British Telecom Innovation Center, Khalifa University of Science, Technology and Research, Abu Dhabi, UAE. Nabil El Ioini and Claus Pahl are with Free University of Bozen, Bolzano, Italy. Theo Dimitrakos is with Huawei Technologies and the University of Kent (at British Telecom Research & Innovation and the University of Kent where the original work was initially conducted).*
E-mail: claudio.ardagna,@unimi.it, rasool.asal@bt.com, ernesto.damiani@kustar.ac.ae, theo.dimitrakos@huawei.com, nelioini@unibz.it, Claus.Pahl@unibz.it

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

TABLE 1
An example of non-functional properties and configuration constraints

Property $\hat{pr}$	Level		
	l_1	l_2	l_3
Isolation	Shared Middleware	Dedicated VM	Dedicated server
Confidentiality in Transit and Rest	Weak encryption (DES-CBC-SHA or DES-CBC-MD5)	Medium encryption (RC4-SHA or RC4-MD5)	Strong encryption (DHE-RSA-AES256-SHA or AES256-SHA)
Integrity	Checksum or AC System	Signature algorithm	Hardware key management for signature
Availability	50≤Uptime<95 (Redundancy<x or Cold DR)	95≤Uptime<99 (x≤Redundancy<y or Hot DR)	Uptime≥99 (Redundancy≥y or Multi-Cloud HA)

2 Basic Concepts

2.1 Non-functional properties

Non-functional properties commonly refer to quality aspects of a system such as reliability and security. A non-functional property pr is defined as a pair ($\hat{pr}$,l), where $\hat{pr}$ is the property class and l is the property level modeling the strength of the supported property. Levels take values in a discrete ordered domain (e.g., l_1, l_2, l_3) and provide a link to the general objectives to be supported by the system under consideration. Each level is then associated with a set of constraints on mechanisms mec supporting pr, including constraints on their configurations. Clearly, the same property level can be achieved by means of different mechanisms, each of which refers to a system component with a specific configuration. For instance, authenticity can be achieved by providing either password-based or biometric authentication. In the literature, mechanisms are sometimes considered as part of complex property definition [10]; in this paper, we decouple property declaration (i.e., the property to be evaluated – ($\hat{pr}$,l)) from its procedural definition (i.e., how to achieve property – mec) to better distinguish the goal and target of our adaptation technique.

We consider different classes of properties such as security (i.e., CIA), performance, anonymity, consistency. These properties can be further classified in two main classes: *context-dependent* and *context-independent* properties. *Context-dependent* properties are strongly coupled with the hosting environment and its configuration. For instance, performance properties depend on the underlying computing infrastructure (e.g., cluster of virtual machines) and its configuration (e.g., number of virtual machines). *Context-independent* properties are strongly coupled with the non-functional mechanisms supporting them. Their support requires corresponding mechanisms to be up and running correctly. For instance, confidentiality of data in transit can be achieved by implementing SSL/TLS. Table 1 presents some examples of properties that can be used as targets for adaptation techniques, giving for each property an example of configuration constraints.

2.2 Continuous Certification process

The components of an autonomic system are aware of changes to their state and environment, and appropriately react to them, with little or no involvement of the users [11]. In this paper, we use the cloud certification process in [12] to continuously check the status of the system and report the collected data at the basis of system adaptation.

The certification process is managed by a *certification authority* with the support of an *accredited lab* responsible for all evaluation activities. It receives as input the Target of Certification (*ToC*), representing the system under verification, the non-functional property pr to be certified, and the list of evaluation activities used to verify constraints on mechanisms mec supporting pr. All this information is specified in a machine-readable *certification model* [12], which is executed on a specific *ToC* to collect evidence ev and certify a property pr. In the following, we use a simplified model that represents the execution paths of the *ToC* as the concatenation of functional and non-functional mechanisms deployed at different layers of the cloud stack as follows.

Definition 1 (Certification Model). A certification model $\mathcal{M}$ is a direct acyclic graph $G^{\mathcal{M}}(V,E,\lambda)$, where a vertex $v_i \in V$ refers to a mechanism (e.g., access control, encryption, functional mechanisms), an edge $(v_i,v_j) \in E$ is annotated with a function call f_i to the mechanism represented by v_j, and λ is a labeling function that associates a set $\lambda(v_i)=\{c_1,\ldots,c_n\} \in \mathcal{C}$ of alternative cloud configuration constraints with each $v_i \in V$.

We note that each function call annotating an edge in $G^{\mathcal{M}}$ triggers a state transition and corresponding mechanism execution. We also note that each mechanism at vertex v must satisfy at least one constraint c_i in $\lambda(v)$ to behave correctly and support property pr. A single c_i is a conjunctive Boolean formula of expressions of the form op(*attr*,*value*), where $op \in \{=,\neq,<,>,\leq,\geq,\in\}$, *attr* is a configuration attribute referring to a non-functional mechanism, and *value* is a (set of) value for the given attribute. Configuration constraints are dynamic, that is, they include context/time-dependent attributes (e.g., performance evaluating response time, availability evaluating uptime) and event-dependent attributes (e.g., confidentiality based on encryption algorithms where algorithm robustness may change according to new vulnerabilities), and are associated with properties as presented in Table 1.

After executing the certification model, the process returns as output a certificate describing a set of evidence ev proving the support of pr by *ToC* [12]. Upon certificate issuing, additional evidence is continuously collected to verify the validity of the certification process and corresponding certificate in production, and in turn the consistency between the observed and expected *ToC* behavior at runtime. Verification of certificate validity is a process that takes as input the certification model and the collected evidence ev, and produces as output either *success* (1), if the evidence conforms to the certification model, or *failure* (0), otherwise, with a description of the type of inconsistency found. A failure returned by the verification process is triggered by *i)* an inconsistency between the certification model and the execution traces, or *ii)* a violation of configuration constraints. Scenario *i)* requires a system adaptation and has been already tackled in our previous work in [13]. Scenario *ii)* is the target of the adaptation approach in this paper. Constraints on each vertex are evaluated by probes that monitor the system configurations or by using methods and test cases that are defined *i)* in standards (e.g., IETF RFC 4231 for HMAC-SHA-*), *ii)* by the certification authority according to existing regulations (e.g., EU Privacy

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

Regulation GDPR), or *iii)* by the cloud provider. If the configuration constraints do not match the in-production system configurations, the corresponding vertices are added to a list of misconfigured vertices, which then become the target of adaptation activities (Section 4). As an example, let us consider an application certified for property *integrity* at level *l2* (data signature) in Table 1. The probe checks whether the storage supports signature mechanisms and all stored documents are signed. Constraints can also specify the expected signature algorithm and its configuration for a successful property evaluation.

2.3 Motivational scenario

Our motivational scenario is a cloud, where services are deployed at all layers of a public cloud stack.

Application layer (Software as a Service – SaaS). We consider three RESTful applications, each providing a specific service, owned by a different tenant as follows: *i) Health Application* that provides services ranging from a normal visit and medicine prescription, to planning a surgery operation or a radiology appointment, *ii) Finance Application* that provides services related to tax, insurance, and online payments, and *iii) Government Application* that provides services to interact with the public administration, including payroll and digital documents management.

Platform layer (Platform as a Service – PaaS). The SaaS applications build on two services provided by different providers: *i) CloudApplicationPlatform* (CAP) supporting horizontal functionalities such as auto-scaling, load balancing, server management; *ii) CloudCipher* supporting the integration of encryption mechanisms used by SaaS applications to secure data in transit and at rest.

Infrastructure layer (Infrastructure as a Service – IaaS). IaaS layer provides IT resources (i.e., CPU, RAM), storage, and network as a service to SaaS applications.

SaaS applications are the target of the adaptation solution in this paper. Each application is certified for a set of properties, possibly conflicting with properties of other applications, and shares common services (non-functional mechanisms) at platform and infrastructure layers. A change in the system, such as a new service joining the cloud infrastructure, a new regulatory constraint impacting on (a subset of) certified properties, new vulnerabilities and/or failures on common services, can cause violations of certified properties and trigger adaptation.

3 Adaptation Gaps

Traditional cloud adaptation approaches (e.g., [14]–[20]) are affected by some gaps that need to be addressed.

- **G1:** Traditional adaptation is mostly driven by performance and availability properties, and based on resource management techniques implementing scalability, elasticity, and reliability algorithms. Automatic reconfiguration of cloud components for other properties, such as security and privacy, is still immature.
- **G2:** Traditional adaptation mainly focuses on the infrastructure layer and resource management. However, we need end-to-end solutions adapting the cloud at all layers of the cloud stack.
- **G3:** Traditional adaptation is often driven by informal requirements and untrusted/unverified evidence. Validating and planning end-to-end adaptation across multiple cloud layers and for many cloud tenants have to rely on rigorous description of requirements and accountable trustworthy evidence.
- **G4:** Traditional adaptation is often provided as a black box, where events triggering adaptation activities, as well as monitoring results and adaptation activities themselves are not available to end users for inspection. New techniques need to provide an anchor of trust for adaptation automation based on accountable evidence.
- **G5:** Traditional adaptation relies on a continuous monitoring based on Service Level Agreements (SLAs) with fixed thresholds. This approach can result in adaptation techniques affected by contextual fluctuations [21] where: *i)* an observed event may require a new adaptation, while another adaptation is ongoing (long adaptation), *ii)* frequent events that overcome adaptation enactments result in continuous loops (oscillations).

We address the above gaps by proposing an adaptation approach that is driven by the certification process described in Section 2.2. Differently from existing adaptation approaches, our approach aims to maintain stable properties over time by continuously verifying certificate validity, according to evidence monitored and tested by probes at all layers of the cloud stack. Collected evidence is used to adapt component configurations to achieve the best local optimum, in terms of certified properties, which balances the requirements of all tenants and addresses dependencies among them.

4 Multi-constrained adaptation

Continuous certificate verification provides a list of misconfigurations that are used as the target of our adaptation plan. We depart from the simplifying assumption that an adaptation activity is self-contained and single tenant, and define our adaptation process as a multi-constrained problem, where a single adaptation activity may affect several services owned by one or more tenants and certified for different conflicting properties.

4.1 System model

We consider a scenario where a cloud provider cp aims to provide an autonomic cloud with stable quality of service, maximizing the satisfaction of its tenants. Provider cp hosts a set $\{s_1,\ldots,s_m\}$ of services owned by different tenants, which insist on horizontal services provided at all layers of the cloud stack. For simplicity but with no lack of generality, we assume that there is exactly one service for each tenant, and use s_i to refer to both services and tenants. Each service s_i comes with one or more certificates, one for each certified property pr_{ij}. Each service s_i also defines preferences on each property pr_{ij} reflecting its importance. First, it marks as *hard* those properties that must be always supported; then, it marks as *soft* those properties that can be weakened. Hard properties are denoted as pr^*_{ij} and are satisfied *iff* the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

same or a stronger property is guaranteed for the service. Soft properties are denoted as pr_{ij} and might be partially affected by the adaptation process possibly resulting in the support of weaker properties.

The requirements of each tenant s_i are then defined as the set R_i=$\{pr_{i1},\ldots,pr_{ik},pr^*_{ik+1},\ldots,pr^*_{in}\}$ of certified properties, where n is the number of existing properties. We note that a service might not be certified for a given property $pr_{ij}\in R_i$; in this case, pr_{ij}=`null`. Additionally, not all soft properties $pr_{ij}\in R_i$ can be assumed to be equally important to s_i. To rate their importance, each tenant specifies a vector of fuzzy weights $W_i[1,\ldots,n]$ (see Section 5), where $W_i[j]$ models the importance (e.g., low, medium, high) of a single property for s_i. Fuzzy weights allow comparison between tenant's preferences. We can then define R=$\bigcup_{i=1}^{m} R_i$, where R_i is the set of requirements of s_i, as the set of all tenants' requirements driving cp in its adaptation plan.

Once certified properties start fluctuating and misconfigurations are observed (see Section 2.2), the cloud provider needs to find an alternative adaptation that fits tenants' requirements. The approach in this paper targets traditional cloud environments, as defined by the NIST in [22], which are multi-tenant, shared, and elastic. These environments introduce some constraints on the design of our approach, which guided its implementation.

C1: In a multi-tenant and shared environment, it is not always possible to restore the original, certified configuration or provide an alternative configuration that satisfies all tenants in the same way, that is, guarantee the same set R of properties supported before the misconfigurations were observed. This is due to the fact that properties have hidden dependencies, in terms of their configurations.

C2: An adaptation mechanism that solves misconfigurations by changing the multi-tenancy model at runtime could be not affordable on a large scale. The multi-tenancy model can be changed by the cloud provider either at application layer (i.e., by creating two instances of a given service) or at hardware layer (i.e., by migrating a service to a different hardware/infrastructure, possibly in isolation). In the first case, the multi-tenant and shared nature of the cloud could make such model changes ineffective, due to the co-existence of different tenants with different conflicting requirements on the same set of resources. In the second case, service migration, which is computational and time intensive, is a viable solution for elastic resource scaling (property performance) only if some prediction models are implemented [23]. The migration approach is even more costly and less viable in our case due to the fact that we consider conflicting non-functional properties. As a last remark, service migration can easily bring to a scenario where tenants are physically isolated to avoid interferences, which clearly violates the foundation of traditional public clouds.

C3: Cloud providers aim to maximize their profits, while cloud tenants to maximize their benefits. We note that a blind adaptation technique that simply maximizes tenant satisfaction in a global way does not provide the best approach for both cloud providers and users. In fact, a solution targeting a global optimum would result in scenarios where some tenants have all properties satisfied, while other tenants all properties violated, with a substantial impact on cloud providers' reputation and in turn on their profits.

Given the above model and constraints, we propose a cooperative and conciliatory adaptation solution achieving consensus between the different tenants in the satisfaction of their requirements, while taking into account conflicting properties. The goal for the adaptation plan selection in Section 4.2 is to find the adaptation A^*=$\{pr^a_1,\ldots,pr^a_n\}\in A$, where pr^a_j is a property and A is a set of possible cloud adaptations that best addresses requirements of all tenants in R, or in other words the best local optimum that satisfies all properties marked as hard (i.e., $pr^*_j\in R$), and minimizes the distance between properties marked as soft supported before adaptation (i.e., $pr_j\in R$) and corresponding properties supported after adaptation (i.e., $pr^a_j\in A$). We note that, in cases where tenants mark all their properties as hard, the adaptation might fail since no consensus can be reached between tenants, unless constraint C2 is relaxed.

4.2 Adaptation plan selection

Our cloud adaptation approach is modeled as a multi-criteria decision making (MCDM) problem, to accomplish the multi-tenant nature of the cloud. MCDM provides informed decisions (e.g., selection, evaluation) over a set of available alternatives, which are characterized by multiple and often conflicting criteria [24]. Adaptation plan selection is triggered upon identifying a (list of) misconfiguration and follows two main goals *i)* the adaptation plan must satisfy the hard properties for all tenants (valid adaptation plan), *ii)* the adaptation plan should provide the best local optimum that balances support for tenants' soft properties.

All valid adaptation plans A=$\{pr^a_1,\ldots,pr^a_n\}$ are first generated according to the following definition.

Definition 2 (Valid adaptation plan** A**). Given R=$\bigcup_{i=1}^{m} R_i$ as the set of all tenants' requirements, A=$\{pr^a_1,\ldots,pr^a_n\}$ is a valid adaptation plan *iff* $\forall R_i\in R$, $\forall pr^*_{ij}\in R_i$, pr^a_j is such that *i)* $pr^a_j.\hat{pr}$=$pr^*_{ij}.\hat{pr}$ and *ii)* $pr^a_j.l\geq pr^*_{ij}.l$.

A valid adaptation plan is a set $\{pr^a_1,\ldots,pr^a_n\}$ of properties such that all hard properties in R are satisfied.

Upon identifying the valid adaptation plans, we use a MCDM method named VIKOR [25], which was introduced by Opricovic in 1998, to model the multi-criteria optimization of complex systems. VIKOR is a fuzzy-based approach that fits the need of integrating heterogeneous properties with different domains in a single solution. It receives as input all valid adaptation plans A, previously rated on the basis of tenants' s_i requirements $R_i\in R$ and corresponding weights $W_i[j]$ (see *Step 0* below), and returns as output a ranking of adaptation plans identifying the best local one A^*. The best local adaptation provided by VIKOR represents a compromise solution among conflicting criteria [25]. VIKOR is a 5-step process, preceded by a preparation step, that works as follows.

Step 0 (Preparation). It rates the suitability of each property $pr^a_{kj}\in A_k$ in fulfilling tenants' requirements $R_i\in R$, weighted according to $W_i[j]$. To this aim, for each $R_i\in R$, it measures

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

the distance d between each property $pr_j \in R_i$ and pr^a_{kj} according to the following definition.

Definition 3 (Distance function $d(pr_1, pr_2)$). Distance function $d(pr_1, pr_2)$ takes as input two properties pr_1 and pr_2, such that $pr_1.\hat{pr}=pr_2.\hat{pr}$, and returns as output the difference between the corresponding levels $pr_1.l$ and $pr_2.l$ (i.e., $pr_1.l-pr_2.l$).

In the following, when clear from the context, we will denote pr^a_{kj} both the j-th property of A_k and the corresponding property rating to be given as input to VIKOR. All preparation steps are discussed in detail in Section 5.

Step 1. It takes as input the properties pr^a_{kj} rated at Step 0, and determines the positive-ideal (pr^{a+}_j) and the negative-ideal (pr^{a-}_j) value for all of them as follows:

$$pr^{a+}_j = \begin{cases} \max\limits_{k=1,\dots,x} pr^a_{kj}, & j \in cat(b) \\ \min\limits_{k=1,\dots,x} pr^a_{kj}, & j \in cat(c) \end{cases} \quad (1)$$

$$pr^{a-}_j = \begin{cases} \min\limits_{k=1,\dots,x} pr^a_{kj}, & j \in cat(b) \\ \max\limits_{k=1,\dots,x} pr^a_{kj}, & j \in cat(c) \end{cases} \quad (2)$$

where, j=1,. . . ,n, x is the number of alternative adaptations A_k, $cat(b)$ is a criterion of type benefit, which means that the goal is to maximize it, and $cat(c)$ is a criterion of type cost, which means that the goal is to minimize it. The positive-ideal value represents the optimal solution, such as, the one that satisfies all properties. The negative-ideal value represents the worst solution, such as, the one that violates the maximum number of properties.

Step 2. It calculates the group maximum utility U_k and the minimum individual regret Z_k of a given adaptation plan A_k, using the following LP-metric aggregation function:

$$L_k(P) = \left[\sum_{j=1}^{n} \left(W[j] \times \frac{pr^{a+}_j - pr^a_{kj}}{pr^{a+}_j - pr^{a-}_j} \right)^P \right]^{1/P} \quad (3)$$

where $1 \leq P \leq \infty$, pr^a_{kj} is the *j*-th property, with j=1,. . . ,n, of the *k*-th adaptation plan A_k, and $W[j]$ is the weighted average fuzzy weight for property pr_j calculated as the average of adaptation plans' fuzzy weight $W_k[j]$. L_k(P) is the distance between A_k and the positive-ideal solution. Based on L_k(P), we calculate the group maximum utility U_k (equation 4) and the minimum individual regret Z_k (equation 5) of the *k*-th adaptation plan A_k as follows:

$$L_k(1) = U_k = \sum_{j=1}^{n} W[j] \times \frac{pr^{a+}_j - pr^a_{kj}}{pr^{a+}_j - pr^{a-}_j} \quad (4)$$

$$L_k(\infty) = Z_k = \max_{j=1,\dots,n} \left[W[j] \times \frac{pr^{a+}_j - pr^a_{kj}}{pr^{a+}_j - pr^{a-}_j} \right] \quad (5)$$

Step 3. It calculates the sorting index Q_k, with k=1,. . . ,x, using Equations 4 and 5:

$$Q_k = v \times \frac{(U_k - U^+)}{(U^- - U^+)} + (1-v) \times \frac{(Z_k - Z^+)}{(Z^- - Z^+)} \quad (6)$$

where, $U^+ = \min\limits_{k=1,\dots,x} U_k$, $U^- = \max\limits_{k=1,\dots,x} U_k$, $Z^+ = \min\limits_{k=1,\dots,x} Z_k$, $Z^- = \max\limits_{k=1,\dots,x} Z_k$. The parameter v is a decision threshold and takes values in [0,1]. A value v>0.5 represents a *voting by majority*, value v=0.5 represents *consensus*, and value v<0.5 represents a *veto*.

Step 4. It ranks the alternative plans by sorting them in relation to values U, Z, and Q in ascending order.

Step 5. It selects the best-ranked adaptation plan ($A^*=A_1$) by the measure Q, if the following two conditions are met:

- **Condition 1** aims to find whether $Q(A_1)$ provides an acceptable advantage, meaning that, the difference between $Q(A_2)$ and $Q(A_1) \geq 1/(k-1)$, where $Q(A_2)$ is the second alternative in the ranking list and k is the number of alternatives.
- **Condition 2** aims to find whether $Q(A_1)$ has an acceptable stability in decision making, meaning that $Q(A_1)$ is ranked best also by U and Z.

If *Condition 1* is not satisfied, all alternative plans $A_1, A_2, \dots, A_m$ are the same compromise solution, and therefore, there is no advantage in choosing A_1. If *Condition 2* is not satisfied, the stability in decision making is not efficient; therefore, A_1 and A_2 have the same compromise solution.

VIKOR finally returns a ranked list $A_1, A_2, \dots, A_x$, which is the starting point for adaptation plan enforcement.

4.3 Adaptation plan enforcement

The selected adaptation plan A^* is enforced by adapting configurations of all services s_i (Section 4.3.2), according to the adaptation strategies in Section 4.3.1.

4.3.1 Adaptation strategies

Adaptation enforcement identifies alternative configurations to successfully adapt a misconfigured system according to the adaptation plan selected in Section 4.2. To this aim, feature model, a formalism heavily used in software product lines, has been adopted in this paper to model alternative configurations [26]. Feature models can be applied in any domain to represent commonalities and differences in product/service configurations. They provide an overview of the configuration domain of a product and enable automated reasoning about features of interest. A feature model is formally defined as follows [27].

Definition 4 (Feature Model $\mathcal{FM}$). A feature model $\mathcal{FM}$ is a 6-tuple $\langle G, E_m, Gr_{xor}, Gr_{or}, C_{req}, C_{ex} \rangle$, where:

- $G(F, E, r)$ is a rooted tree with F as a finite set of features, $E \subseteq F \times F$ is a finite set of edges, $r \in F$ is the root feature;
- $E_m \subseteq E$ identifies the set of mandatory features;
- $Gr_{xor}, Gr_{or} \subseteq PF \times F$ represent alternative and optional feature groups, respectively, where PF is the set of parent features. Gr_{xor} and Gr_{or} are therefore sets of pairs of parent and child features;
- C_{req} and C_{ex} define finite sets of constraints specifying required and excluded features, respectively.

Our feature model specifies configurations in terms of properties pr and mechanisms mec supporting properties. A configuration can be formally defined as follows.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

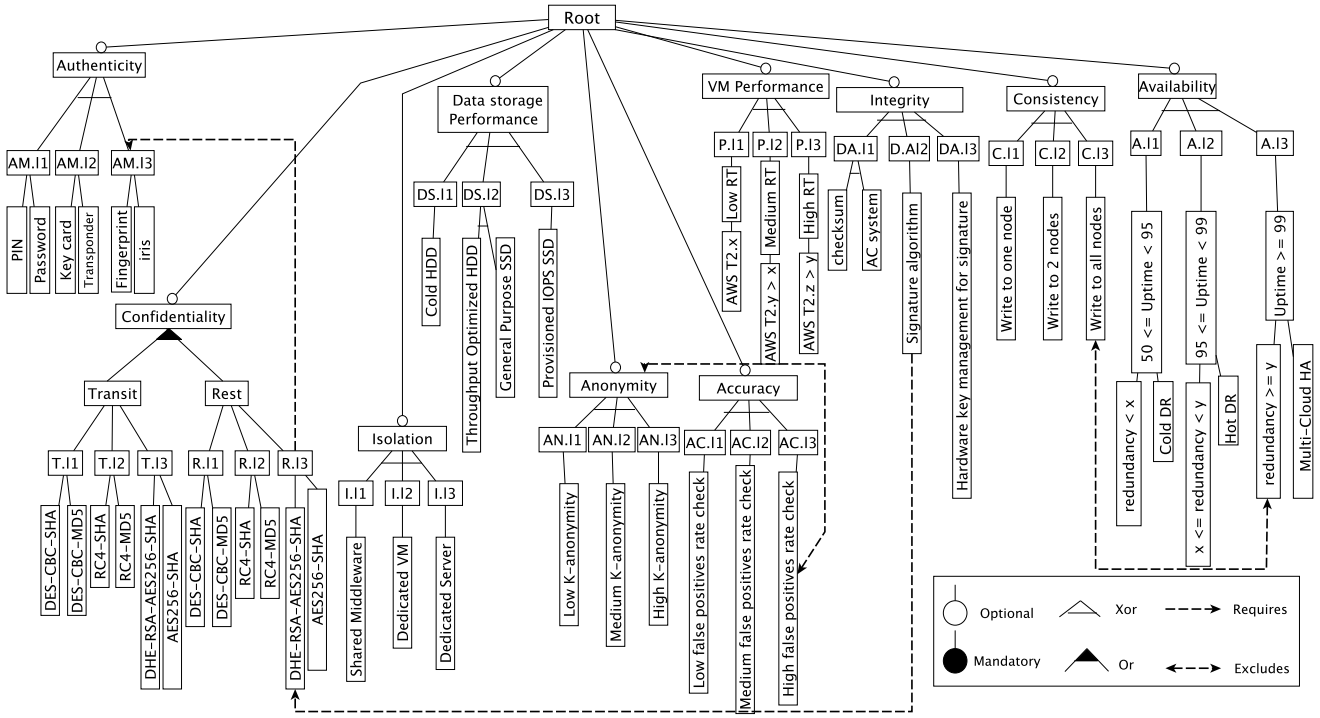


Fig. 1. An example of a feature model

Definition 5 (Configuration). Given a feature model $\mathcal{FM}$, a configuration Cf is a product in $\mathcal{FM}$ modeled as a set of features of the form $\{F_1,\ldots,F_n\}$, where $F_1{=}r$ is the root feature, F_n is a leaf feature, and $\forall_{i=1}^{n-1} F_i$, $(F_i,F_{i+1}){\in}E$.

In other words, each configuration specifies a path from root to leaf in $\mathcal{FM}$ including *i)* a property class $\hat{pr}$, *ii)* a property level l, and *iii)* a mechanism mec. In the following, we will denote a configuration Cf as [pr,mec], where $pr{=}(\hat{pr},l)$. Figure 1 presents an example of a feature model. For simplicity but with no lack of generality, each property has always three levels, whose meanings depend on the property itself. For instance, property (*VM Performance*, *P.l3*) means higher performance compared to *P.l1* and *P.l2*; property (*Confidentiality in transit*, *T.l3*) means that the mechanism used for preserving data confidentiality is stronger than the one used at *T.l1* and *T.l2*. Also, for conciseness, some configurations in the feature model are presented in a compressed form. For instance, confidentiality mechanism *AES256-SHA* both contains a constraint on the encryption algorithm (i.e., AES-SHA) and the key length (i.e., 256bit). We note that different feature models with different configurations and relations can be defined on the basis of the considered domain, meaning that the values in the feature model do not affect the soundness of the proposed approach.

Our adaptation enforcement process queries the feature model to find possible configurations that satisfy the properties in adaptation plan A^*. The query is executed using the feature model *analysis operations*.

- ***Valid_product:*** it takes as input a configuration Cf and returns as output a Boolean value stating if the input configuration exists in the feature model.
- ***Filter:*** it takes as input a pair (F_s,F_r), where F_s models the set of features to be selected and F_r the set of features to be disregarded, such that $F_s{\cap}F_r{=}\emptyset$, and returns as output the set Cf_i of configurations satisfying the input criteria.

The query does not only consider the alternatives for single configurations; it also considers cross-tree constraints in $\mathcal{FM}$, which represent both *include* and *exclude* dependencies between different configurations of the same tenant. For instance, it is possible to state that anonymity configurations are not compatible with accuracy at level *AC.l3*. This means that configurations including both will be considered as invalid configurations.

Operations *valid_product* and *filter* are used to implement the three adaptation strategies at the basis of our approach.

Restore Configuration. Strategy restore configuration aims to restore a configuration back to the state that reflects the original certified configuration. This strategy has the advantage of restoring only the needed configuration (the misconfigured one in the certification model), rather than instantiating the whole infrastructure and certification process from scratch. The restore configuration strategy uses the operation *Valid_product* to check whether the certified configuration exists in the feature model.

Example 1. Let us consider Health Application certified for property *Confidentiality in transit* at level *T.l3* (encryption of data in transit). Now assume that a system malfunctioning is forbidding the selected encryption mechanism (e.g., *AES256-SHA*) from performing the required encryption activities. This misconfiguration triggers an adaptation process that uses operation *valid_product* to check whether a certified configuration {*Confidentiality*,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

Transit, *T.l3*, *AES256-SHA*} still exists in the feature model and can be restored.

Equivalent Configuration. Strategy equivalent configuration suggests an alternative cloud configuration that is equivalent to the certified one. Formally, a configuration equivalence relation is defined as follows.

Definition 6 (Configuration equivalence relation $\equiv_{cf}$). Configuration $Cf_1=[pr_1,mec_1]\in\mathcal{FM}$ is equivalent to configuration $Cf_2=[pr_2,mec_2]\in\mathcal{FM}$, denoted $Cf_1\equiv_{cf}Cf_2$, *iff* $pr_1=pr_2$ and $mec_1\neq mec_2$.

Each level in the feature model can contain a set of alternative mechanisms that satisfy the same property pr for this specific level. Operation *Filter* is used to filter the set of configurations that contain the alternative mechanisms.

Example 2. Following on Example 1, let us assume that operation *valid_product* returns *false*, that is, encryption algorithm AES256-SHA is deprecated and not part of the feature model. At this point, strategy equivalent configuration is executed and an alternative configuration is searched. Operation filter receives as input a pair (F_s,F_r)=({*Confidentiality*, *Transit*, *T.l3*},{*T.l1*, *T.l2*}) and returns as output the following alternative configuration: {*Confidentiality*, *Transit*, *T.l3*, *DHE-RSA-AES256-SHA*}.

Restrictive Configuration. Strategy restrictive configuration suggests a configuration more restrictive (or stronger) than the certified one. Formally, a configuration restriction relation is defined as follows.

Definition 7 (Configuration restriction relation $>_{cf}$). Configuration $Cf_1=[pr_1,mec_1]\in\mathcal{FM}$ is more restrictive than configuration $Cf_2=[pr_2,mec_2]\in\mathcal{FM}$, denoted $Cf_1>_{cf}Cf_2$, *iff* $pr_1.\hat{pr}=pr_2.\hat{pr}$ and $pr_1.l>pr_2.l$, that is, $pr_1.l$ is more restrictive or stronger than $pr_2.l$.

Operation *Filter* is used to find those configurations that are more restrictive than the certified one.

Example 3. Following on Example 2, let us assume that Health Application is also certified for property *VM Performance* at level *P.l2* using *AWS T2.y* $> x$ and all configurations (encryption algorithms) returned by strategy equivalent configuration have a negative impact on system performance. The intensive computation required by the selected encryption algorithm causes the utilization of 100% of the CPU capacity triggering a new adaptation on property performance. In this case, Operation *Filter* is used to search a configuration that is more restrictive (high performance) than the certified configuration. The pair (F_s,F_r)=({*VM Performance*},{*P.l1*, *P.l2*, *Authenticity*, *Confidentiality*, *Isolation*, *Data Storage Performance*, *Integrity*, *Consistency*, *Availability*}) is given as input to operation *Filter*, which returns as outputs configuration {*VM Performance*, *P.l3*, *AWS T2.z* $> y$}.

4.3.2 Adaptation plan enforcement process

The adaptation plan enforcement process identifies alternative configurations to successfully adapt a misconfigured system according to the adaptation plan A^* selected in Section 4.2. It receives as input *i)* $A^*=\{pr_1^a,\ldots,pr_n^a\}$ and *ii)* for each s_i, the set of existing configurations $CF_i=\{Cf_{i1},\ldots,Cf_{in}\}$, where $Cf_{ij}\in CF_i$ supports property $pr_{ij}\in R_i$. For each s_i, it then produces as output the new set CF_i^* of adapted configurations. The enforcement process proceeds as follows.

Preparation. For each property $pr_j^a\in A^*$, a set CF_j^e of equivalent configurations supporting pr_j^a are pre-calculated using the feature model and operation *Filter* (see Definition 6).

Plan enforcement. For each service s_i, for each configuration $Cf_{ij}\in CF_i$, one of the following three enforcement approaches is executed.

- *No operation or restore configuration.* It applies when $\big[(\exists Cf_e\in CF_j^e\colon\ Cf_e=Cf_{ij}\ \vee\ Cf_e>_{cf}Cf_{ij}\ )\ \wedge$ *valid_product*(Cf_{ij})=*true*$\big]$, meaning that either Cf_{ij} or a configuration more restrictive than Cf_{ij} belongs to CF_j^e, and Cf_{ij} exists in the feature model. In this case, no operation is done or at least a restore activity is executed in case configuration Cf_{ij} is among the ones misbehaving. We note that the choice of maintaining the original configuration when a more restrictive one is suggested aims to minimize the number of adaptations done at each enforcement. This guarantees that each tenant's certified configurations will be kept stable as much as possible.
- *Downgrade configuration.* It applies when $\exists Cf_e\in CF_j^e$: $Cf_{ij}>_{cf}Cf_e$, meaning that it exists at least a configuration Cf_e in CF_j^e that is less restrictive than Cf_{ij}. A configuration adaptation is executed decreasing configuration Cf_j to configuration Cf_e. If multiple Cf_e exists, a random one among the most restrictive is picked. We note that different approaches can be used in place of a random selection, for instance considering the cost of adaptation, the type of property, and the number of interferences involving the property. This topic is outside of the scope of this paper.
- *Equivalent configuration.* It applies when *valid_product*(Cf_{ij})=*false*, meaning that Cf_{ij} does not exist in the feature model. In this case, a random configuration in CF_j^e is chosen.

We note that the practical applicability of our approach does not only depend on the ability to find an alternative configuration, but also on the overhead required to deploy such a configuration, while keeping the system consistent. For instance, what happens if a given encryption algorithm is deprecated? Should the provider decrypt and encrypt all data? Our approach to configuration enforcement aims to reduce such overhead by integrating as much as possible with cloud functionalities and normal cloud provider activities. This is straightforward for context-dependent properties, which are strongly coupled with the hosting environment and can use it for configuration enforcement. For instance, an adaptation of property performance that requires to move a service from a virtual machine (AWS T2.X) to a bigger one (AWS T2.Y) can be enforced by using either *i)* cloud elasticity features treating the adaptation enforcement as any other scaling operation or *ii)* cloud migration services. Configuration enforcement is instead more complex for context-independent properties, each having specific peculiarities that need to be managed. In this case, configuration enforcement must resemble to normal cloud activities thus reducing the adaptation overhead. For instance, when an encryption algorithm for confidentiality

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

of data at rest becomes deprecated, new storing activities will use the new selected algorithm, while ongoing storing activities will be first stopped and then restarted using the new algorithm. The treatment of the data already encrypted with the deprecated algorithm depends on the provider guidelines: data can be all decrypted and re-encrypted with the new algorithm (when a vulnerability is found in the original algorithm), or can be re-encrypted with the new algorithm when the user requires a decryption-encryption activity (when the algorithm is simply removed from the set of offered algorithms).

Plan verification. Each generated CF_i^* is checked for consistency against cross-dependencies in the feature model. In particular, the whole set of configurations is given as input to operation *valid_product*. If, for each CF_i^*, *valid_product*(CF_i^*)=*true*, the enforcement is successful; otherwise, the reasons causing a failure in operation *valid_product* (i.e., *valid_product*(CF_i^*)=*false*) are pinpointed, and the corresponding configurations further adapted, until either a solution is found or all possible configurations are tried. In the latter case, the adaptation process fails and the next adaptation plan ranked by VIKOR is enforced.

Plan monitoring. Upon all services s_i have been adapted, a predefined time window is specified to check misconfigurations caused by the enforced adaptation and due to hidden dependencies between properties. This check is carried out using the monitoring capabilities of our certification process (Section 2.2). In case a misconfiguration is raised within the predefined time window, the next adaptation plan ranked by VIKOR is enforced. Otherwise, the adaptation plan process is executed from scratch. This approach guarantees that, if a plan addressing all hidden dependencies exists, it is found by our approach; recurrent hidden dependencies can then be added to the feature model to reduce adaptation costs. Additionally, there are cases where none of the adaptation plans ranked by VIKOR is a valid solution due to hidden dependencies; in such cases, the cloud provider can implement an approach based on *spillover*, where part of the services are moved to cloud infrastructures owned by different providers.

Once the adaptation is found to be consistent, the issued certificates are updated to reflect the new adapted mechanisms and the continuous certificate verification restarts. Certificate update is based on our approach to incremental certification presented in [28]. We note that, in case new and multiple adaptations are needed, we always consider the original certificates, not the adapted ones, as the starting point for our VIKOR-based approach. This choice is due to the fact that, otherwise, incremental adaptations could keep diverging from the original configurations.

5 Walkthrough Example

We propose a walkthrough example of our approach, considering Health Application (s_1), Finance Application (s_2), and Government Application (s_3) in our motivational scenario. We assume the three tenants to be certified for property classes *i)* $p\hat{r}_1$: Confidentiality in transit (C), *ii)* $p\hat{r}_2$: Availability (A), *iii)* $p\hat{r}_3$: Data storage performance (DS). Each tenant s_i then comes with three certified properties pr_{i1}, pr_{i2}, and pr_{i3}, which become tenants' requirements as shown in Table 2. For instance tenant s_1 has been certified for property confidentiality in transit with level 3 (pr_{11}). Requirements are mapped onto the feature model in Figure 1.

TABLE 2
Tenants' requirements for property classes Confidentiality in transit (C), Availability (A), Data storage performance (DS). Hard properties are denoted in bold characters.

	pr_1	pr_2	pr_3
s_1	(C, T.l3)	**(A, A.l2)**	(DS, DS.l3)
s_2	**(C, T.l2)**	(A, A.l2)	(DS, DS.l1)
s_3	(C, T.l2)	(A, A.l3)	(DS, DS.l1)

5.1 Adaptation plan selection

We first present an example of the adaptation plan selection process based on VIKOR in Section 4.2.

Preparation activities. Each valid adaptation plan A_k is rated according to the tenants' requirements $R_i \in R$. To this aim, tenants use three linguistic variables (Table 3(a)) to assign importance weights to each of their service properties (Table 3(b)). For instance, Health Application (s_1) specifies weight *Medium* (M) for property pr_{12}=(*Availability*, *A.l3*), which is defined by a triangular fuzzy number (0.25, 0.5, 0.75), as presented in Table 3(a).

TABLE 3
Linguistic variables for weights (a) and weights on services' properties (b)

Low (L)	(0, 0, 0.25)
Medium (M)	(0.25, 0.5, 0.75)
High (H)	(0.75, 1, 1)

(a)

	pr_1	pr_2	pr_3
s_1	L	M	H
s_2	L	H	M
s_3	M	M	H

(b)

TABLE 4
Adaptation plans

	pr_1	pr_2	pr_3
A_1	(C, C.l2)	(A, A.l2)	(DS, DS.l3)
A_2	(C, C.l2)	(A, A.l3)	(DS, DS.l1)
A_3	(C, C.l3)	(A, A.l2)	(DS, DS.l1)
A_4	(C, C.l2)	(A, A.l3)	(DS, DS.l3)

TABLE 5
Linguistic variables for ratings

Poor (P)	(0, 0, 0.25)
Fair (F)	(0.25, 0.5, 0.75)
Good (G)	(0.75, 1, 1)

Once the set of possible valid adaptations that satisfy all the hard constraints are generated according to Definition 2 and requirements in Table 2, they are rated using the distance function in Definition 3. For simplicity, we only consider four alternative adaptations (Table 4). In particular, the distance between a certified property pr_{ij} in Table 2 and the corresponding property $pr_{kj}^a \in A_k$ in Table 4 is calculated and then mapped onto the three linguistic variables in Table 5 as follows:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

$$map(pr^a_{kj} pr_{ij}) = \begin{cases} G, & d(pr^a_{kj}, pr_{ij}) \geq 0 \\ F, & d(pr^a_{kj}, pr_{ij}) = -1 \\ P, & d(pr^a_{kj}, pr_{ij}) = -2 \end{cases} \quad (7)$$

where pr_{ij} is the j-th certified property of service s_i and pr^a_{kj} is the j-th property in the adaptation plan A_k. For instance, if a service is certified for pr_{ij}=(*Confidentiality in transit*, T.l1) and adaptation plan A_k proposes pr^a_{kj}=(*Confidentiality in transit*,T.l2), distance $d(pr^a_{kj}, pr_{ij})$=T.l2−T.l1=1 is mapped onto linguistic variable Good (G). We note that the mapping in Equation 7 depends on the specific scenario and cannot be fixed a priori. Table 6 summarizes the evaluation of valid adaptation plans according to the tenants' requirements in Table 2. Each cell in the table is calculated as the distance between requirements and corresponding properties in the adaptation plan, and mapped onto linguistic variables G, F, P.

TABLE 6
Evaluation of valid adaptation plans

		pr_1	pr_2	pr_3
s_1	A_1	F	G	G
	A_2	F	G	P
	A_3	G	G	P
	A_4	F	G	G
s_2	A_1	G	G	G
	A_2	G	G	G
	A_3	G	G	G
	A_4	G	G	G
s_3	A_1	G	G	G
	A_2	G	G	G
	A_3	G	F	G
	A_4	G	F	G

Fuzzy weights (Table 7) are then generated by merging fuzzy representation of linguistic variables in Table 3(a) within linguistic weights in Table 3(b). Similarly, fuzzy ratings (Table 8) are generated by merging fuzzy representation of linguistic variables in Table 5 within linguistic ratings in Table 6. Tables 7 and 8 are the starting point to build a fuzzy decision matrix (Table 9) as follows.

TABLE 7
Fuzzy weights $\hat{w}_{ij}$ for weights in Table 3(b).

	pr_1	pr_2	pr_3
s_1	(0, 0, 0.25)	(0.25, 0.5, 0.75)	(0.75,1.00,1.00)
s_2	(0, 0, 0.25)	(0.75,1.00,1.00)	(0.25, 0.5, 0.75)
s_3	(0.25, 0.5, 0.75)	(0.25, 0.5, 0.75)	(0.75,1.00,1.00)

The matrix takes the tenants' requirements as input to find the aggregated fuzzy weight for each property, and the aggregated fuzzy rating for each adaptation plan. Let m be the number of services, the aggregated fuzzy weight $\hat{w}_j$ for each property pr_j can be calculated as [29]:

$$\hat{w}_j = \frac{1}{m}[\hat{w}_{1j} \oplus \hat{w}_{2j} \oplus \cdots \oplus \hat{w}_{mj}] \quad (8)$$

and the aggregated fuzzy rating $\hat{x}_{kj}$ for each adaptation A_k and property pr_j can be calculated as:

$$\hat{x}_{kj} = \frac{1}{m}[\hat{x}_{k1j} \oplus \hat{x}_{k2j} \oplus \cdots \oplus \hat{x}_{kmj}] \quad (9)$$

TABLE 8
Adaptation plan ratings $\hat{x}_{kij}$ for evaluation in Table 6.

	Adaptation plan A_1		
	pr_1	pr_2	pr_3
s_1	(0.25, 0.5, 0.75)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
s_2	(0.75,1.00,1.00)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
s_3	(0.75,1.00,1.00)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
	Adaptation plan A_2		
s_1	(0.75,1.00,1.00)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
s_2	(0.75,1.00,1.00)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
s_3	(0.75,1.00,1.00)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
	Adaptation plan A_3		
s_1	(0.75,1.00,1.00)	(0.75,1.00,1.00)	(0, 0, 0.25)
s_2	(0.75,1.00,1.00)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
s_3	(0.75,1.00,1.00)	(0.25, 0.5, 0.75)	(0.75,1.00,1.00)
	Adaptation plan A_4		
s_1	(0.25, 0.5, 0.75)	(0.75,1.00,1.00)	(0.75,1.00,1.00)
s_2	(0.75,1.00,1.00)	(0.25, 0.5, 0.75)	(0.75,1.00,1.00)
s_3	(0.75,1.00,1.00)	(0.25, 0.5, 0.75)	(0.75,1.00,1.00)

TABLE 9
Fuzzy decision matrix

	pr_1	pr_2	pr_3
A_1	(0.58, 0.83, 0.92)	(0.75, 1.0, 1.0)	(0.75, 1.0, 1.0)
A_2	(0.75, 0.92, 0.92)	(0.75, 1.0, 1.0)	(0.75, 1.0, 1.0)
A_3	(0.75, 1.00, 1.00)	(0.58, 0.83, 0.92)	(0.50, 0.67, 0.75)
A_4	(0.58, 0.83, 0.92)	(0.42, 0.67, 0.83)	(0.75, 1.00, 1.00)
Weight	(0.08, 0,17, 0.42)	(0.42, 0.67, 0.83)	(0.58, 0.83, 0.92)

where $\hat{w}_{ij}$ and $\hat{x}_{kij}$ are fuzzy numbers of the form (a_1, a_2, a_3) and $\oplus$ is a sum operator such that $\hat{w}_{1j} \oplus \hat{w}_{2j}$ ($\hat{x}_{k1j} \oplus \hat{x}_{k2j}$, resp.) is equal to ($a_1$+$a'_1$, a_2+a'_2, a_3+a'_3). For instance, in Table 7, the weights of the first property are (0,0,0.25), (0,0,0.25), and (0.25,0.5,0.75). The corresponding aggregated fuzzy weight is calculated as $\hat{w}_1$ $\big((0+0+0.25)/3,(0+0+0.25)/3,(0.25+0.5+0.75)/3\big)$=(0.08,0.17,0.42) (weight of the first property in Table 9). The same reasoning applies to the calculation of the aggregated fuzzy rating.

Once the decision matrix in Table 9 is built, we choose a representative crisp value for each weight and property, as the average of the three corresponding fuzzy numbers (a, b, c) of linguistic variables. For instance, in Table 9, the weight $\hat{w}_1$ of the first property is (0.08,0.17,0.42). The corresponding crisp value is (0.08+0.17+0.42)/3=0.22 (weight of the first property in Table 10).

TABLE 10
Crisp values for the decision matrix and property weights

	pr_1	pr_2	pr_3
A_1	0.78	0.92	0.92
A_2	0.86	0.92	0.92
A_3	0.92	0.78	0.64
A_4	0.78	0.64	0.92
Weights	0.22	0.64	0.78

The VIKOR approach can now be applied following the five steps in Section 4.2.

Step 1. Find positive-ideal and negative-ideal for all properties based on crisp values in Table 10 as described in equations (1) and (2).

pr^{a+}_1=0.92, pr^{a+}_2=0.92, pr^{a+}_3=0.92
pr^{a-}_1=0.78, pr^{a-}_2=0.64, pr^{a-}_3=0.64

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

Steps 2 and 3. Compute values U (group maximum utility), Z (minimum individual regret) and Q (sorting index) (Table 11) as described in equations (4), (5), and (6).

TABLE 11
The values of U, Z and Q for all alternative adaptations

	A_1	A_2	A_3	A_4
U	0.22	0.09	1.10	0.86
Z	0.22	0.09	0.78	0.64
Q	0.16	0.00	1.00	0.78

Step 4. Rank the alternative adaptations by U, Z, and Q in ascending order (Table 12). The ranking permits to understand whether there is an advantage in choosing one adaptation plan over the others, or if they are similar.

TABLE 12
The ranking of adaptation plans according to U, Z, and Q

	1	2	3	4
U	A_3	A_4	A_1	A_2
Z	A_3	A_4	A_1	A_2
Q	A_3	A_4	A_1	A_2

Step 5. Table 12 shows that adaptation plan A_3 is best ranked by Q. Additionally, both conditions 1 and 2 are satisfied, which means that A_3 is also best ranked by U and Z. Therefore, A^*=A_3. This means that A_3 has an acceptable advantage over the other adaptation plans.

5.2 Adaptation enforcement

The best adaptation plan A^*={(C,C.l3),(A,A.l2),(DS,DS.l1)} (adaptation A_3 in Table 4) returned by VIKOR is enforced according to the tenants' certificates. We recall that each certificate awarded to a tenant s_i specifies a complete configuration [pr_{ij},mec], where pr_{ij} is the certified property and mec is a (set of) mechanism supporting the property. The goal here is to enforce the selected adaptation in a way that minimizes changes to the certified configurations. The first step identifies the equivalent configurations (see Definition 6) compatible with A^* according to the feature model in Figure 1. Table 13 presents the retrieved equivalent configurations.

TABLE 13
Equivalent configurations for A^*.

Property	Equivalent configuration
pr_1	[(C, T.l3),DHE-RSA-AES256-SHA]
	[(C,T.l3),AES256-SHA]
pr_2	[(A, A.l2),x≤Redundancy<y]
	[(A,A.l2),Hot DR]
pr_3	[(DS, DS.l1),Cold SSD]

The second step executes all adaptation activities. For conciseness, we discuss adaptation enforcement for tenant s_3 only, supporting the following configurations: {[(C,T.l2),RC4-MD5],[(A,A.l3),Multi-Cloud HA],[(DS, DS.l1),Cold SSD]}.

- Configuration Cf_{11}=[(C,T.l2),RC4-MD5]: two more restrictive configurations exist for Cf_{11} in Table 13, namely [(C,T.l3),DHE-RSA-AES256-SHA] and [(C,T.l3),AES256-SHA]. As discussed in Section 4.3, to minimize the number of adaptations done at each enforcement process and to guarantee stable quality of service, original configuration Cf_{11} is either restored or kept as is. If Cf_{11} is not working correctly, a new configuration is randomly selected among the two more restrictive configurations.
- Configuration Cf_{12}=[(A,A.l3),Multi-Cloud HA]: no equivalent or more restrictive configurations exist for Cf_{12} in Table 13. Cf_{12} is downgraded to either [(A,A.l2),x≤Redundancy<y] or [(A,A.l2),Hot DR].
- Configuration Cf_{13}=[(DS,DS.l1),Cold SSD]: Cf_{13} is the only possible configuration according to Table tab:possiblemechanisms. Adaptation enforcement either restores Cf_{13} or no operation is done.

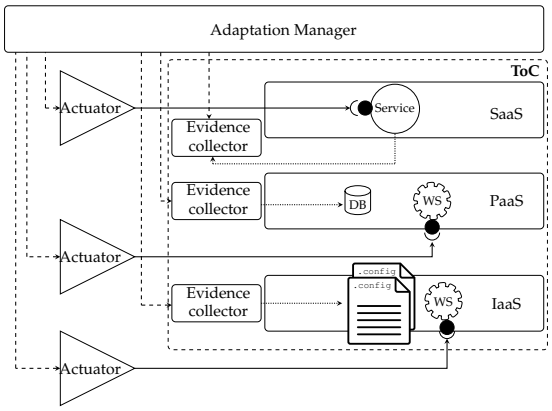


Fig. 2. A simplified view of the framework architecture

Before executing the above activities, the selected configurations are checked for consistency against the feature model in Figure 1. Since no inconsistencies are found, the adaptation activities are executed, and the enforced plan is monitored against hidden dependencies.

6 Experimental Evaluation and Simulation

We experimentally evaluated the performance and quality of our adaptation approach in a simulated environment with a wide range of settings.

6.1 Adaptation framework architecture and simulation environment

Our adaptation framework has the main responsibility of supporting the provisioning of an autonomic cloud with definite behavior of its applications and stable quality of service. To this aim, its architecture and components (Figure 2) support three main functionalities: *i)* evidence collection at the basis of certificate validity verification, *ii)* non-functional property verification and adaptation plan selection, *iii)* adaptation plan enforcement.

The framework is built around *evidence collectors*, which are responsible for collecting the evidence needed to either award a certificate or verify its validity. The collectors exercise the mechanisms in the ToC supporting the properties, at all layers of the cloud stack. This is achieved by executing test cases or monitoring events to evaluate the behavior of the system under certification. Data streams

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

from the collectors are connected to the *adaptation manager*, which implements the multi-constrained adaptation process in Section 4. In particular, it analyzes the collected data to uncover misconfigurations and trigger new adaptation processes. Upon misbehavior identification, the adaptation manager selects the best adaptation plan using VIKOR. It then *i)* configures and deploys actuators and *ii)* connects them to the mechanisms in the ToC to enforce the selected plan. Each *actuator* is responsible for enforcing a portion of the selected adaptation plan, by reconfiguring/replacing one or more mechanisms in ToC. In particular, it receives as input the mechanism(s) to be adapted and the adaptation activities to be executed, and returns as output the adaptation results. Once all actuators complete their activities, the adaptation plan enforcement is verified by the adaptation manager.

All our experiments have been run on a virtual workstation hosted at cloudlab (www.cloudlab.us) equipped with *16 CPUs Intel(R) Xeon(R) CPU D-1548 @ 2.00GHz and 512 GB of RAM*, and running *Ubuntu 16* OS. Our simulation environment includes a set of tools at the basis of the architecture in Figure 2: *i)* a custom tool we developed to find the valid adaptations, *ii)* FaMa tool for feature model management,[1] and *iii)* R with the MCDM package to support VIKOR method computation.[2] To avoid biased results due to the specific selection of the adaptation parameters, experiments have been run on a diverse range of parameters.

6.2 Performance

We evaluated the performance of our adaptation approach by measuring the execution time for adaptation plan selection. We considered a cloud environment consisting of a number $|s|$ of tenants varying between 10 and 60, and competing for resources at PaaS/IaaS layers. We defined the feature model with a number $|\hat{pr}|$ of property classes varying between 10 and 60, a number $|l|$ of levels varying between 10 and 60, and a number $|Cf|$ of configurations for each level varying between 1000 and 20000. Each tenant was certified for all property classes $\hat{pr}$ in the feature model. We executed a total of 15000 adaptation plan selection processes and the results shown here are the average over them. Each run of the experiments generated a new setting and a new adaptation plan without enforcing it. The performance of the enforcement process in fact can substantially vary depending on the selected environment and therefore is not a good estimator of the overall performance. We note that we also did not consider the time needed to generate the initial set of possible configurations (before applying the hard constraints filter), since this process is usually executed once, offline, before the adaptation process.

Figures 3(a)-3(c) show the execution time when the number of tenants, property classes, and levels is varied between 10 to 60 (step 10), respectively. Each scenario has then been evaluated in 4 settings varying the remaining two parameters in (3,3), (3,20), (20,3), (20,20). Figure 3(d)-3(f) shows the execution time when two parameters among number of tenants, property classes, and levels, are selected and varied between (10,10) and (60,60) (step 10). Each scenario has then been evaluated in 4 settings varying the remaining parameter in 3, 5, 10, 20. The black line with triangular tics in Figures 3(a)-3(f) presents our worst case scenario in which the number of property classes, levels, and tenants vary between (10,10,10) and (60,60,60) (step 10). The execution time of the adaptation algorithm is linear when varying each single parameter alone (tenants, properties and levels), or a pair thereof; the overall complexity increases exponentially when all variables increase at the same time.

1. http://www.isa.us.es/fama/?FaMa_Framework
2. https://cran.r-project.org/web/packages/MCDM/index.html

TABLE 14
Summary of quality evaluation

$Avg(\Theta(A^*))$	$\sigma^2(A^*)$	$\sigma^2(\overline{A})$
0.78	0.33	0.51

6.3 Quality

We evaluated the quality of our VIKOR-based adaptation process on a subset of 1000 adaptation plans A^* selected to cover the different settings in Section 6.2. Our goal was to compare our approach and the optimum adaptation with best quality selected without balancing the requirements of each tenant. To this aim, we have defined a penalty metric P_k, which indicates the degree to which requirements of all tenants diverge from an adaptation plan A_k, as follows.

Definition 8 (Penalty P_k). Penalty P_k introduced by an adaptation plan A_k with reference to tenants' requirements $R_i \in R$ is calculated as

$$P_k = \sum_{i=1}^{m}\sum_{j=1}^{n} \begin{cases} w_{ij} \times \frac{d(pr_{ij}, pr^a_{kj})}{d(\max(pr_j), pr^a_{kj})}, & pr_{ij} > pr^a_{kj} \\ 0, & otherwise \end{cases} \quad (10)$$

where: *i)* $pr_{ij} \in R_i$ is the j-th property certified for the i-th tenant, *ii)* $pr^a_{kj} \in A_k$ is the j-th property of adaptation plan A_k, *iii)* $\max(pr_j)$ is the property of class $\hat{pr}_j$ in the feature model with maximum level, and *iv)* w_{ij} is the crisp value corresponding to the fuzzy weight $\hat{w}_{ij}$ defined by tenant s_i on property pr_{ij} and calculated as the average of the three corresponding fuzzy numbers.

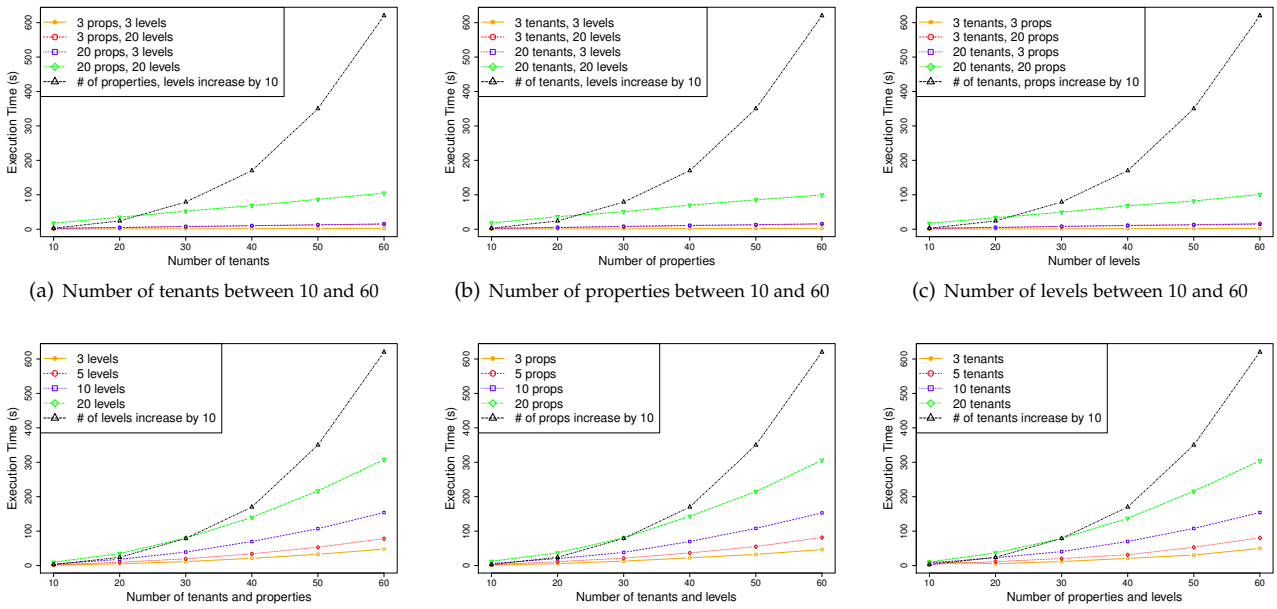
We note that penalty P_k is calculated as the sum of penalties observed by each single tenant $P_k(i)$, formally, $P_k = \sum_{i=1}^{m} P_k(i)$. Quality Θ of the selected adaptation process A^* is then calculated as:

$$\Theta = 1 - \frac{P^* - \min(P_k)}{\max(P_k) - \min(P_k)} \; with \; k = 1, .., m \quad (11)$$

where, *i)* $\max(P_k)$ and $\min(P_k)$ are the maximum and minimum penalty among all adaptations A_k and *ii)* P^* is the penalty of A^*. Θ=0 is retrieved when the valid adaptation with lowest quality is selected, Θ=1 is retrieved when the valid adaptation with highest quality ($\overline{A}$) is selected.

We evaluated the fairness of our VIKOR-based approach against the one producing the adaptation with maximum quality (global optimum). Table 14 shows our results. We first calculated the average quality $Avg(\Theta(A^*))$ of the 1000 adaptations A^* retrieved with our approach. Our approach captures 78% of the quality of the optimum approach on average (i.e., $Avg(\Theta(A^*))$=78%), with each single adaptation quality $\Theta(A^*)$ between 70% and 85%.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing



(a) Number of tenants between 10 and 60 (b) Number of properties between 10 and 60 (c) Number of levels between 10 and 60

(d) Number of tenants/properties between 10 and 60 (e) Number of tenants/levels between 10 and 60 (f) Number of properties/levels between 10 and 60

Fig. 3. Execution times based on varying numbers of tenants, classes of properties, levels

We then compared, for each single adaptation process, the variance $\sigma^2(A^*)$=$Var(\{P_k(i)\})$ and $\sigma^2(\overline{A})$=$Var(\{P_k(i)\})$ of the penalties observed by each tenant i with our approach and the optimum approach, respectively. In 90% of the cases $\sigma^2(A^*)$ is lower than $\sigma^2(\overline{A})$, whereas in the remaining 10% of the cases they are the same. We note that the average variance $Avg(\sigma^2(A^*))$ of penalties over all adaptation processes is 0.33 with our approach, which increases to 0.51 with the optimum approach (i.e., $Avg(\sigma^2(\overline{A}))$=0.51). Finally, another important aspect is the distribution of the penalties among tenants or, in other words, the contribution each single tenant provides to the overall penalty in equation 10. This aspect is fundamental to avoid scenarios in which the penalty of the adaptation process observed by each tenant substantially varies, resulting in a random adaptation quality for the tenants. To this aim, for each adaptation process, we retrieved the highest normalized penalty $\frac{\max(P_k(i))}{P_k}$ among all tenants and calculated the average for both our and the global optimum solutions. Figure 4 shows that in our approach the highest normalized penalty for a tenant is around 0.2, while, in the global optimum, it raised to around 0.6.

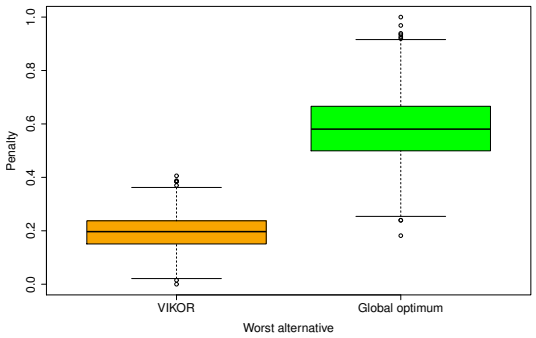


Fig. 4. Comparison between VIKOR and optimum approaches

7 Discussion and Related Work

We summarize the main characteristics and limitations of our approach, and compare it with the state of the art.

7.1 Discussion on practical usability and usefulness

We validated our adaptation approach in an environment designed to be more critical than the ones observed in the field at British Telecommunication and Huawei. Namely, we considered a worst-case scenario of 60 tenants sharing a single set of resources and 60 properties, each with 60 levels, a far greater number than the ones observed in practice. We then validated our approach along two lines, performance and quality, to measure its responsiveness in a fast and dynamic environment and its usefulness in addressing tenants' requirements, respectively. Performance evaluation showed that our approach computes and enforces an adaptation plan in a very short time. In particular, adaptation time is linear when two out of three parameters (i.e., tenants, properties, levels) are increased, while it becomes non-linear when all parameters are varied together. Even in the worst case, execution time is still bound by a reasonable upper limit of 600s. An average standard deviation of 7% (1.63s) has been observed, with 80% of the experiments below 10%. For the remaining 20%, the average standard deviation is 15% (0.97s) with the worst case achieving 21% (0.57s). We note that adaptation performance is not affected by adaptation plan enforcement that has been designed to integrate with cloud services and normal operations of the cloud provider, keeping interference with them at the minimum. Quality evaluation finally shows the usefulness

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

TABLE 15
Comparison with existing state of the art approaches
Legend: PA: Property Agnostic, CLA: Cross Layer Adaptation, TAP: Transparent Adaptation Process, TEC: Trusted Evidence Collection, DT: Dynamic Thresholds, DAC: Driven by Adaptation Costs, CA: Conciliatory Approach

	PA	CLA	TAP	TEC	DT	DAC	CA
Our	✓	✓	✓	✓	✓	×	✓
[14]	✓	×	✓	×	×	✓	×
[15]	×	×	✓	×	×	×	×
[16]	✓	×	✓	×	✓	×	×
[17]	✓	×	✓	×	✓	×	✓
[18]	×	×	×	×	✓	×	✓
[19]	✓	×	✓	×	✓	×	×
[20]	✓	×	✓	×	✓	✓	×

of the approach when compared with traditional solutions aiming to find the global optimum (see Table 15 in Section 7.2 for more details). Our technique finds a compromise solution among the requirements of all tenants, satisfying as much as possible their expectations in terms of hard and soft constraints; the global optimum solution can instead be perceived by tenants as acting randomly, as it may fully satisfy some of them while penalizing others, depending on the specific execution. Our compromise solution promises benefits for both cloud users and providers. Cloud users observe a substantial dampening of oscillations of property values within their SLAs; cloud providers can increase average user satisfaction balancing the observed penalty among users, and improve resource utilization addressing tenants' requirements with shared plans. We remark that, according to our experiments in Section 6.3, the highest penalty observed in a global optimum approach is reduced by 66% with the approach proposed in this paper.

Our VIKOR-based approach has some room for improvement in terms of cost management, the lack of which represents its main limitation. Our approach, in fact, selects the locally optimal adaptation plan regardless of its cost (e.g., in term of resource usage) and overhead (e.g., in term of delay due to service migration). Let us consider two adaptation plans A_1 and A_2, where *i)* A_1 is ranked first by VIKOR, though the difference in quality with A_2 is very small, *ii)* A_2 has substantially lower enforcement cost and overhead with respect to A_1. In this case, our approach selects A_1 as the best adaptation plan, while it could be better to select A_2. Our future work will target scenarios where costs and overheads should subvert VIKOR ranking.

7.2 Comparison with the state of the art

We compared our approach against the most relevant adaptation approaches for (cloud) services, which are briefly summarized in the following. Nallur and Bashoon [14] propose an adaptation approach for composite services, triggered by the Quality of Service (QoS) violations, which focuses on requirements of single tenants. Sliem et al. [15] present an approach for cloud self-adaptation, based on a reduction method and a model of the system as Stochastic Petri Nets, to find the best balance between performance and resource consumption. Calinescu et al. [16] present QoSMOS, a framework for adaptive service-based systems that focuses on QoS modeling to optimize resource allocation. Schroeter et al. [17] use an extended feature model (EFM) to represent variability of functionality and service quality, and handle runtime self-adaptive configurations. Arman et al. [18] propose a multi-objective, constraint-based adaptation for the cloud, implementing a consensus-based process and supporting multiple applications with different requirements. Alferez et al. [19] present a variability model to enable self-adaptation of service compositions, by activating or deactivating portions of the model. Leitner et al. [20] model the problem of finding the optimal adaptation plan with the lowest cost as a complex optimization problem and present a deterministic heuristic to prevent SLA violations in a cost-effective manner.

Table 15 summarizes the results of our comparison across several dimensions, inspired by the gaps identified in Section 3. Most of the existing adaptation solutions monitor a limited number of pre-defined properties, target a global optimum, and, in many cases, require manual intervention. Our adaptation approach supports any class of non-functional properties (G1) at all layers of the cloud stack (G2). Adaptation is a transparent and trustworthy end-to-end process, whose activities are triggered by trustworthy evidence based on certification (G3) that can be fully inspected and verified by any (trusted) party (G4). Our approach can cope with cloud contextual fluctuations (G5) thanks to a collaborative and conciliatory approach based on VIKOR, to reach an agreement that provides a maximum utility of the majority and a minimum individual regret. Finally, addressing a novel problem, our work is complementary to existing solutions for cloud adaptation and can be applied in conjunction with them. In particular, our solution can be applied in conjunction with both academic [6], [7] and industrial approaches [30], [31], providing a methodology for cloud configuration adaptation, to maintain stable non-functional properties in a multi-tenant environment.

8 Conclusions

We presented a certification-based adaptation technique for cloud services, which maintains stable non-functional behavior and quality of service of cloud-based systems across all tenants. Our approach relies on the VIKOR multi-criteria decision making method to find the best local optimum adaptation, which balances conflicting requirements of all tenants. It provides a multi-tenant, multi-layer, transparent, and user-informed adaptation based on trustworthy evidence coming from certification activities. Being based on a certification process, our adaptation technique *i)* can be applied to any non-functional properties, *ii)* considers all layers of the cloud stack, thus supporting cross-layer adaptations, *iii)* provides an increased transparency on adaptation techniques based on trustworthy and verified evidence, and *iv)* manages cloud fluctuations counteracting phenomena like oscillations.

References

[1] K. J. Sullivan, W. G. Griswold, Y. Cai, and B. Hallen, "The structure and value of modularity in software design," *SIGSOFT*, vol. 26, no. 5, 2001.
[2] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: a systematic review," *IEEE TCC*, vol. 1, no. 2, pp. 142–157, 2013.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TSC.2018.2793268, IEEE Transactions on Services Computing

[3] M. Almorsy, J. Grundy, and A. S. Ibrahim, "Tossma: A tenant-oriented saas security management architecture," in *Proc. of CLOUD 2012*, Honolulu, HI, USA, June 2012.

[4] Q. Liang, G. Zhao, F. Gioachin, and S. S. Chang, "A framework of scaling for inter-tenant collaborative systems," in *Proc. of CloudCom 2012*, Taipei, Taiwan, December 2012.

[5] S. Singh and I. Chana, "Qos-aware autonomic resource management in cloud computing: A systematic review," *ACM CSUR*, vol. 48, no. 3, p. 42, 2015.

[6] C. Inzinger, B. Satzger, P. Leitner, W. Hummer, and S. Dustdar, "Model-based adaptation of cloud computing applications." in *Proc. of MODELSWARD 2013*, Barcelona, Spain, February 2013.

[7] S. Farokhi, P. Jamshidi, I. Brandic, and E. Elmroth, "Self-adaptation challenges for cloud-based applications: a control theoretic perspective," in *Proc. of Feedback Computing 2015*, Seattle, USA, April 2015.

[8] M. Galster, D. Weyns, D. Tofan, B. Michalik, and P. Avgeriou, "Variability in software systems: A systematic literature review," *IEEE TSE*, vol. 40, no. 3, pp. 282–306, March 2014.

[9] C. Ardagna, R. Asal, E. Damiani, N. El Ioini, C. Pahl, and T. Dimitrakos, "A certification technique for cloud security adaptation," in *Proc. of SCC 2016*, San Francisco, CA, USA, June-July 2016.

[10] A. Bousquet, J. Briffaut, E. Caron, E. M. Dominguez, J. Franco, A. Lefray, O. López, S. Ros, J. Rouzaud-Cornabas, C. Toinard *et al.*, "Enforcing security and assurance properties in cloud environment," in *Proc. of UCC 2015*, Limassol, Cyprus, 2015.

[11] A. Lapouchnian, S. Liaskos, J. Mylopoulos, and Y. Yu, "Towards requirements-driven autonomic systems design," in *Proc. of DEAS 2005*, New York, NY, USA, 2005.

[12] M. Anisetti, C. Ardagna, E. Damiani, and F. Gaudenzi, "A semi-automatic and trustworthy scheme for continuous cloud service certification," *IEEE TSC*, 2017, to appear.

[13] M. Anisetti, C. Ardagna, E. Damiani, and N. El Ioini, "Trustworthy cloud certification: A model-based approach," in *Proc. of SIMPDA 2014*, Milan, Italy, November 2014.

[14] V. Nallur and R. Bahsoon, "A decentralized self-adaptation mechanism for service-based applications in the cloud," *IEEE TSE*, vol. 39, no. 5, pp. 591–612, 2013.

[15] M. Sliem, N. Salmi, and M. Ioualalen, "Towards modelling-based self-adaptive resource allocation in multi-tiers cloud systems," in *Proc. of IDCS 2015*, Windsor, UK, September 2015.

[16] R. Calinescu, L. Grunske, M. Kwiatkowska, R. Mirandola, and G. Tamburrelli, "Dynamic qos management and optimization in service-based systems," *IEEE TSE*, vol. 37, no. 3, pp. 387–409, 2011.

[17] J. Schroeter, P. Mucha, M. Muth, K. Jugel, and M. Lochau, "Dynamic configuration management of cloud-based applications," in *Proc. of SPLC 2012*, Salvador, Brazil, September 2012.

[18] A. Arman, S. Foresti, G. Livraga, and P. Samarati, "A consensus-based approach for selecting cloud plans," in *Proc. of IEEE RTSI 2016*, Bologna, Italy, September 2016.

[19] G. Alferez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, "Dynamic adaptation of service compositions with variability models," *Journal of Systems and Software*, vol. 91, pp. 24 – 47, 2014.

[20] P. Leitner, W. Hummer, and S. Dustdar, "Cost-based optimization of service compositions," *IEEE Transactions on Services Computing*, vol. 6, no. 2, pp. 239–251, 2013.

[21] N. Ferry, A. Rossini, F. Chauvel, B. Morin, and A. Solberg, "Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems," in *Proc. of the IEEE CLOUD 2013*, Santa Clara, CA, USA, June–July 2013.

[22] P. M. Mell and T. Grance, "Sp 800-145. the nist definition of cloud computing," Gaithersburg, MD, United States, Tech. Rep., 2011.

[23] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes, "Cloudscale: elastic resource scaling for multi-tenant cloud systems," in *Proc. of SOCC 2011*, Cascais, Portugal, October 2011.

[24] J. García-Galán, L. Pasquale, P. Trinidad, and A. Ruiz-Cortés, "User-centric adaptation analysis of multi-tenant services," *ACM TAAS*, vol. 10, no. 4, p. 24, 2016.

[25] S. Opricovic and G.-H. Tzeng, "Compromise solution by mcdm methods: A comparative analysis of vikor and topsis," *Elsevier EJOR*, vol. 156, no. 2, pp. 445–455, 2004.

[26] J. García-Galán, P. Trinidad, O. F. Rana, and A. Ruiz-Cortés, "Automated configuration support for infrastructure migration to the cloud," *FGCS*, vol. 55, pp. 200–212, 2016.

[27] V. Štuikys, *Smart Learning Objects for Smart Education in Computer Science: Theory, Methodology and Robot-Based Implementation*. Springer, 2015, pp. 185–209.

[28] M. Anisetti, C. Ardagna, and E. Damiani, "A test-based incremental security certification scheme for cloud-based systems," in *Proc. of SCC 2015*, New York, NY, USA, June-July 2015.

[29] C.-T. Chen, "Extensions of the topsis for group decision-making under fuzzy environment," *Fuzzy sets and systems*, vol. 114, no. 1, pp. 1–9, 2000.

[30] J. Daniel, F. El-Moussa, G. Ducatel, P. Pawar, A. Sajjad, R. Rowlingson, and T. Dimitrakos, "Integrating security services in cloud service stores," in *Proc. of IFIPTM 2015*, Hamburg, Germany, May 2015.

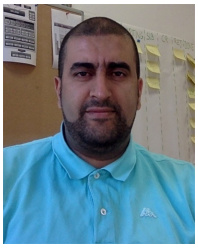
[31] J. Daniel, T. Dimitrakos, F. El-Moussa, G. Ducatel, P. Pawar, and A. Sajjad, "Seamless enablement of intelligent protection for enterprise cloud applications through service store," in *Proc. of CloudCom 2014*, Singapore, December 2014.

Claudio A. Ardagna is an Associate Professor at the Dipartimento di Informatica, Università degli Studi di Milano. His research interests are in the area of big data and cloud security and assurance. He is the recipient of the ERCIM STM WG 2009 Award for the Best PhD Thesis on Security and Trust Management. He has co-authored the Springer book "Open Source Systems Security Certification". The URL for his web page is http://www.di.unimi.it/ardagna

Rasool Asal is Chief Scientist at British Telecommunications Research and Innovation and a visiting professor to the Dipartimento di Informatica, Università degli Studi di Milano. His research interests include Cloud and Cyber Security, Information Assurance, Compliance Assessment, SDN, NfV. He is a speaker at many international conferences and events and has published research papers in leading international flagship conferences and journals.

Ernesto Damiani is a Full Professor at the Università degli Studi di Milano and the Director of the Khalifa University Information Security Centre in Abu Dhabi. He is the Principal Investigator of the H2020 TOREADOR project on Big-Data-as-a-Service. He was a recipient of the Chester-Sall Award from the IEEE IES Society (2007). Ernesto is ACM Distinguished Scientist (2008) and received the IFIP TC2 Outstanding Contributions Award (2012).

Theo Dimitrakos is the Head of Network Function Virtualization and Cloud Security research activities at the Huawei Technologies, Germany. He is also a Professor of Computer Science at the University of Kent, UK. His research expertise includes Cloud Security, Cybersecurity, and Information Assurance. Prior joining Huawei, he was a Chief Security Researcher at BT labs, UK. He has authored 7 books, over 70 scientific papers and over 40 patents.



Nabil El Ioini is a researcher at the Free University of Bolzano, Italy. He holds a Ph.D. in computing from the Free University of Bolzano. His research interests include: Software certification, Software Security and Software Testing, with a focus on Cloud Security and Cloud certification. He has co-authored the Springer book "Open Source Systems Security Certification".

Claus Pahl is an associate professor at the Free University of Bozen-Bolzano. Prior to his current employment he was a principal investigator of the Irish Centre for Cloud Computing and Commerce IC4. His research interests include software engineering in service and cloud computing. He holds a Ph.D. in computing from the University of Dortmund and an M.Sc from the University of Technology in Braunschweig.