Cost-Effective Deployment of Certified Cloud Composite Services

Marco Anisetti, Claudio A. Ardagna, Filippo Gaudenzi

DI – Università degli Studi di Milano, Italy

Ernesto Damiani

EBTIC – Khalifa University, UAE

Gwanggil Jeon

Incheon National University, Department of Embedded Systems Engineering, Incheon, Republic of Korea

Abstract

The advent of cloud computing has radically changed the concept of distributed environments, where services can now be composed and reused at high rates. Today, service composition in the cloud is driven by the need of providing stable QoS, where non-functional properties of composite services are proven over time and composite services continuously adapt to both functional and non-functional changes of the component services. This scenario introduces substantial costs on the cloud providers that go beyond the cost of deploying component services, and require to consider the costs of continuously verifying non-functional properties of composite and component services. In this paper, we propose a cost-effective approach to certification-based cloud service composition. This approach is based, on one side, on a portable certification process for the cloud evaluating non-functional properties of composite services and, on the other side, on a cost-evaluation methodology aimed to produce the service composition that minimizes the total cost paid by the cloud providers, taking into

[1]Email: *firstname.lastname@unimi.it*
[2]Email: *ernesto.damiani@kustar.ac.ae*
[3]Email: *gjeon@inu.ac.kr*

account both deployment and certification/verification costs. Our service composition approach is driven by certificates awarded to single services and by a fuzzy-based cost evaluation methodology, and assumes certified properties as *must-have* requirements for service selection and composition.

Keywords: Cloud, Certification, Cost Optimization, Service composition, Security

1. Introduction

The maturity reached by cloud computing has fostered the implementation of a number of distributed infrastructure, platform, and application services available worldwide. Current trends in software distribution and provisioning envision services made available as commodities over distributed systems including the Internet or the cloud marketplace. At the same time, the trend towards coarse-granularity business services, which cannot be managed by a single entity, resulted in several approaches to service composition that maximize software re-use by dinamically composing single services on the basis of their functionalities [1].

A major challenge faced by distributed service-based systems deployed on the cloud goes beyond the ability to guarantee the functionality of composite services, and must consider the importance of guaranteeing stable Quality of Service (QoS) in the form of non-functional properties requirements such as security, performance, and trust [2]. Service compositions need to guarantee optimal and verifiable properties, managing different events that might change their structure such as component relocation, substitution, malfunctioning, versioning, adaptation [3]. Continuous monitoring and verification of service non-functional properties is needed and usually achieved by means of assurance techniques [4, 5]. Recently, certification-based assurance techniques have been introduced to guarantee stable QoS in the cloud [6, 7, 8, 9, 10, 11]. They are based on continuous collection of evidence on the behavior of the system, which is used to verify whether the considered system holds a specific (set of) non-functional property and award a certificate proving it. To this aim, distributed agents are instrumented to connect to different endpoints in the cloud and retrieve evidence used to evaluate the non-functional status of the target cloud-based system. Current certification techniques mostly focus on the certification of single-service systems and often do not consider the cost of maintaining stable QoS. Even worse, a trend in service composition is to provide an ad hoc composite service for each request, with high costs on the cloud providers (CPs).

In this scenario, two colliding requirements emerge. On one side, there is the need to guarantee non-functional properties of a service composition. This is a challenging task that requires continuous evaluation of compositions at cloud-provider side, to accomplish the dynamic and evolving nature of the cloud. On the other side, there is the need to take the costs observed by cloud providers for certified composition management under control. These costs, in fact, rapidly increase because the costs of continuous certification and verification become substantial. Current research on cloud computing has privileged solutions minimizing costs on the final users [12, 13, 14], neglecting the costs on the cloud providers that often represent a major source of fee increase.

In this paper, we propose the first cost-effective approach to certification-based cloud service composition that addresses the above problems. It is inspired by our previous work in [4] and extends it according to the cloud challenges discussed in [11] and [15]. Differently from existing work [12, 13], our service composition approach is driven by certificates awarded to single services and by a fuzzy-based cost evaluation methodology, and assumes certified properties as must-have requirements for service selection [16] and composition [17]. This methodology aims to decrease the costs of cloud providers, also analyzing those costs introduced by the need of keeping the composition continuously monitored and certified. More specifically, the cost of deploying a certified service composition includes *i) direct costs*, traditional costs of service deployment on the cloud, or costs of third-party services building the composition (i.e., multi-cloud composition), *ii) indirect costs*, the costs introduced by the certification infrastructure to continuously monitor certificate validity, *iii) mismatch costs*, the costs modeling the discrepancy between what was agreed in terms of certified properties and what was actually provided. The mismatch costs are often neglected by existing approaches. They evaluate the additional costs observed by a CP when sharing a service whose properties in the certificate are stronger than the properties requested by a composite service. For instance, providing a storage service ensuring end-to-end confidentiality, while just confidentiality of data at rest is requested, means that resources for confidentiality in transit are overspent without a real revenue.[4]

The contribution of this paper is twofold. First, we present a certification process for composite services that fits the dynamics of the cloud (Sections 3 and 4). Our process guarantees continuous monitoring of certified properties, evaluating certificate validity over time and portability across different deployments. Second, after introducing the cost factors and profiles affecting the costs of cloud providers (Section 5), we provide a fuzzy-based cost evaluation methodology at the basis of a cost-effective, certification-based cloud service composition approach (Section 6). Our approach selects component services on the basis of their certified properties and is implemented by means of two run-time heuristics for composition cost minimization, which are experimentally evaluated in terms of quality and performance (Section 7). It contributes to the resolution of the long-standing problem of managing non-functional properties of distributed applications and composite services in a cost-effective way. It provides an approach that effectively relocates and refines service compositions in the cloud at run time guaranteeing stable QoS.

2. Problem Statement

Our reference model is a cloud infrastructure where single services are composed to form complex services and certification-based assurance techniques

[4]We note that mismatch cost is a crucial metric for internal cost optimization, permitting a more effective monitoring of requests vs offers.

are deployed for continuous QoS evaluation. The participating entities are: *i) cloud provider*, providing functionalities for service delivery and composition; *ii) composite service owner*, managing a service composition; *iii) certification authority*, providing functionalities for continuous non-functional property certification. Current approaches to service composition in the cloud are affected by a few limitations, which show a clear disalignment with the maturity reached by the cloud. These limitations, which are described in the following, must be addressed to provide a cost-effective service composition for the cloud with continuous QoS assessment.

- *Functional composition.* Service composition in the cloud puts great emphasis on functionalities. Component services are selected on the basis of the implemented functionalities, while overall non-functional aspects are, in most of the cases, pushed aside. For example, a composite e-Health service composes services for planning for a visit, access clinical reports, and get medicine prescription, a payment service, and a database/storage service. This practice however increases the likelihood of composite services that, on one side, satisfy the expectations of the users, while on the other side increase risks of failures and misbehaviors (e.g., privacy risks in the e-Health service). A proper approach to service composition must not only focus on functional requirements, but also consider non-functional requirements from the outset. For example, non-functional requirements may refer to security, privacy, reliability requirements, and can be addressed by proving specific properties such as confidentiality, integrity, availability, or showing compliance to specific standards/regulations, such as Payment Card Industry Data Security Standard (PCI-DSS), EU General Data Protection Regulation (GDPR).

- *Ad hoc composite services.* Service composition in the cloud often consists of ad hoc workflows, where component services are designed and developed for a specific composite service. For example, similarly to what happens with reserved instances and dedicated hosts in the cloud, component services are developed for and assigned to a specific, static service composition and never shared with other composite services. This approach substantially decreases the utility of service composition, from both a flexibility and a cost point of view. Having no possibility of sharing a single service among multiple service compositions bound current approaches to hold fashion monolithic service deployments. This approach is often adopted at infrastructure layer, where the huge amount of available resources often point to single tenant scenarios, where a user is usually provided with isolated resources not shared with other tenants. If, on one side, ad hoc composite services lower complexity of QoS evaluation and management, on the other side, it substantially increases costs and reduces the benefits of service compositions.

- *QoS evaluation.* It mostly focuses on deployment-time evaluation and on composition adaptation in case of component service malfunctioning/failure. QoS evaluation is however a more powerful concept that should represents a first-class requirement driving composition operations. First, it should be based on assurance (e.g., certification) techniques guaranteeing stable and verifiable QoS; then, it should consider how the QoS of a single service contributes to the QoS of the whole composition; finally, it should implement a continuous process that evaluates non-functional properties over time and drives adaptation of service compositions to provide stable QoS.

- *Direct costs.* The evaluation of service composition costs, which mainly focuses on direct costs due to component service integration, does not fit a multi-tenant cloud environment where *i)* services can be shared, relocated and migrated among different compositions and *ii)* non-functional properties are modeled as QoS requirements and integrated with the composite service life cycle. A proper cost evaluation at the basis of a cost-effective service composition must also consider the costs introduced by the infrastructure responsible for continuous QoS evaluation, and the costs introduced when QoS requested by the users are lower than the ones provided by the cloud infrastructure.

In the following of this paper, we provide a cost-effective, certification-based service composition approach for the cloud that fills in the above limitations. It is based on *i)* the concept of *portable certification*, supporting continuous QoS evaluation also in case of service migration and relocation and *ii)* a new cost evaluation methodology, considering direct, indirect, and mismatch costs on the cloud providers. We recall that our approach considers non-functional properties of composite services as must-have requirements; in other words, the QoS requirements of composite services are satisfied by design following our certification-based service composition. The design of an approach where QoS requirements in the form of non-functional properties in certificates are relaxed is out of the scope of this paper and will be the target of our future work.

3. Basic Concepts

A certification scheme for the cloud implements a continuous process whose goal is to verify whether a cloud service holds a given (set of) property [18]. The cloud service under evaluation is referred to as *Target of Certification* (ToC). Properties p=($\hat{p}$,l), as defined by the Cloud Security Alliance (CSA) [19], are composed of a controlled name $\hat{p}$ (e.g., confidentiality of data in transit) and a level l modeling the strength of the supported property. Properties can be organized in a hierarchy based on their strength such that $p_i \leq p_j$ (meaning p_i is weaker than p_j) *iff* $p_i.\hat{p}=p_j.\hat{p}$ and $p_i.l<p_j.l$. Based on levels l, a distance $Dist(p_i, p_j)$ between two properties with the same $\hat{p}$ is defined as:

$$Dist(p_i, p_j) = |p_i.l - p_j.l| \tag{1}$$

In this paper, without loss of generality, we consider security properties including, among the others, confidentiality, authentication, and data replication. For instance, property *confidentiality* can be further specified in properties *confidentiality at rest* and *confidentiality in transit*, each with three levels {AES128, AES192, AES256} and {TLS1.0, TLS1.1, TLS1.2}, respectively.

Our certification process is driven by a Certification Authority that manages all certification activities leading to certification. It is composed of two sub-processes: *i) evidence collection sub-process* and *ii) life cycle sub-process*. The *evidence collection sub-process* collects the evidence at the basis of a trustworthy certification and is carried out by the certification infrastructure. The *life cycle sub-process* implements a continuous certification process that accomplishes the evolution of the ToC, managing ToC migrations and versioning.

The certification process is based on two models, namely Certification Model (CM) Template and Instance, driving certification activities [20]. The certification authority defines CM Template $\mathcal{T}$ specifying evidence collection activities for a class of ToC and a (set of) property; the certification infrastructure implements and executes the corresponding CM Instance $\mathcal{I}$ specifying evidence collection activities for a given ToC instance and a (set of) property. Collected evidence is based on testing or monitoring, and permits to evaluate whether the observed ToC behavior conforms to the expected one. Upon a positive evaluation is retrieved following activities in $\mathcal{I}$, a certificate $cert_{\mathcal{I}}$ is released. Certificate $cert_{\mathcal{I}}$ is signed by the certification authority and contains: *i)* a description of the property certified for a given service, *ii)* a link to the ToC, and *iii)* a reference to the collected evidence and the relevant $\mathcal{I}$.

Certification Model Template ($\mathcal{T}$). It is a declarative model that describes the activities to be done to verify a set of properties according to the expected behavior of a class of ToC. Formally, a CM Template $\mathcal{T}_i$ is a triple (f_i, R_i,*d-eval*$_i$), where *i)* f_i is a functionality in the set F of functionalities offered by a cloud provider, *ii)* r_k is a user requirement in the set R_i of requirements used to annotate f_i, with $r_k \in R_i$ a property ($\hat{p}$,l), and *iii) d-eval*$_i$ is a declarative description of the evaluation activities to be carried out on the ToC to verify requirements R_i. $\mathcal{T}$ is built around *d-eval*, which is defined as a set of annotated workflows.

Definition 3.1 (*d-eval*). d-eval *is a pair* $\langle \phi, \omega \rangle$*, where:*

- ϕ *is a set of sequential workflows* $\{n_1,\ldots,n_n\}$ *for evidence collection, where* *each node n_i defines an abstract action (e.g., test authentication interface) and each edge (n_i,n_j) the flow between two actions.*

- *ω is an annotation function on nodes n. $\omega(\{n_i\})$ defines constraints (e.g., two factor authentication required) for a subset $\{n_i\}$ of abstract actions.*

We recall that *d-eval* refers to a generic class of *ToC* (e.g., an authentication system), while it precisely pinpoints security and deployment requirements (e.g., a given password strength policy). This means that, although there are a number of different *ToC* for the selected class, their evaluation w.r.t. security/deployment requirements should follow the same declarative description.

Certification Model Instance ($\mathcal{I}$). It is a procedural, executable model generated by instantiating $\mathcal{T}$ on a real *ToC*. It drives the certification process, including the evidence collection process. Formally, a CM Instance $\mathcal{I}_i$ is a triple (cs_i,$\mathcal{P}_i$,*p-eval*$_i$), where *i)* cs_i is the *ToC*, *ii)* $\mathcal{P}_i$ is the set of properties supported by $\mathcal{I}_i$, and *iii)* *p-eval*$_i$ defines certification activities as a concrete instantiation of *d-eval* for a specific *ToC*. $\mathcal{I}$ is built around *p-eval*, which covers the peculiarities of the specific *ToC* w.r.t. the given properties. *p-eval* is an annotated workflow defined as follows.

Definition 3.2 (*p-eval*). p-eval *is a triple $\langle\phi^{'}, \lambda\rangle$, where:*

- *$\phi^{'}$ is a set of sequential workflows $\{n_1,\ldots,n_n\}$ for evidence collection, where each node n_i defines an action implemented on the ToC instance and each edge (n_i,n_j) the flow between two implemented actions.*

- *λ is an annotation function. $\lambda(\{n_i\})$ defines the configuration settings of each action, describes how to deploy* p-eval*, and describes possible dependencies on its execution.*

We note that CM Instance $\mathcal{I}$ can be not unique for CM Template $\mathcal{T}$.

Example 3.1. *Let us consider a Certification Model Template $\mathcal{T}$=(*Storage*, Confidentiality via encryption at rest,* d-eval*), with* d-eval=$\langle\phi, \omega\rangle$*. For simplicity, we assume ϕ composed of a single sequential workflow $\{n_1, n_2, n_3\}$, where*

n_1=“ToC login”, n_2=“Test encryption”, n_3=“ToC logout”, and annotations $\omega(\{n_1\})$=[Administration credentials required], $\omega(\{n_2\})$=[Resource URI].

The same Certification Model Template $\mathcal{T}$ is instantiated in two different Certification Model Instances $\mathcal{I}$ for a Linux file system and Amazon Simple Storage Service (S3). Both instances drive a certification process and evidence collection activity targeting the same property “Confidentiality at rest via encryption”.

Let us first consider a Linux file system using LUKS. p-eval$_l$=$\langle\phi_l^{'}, \lambda_l\rangle$ *implementing the above* d-eval *is defined as follows:* $\phi_l^{'}$=*{SSH login, Script testing encrypted volumes, SSH logout}, $\lambda_l(\{n_1\})$=[root,cert], $\lambda(\{n_2\})$=Volume path. Let us then consider Amazon S3.* p-eval$_{s3}$=$\langle\phi_{s3}^{'}, \lambda_{s3}\rangle$ *implementing the above* d-eval *is defined as follows:* $\phi_{s3}^{'}$=*{Amazon login, API call for S3 configuration, Amazon logout}, $\lambda_l(\{n_1\})$=[credentials, APIkey], $\lambda(\{n_2\})$=[Config item].*

4. Portable Certification of Composite Services

We present a certification approach specifically tailored for cloud composite services, which is grounded on and extends the one in Section 3 to *i)* support service versioning, migration, and deployment changes (portability) and *ii)* accomplish the dynamics of service orchestrations where component services can be replaced and migrated at run time according to contextual events. In the following, we first describe the portability of our certification process and then describe how we use it in the framework of composite service certification.

4.1. Portability

A portable certification process is a certification process that is not bound to a specific *ToC* and can be easily applied to different service instances. It permits to apply the same certification process to different *ToC* with sufficient commonalities. Using our formalism, a certification process that derives from requirements in a template $\mathcal{T}$ can be re-used (with or without minor modifications) to certify all the services having an instance $\mathcal{I}$ consistent with $\mathcal{T}$. To verify this consistency we define a consistency check function, inspired by the work in [20], as follows.

Definition 4.1 ($\stackrel{I}{\rightarrow}$). *CM Instance* $\mathcal{I}_i$=(cs_i, $\mathcal{P}_i$,p-eval$_i$) *is consistent with CM Template* $\mathcal{T}_i$=(f_i, R_i,d-eval$_i$)*, denoted as* $\mathcal{T}_i \stackrel{I}{\rightarrow} \mathcal{I}_i$*, iff* i) cs_i *implements* f_i*,* ii) $\mathcal{P}_i$ *is such that* $R_i \leq \mathcal{P}_i$*, that is,* $\forall r_j \in R_i, p_j \in \mathcal{P}_i$*,* $r_j \leq p_j$*, meaning that the properties are stronger than the requirements according to property levels, and* iii) d-eval$_i \stackrel{i}{\rightarrow}$p-eval$_i$ *(see Definition 4.2), meaning that* p-eval$_i$ *is an instantiation of* d-eval$_i$*.*

Consistency check $\stackrel{I}{\rightarrow}$ is the cornerstone of process portability. A certification process can be implemented and executed using different instances $\mathcal{I}$, thanks to the decoupling between abstract definition ($\mathcal{T}$) and concrete actuation ($\mathcal{I}$) of the certification process. This decoupling also permits multiple consistent instantiations ($\mathcal{I}$) of the same process ($\mathcal{T}$). We note that, having $\mathcal{T}$ and $\mathcal{I}$ the same logical structure, $\stackrel{I}{\rightarrow}$ can be used to verify the consistency between two templates ($\mathcal{T}_i \stackrel{I}{\rightarrow} \mathcal{T}_j$) or two instances ($\mathcal{I}_i \stackrel{I}{\rightarrow} \mathcal{I}_j$).

As a complement to Definition 4.1, we detail how *p-eval* in $\mathcal{I}$ is checked for consistency against *d-eval* in $\mathcal{T}$.

Definition 4.2 ($\stackrel{i}{\rightarrow}$). p-eval$_i$=$\langle \phi', \lambda \rangle$ *is an instantiation of* d-eval$_i$=$\langle \phi, \omega \rangle$*, denoted as* d-eval$_i \stackrel{i}{\rightarrow}$p-eval$_i$*, iff* i) ϕ' *implements* ϕ*,* ii) *configurations* $\lambda(\{n_i\})$ *in* p-eval *instantiate constraints* $\omega(\{n_i\})$ *in* d-eval*,* iii) λ *permits the binding between each action in* ϕ' *and the corresponding end-point in the ToC.*

Definition 4.1 ($\stackrel{I}{\rightarrow}$) and Definition 4.2 ($\stackrel{i}{\rightarrow}$) are at the basis of a portable certification process that addresses two main scenarios: *service versioning* and *service replacement.*

Service versioning. It considers a single service that either is migrated as is to another location or evolves to a new version. It is defined as follows.

Definition 4.3 (Process Portability (Versioning)). *Let us consider a certification process driven by* $\mathcal{I}_i$=(cs_i, $\mathcal{P}_i$,p-eval$_i$) *for service* cs_i*, and a service* cs_k

such that either i) cs_i=cs_k *but they are deployed in different locations or* ii) cs_k *is the new version of* cs_i*. The certification process driven by* $\mathcal{I}_i$ *can be ported to* cs_k *iff* λ_i *is modified to connect* p-eval$_i$ *to* cs_k*.*

Process portability (versioning) properly configures the certification model instance in a way that permits the certification activities in *p-eval*$_i$ to connect to a different *ToC* (i.e., service cs_k). To this aim, λ_i of *p-eval*$_i$ must provide the new configurations required to connect each action to cs_k.

Service replacement. It considers a migration of a service to another service of the same class. For instance, a service implementing a MySQL database is migrated to a service implementing an SQLServer database. Process portability for service replacement is defined as described in the following definition.

Definition 4.4 (Process Portability (Replacement)). *Let us consider* $\mathcal{I}_i$ *=(*$cs_i, \mathcal{P}_i$*,* p-eval$_i$*) and* $\mathcal{I}_k$=($cs_k, \mathcal{P}_k$,p-eval$_k$*) such that* $cs_i \neq cs_k$*. The certification process driven by* $\mathcal{I}_i$ *can be ported to* $\mathcal{I}_k$ *according to the following conditions:*

- $\mathcal{I}_i \xrightarrow{I} \mathcal{I}_k$
- cs_i *and* cs_k *provide the same functionality* f_i*.*

Process portability (replacement) instantiates certification activities on different services cs_i and cs_k. To this aim, Condition 1 states that $\mathcal{I}_i$ is consistent with $\mathcal{I}_k$, and in turn their $\mathcal{T}$ are consistent as well. We note that the consistency at CM Instance level implies that *p-eval*$_k$ is equivalent to *p-eval*$_i$ (see Definition 4.2). In other words the workflows for evidence collection in *p-eval*$_i$ must be available also in *p-eval*$_k$ possibly with different annotation functions [17]. Details about this implications, and how to relax it are out of the scope of this paper, and will be detailed in future work.

Condition 2 states that cs_i and cs_k provide the same functionality f_i, which is specified in the corresponding templates $\mathcal{T}_i$ and $\mathcal{T}_k$. In other words, a certification process can be ported to a service or in an environment where the

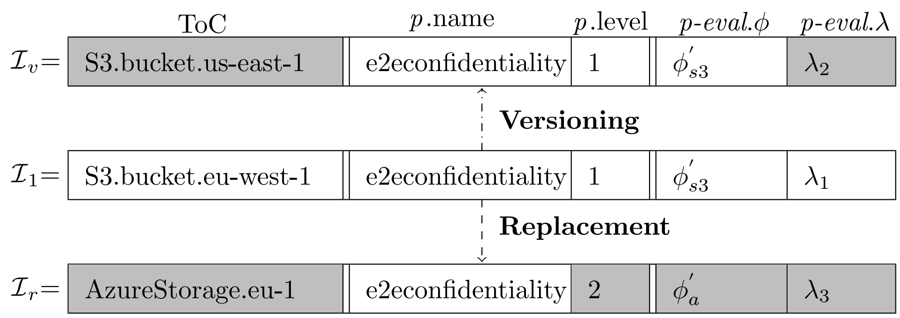


Figure 1: An example of Versioning and Replacement of a storage service

certification is driven by a different $\mathcal{T}$ without the need to re-build the certification process from scratch.

Example 4.1. *Let us consider a CM template for a storage service defined as follows* $\mathcal{T}_1$*=*$\{$*Storage,* $\{$*(e2econfidentiality,1)* $\}$,d-eval$_{storage}\}$*, where* end-to-end confidentiality *(e2econfidentiality) is requested (i.e., both confidentiality in transit and confidentiality at rest). Let us consider a storage service based on Amazon Simple Storage Service (S3) and specifically a bucket hosted on S3 eu-west-1 AWS region. Let us consider that this service has been certified according to the CM Instance* $\mathcal{I}_1$*=(*S3*.bucket.ue-west,*$\{$*(e2econfidentiality,1)*$\}$,p-eval$_{s3}$*) with* $\mathcal{T}_1 \xrightarrow{I} \mathcal{I}_1$.

Figure 1 also shows the case where this service is moved to a different region (versioning). In this scenario CM Instance $\mathcal{I}_1$ *is ported to* $\mathcal{I}_v$*=(*S3*.bucket.us-east-1,*$\{$*(e2econfidentiality,1)*$\}$,p-eval$_{s3}$*), where p-eval*$_{S3}$ *is re-configured to access the new bucket in the different region. We note that only parameters* λ_2 *are modified since the service is exactly the same but in different location.*

Figure 1 shows the case where this service is migrated to a different service (replacement). More specifically the service is replaced with an Azure Storage service which offers the same functionality and is certified for a given $\mathcal{T}_2$ *for the same property but with higher level (*$\{$*(e2econfidentiality,2)*$\}$*) with (*$\mathcal{T}_1 \xrightarrow{I} \mathcal{T}_2$*). The corresponding CM Instance* $\mathcal{I}_r$*=(AzureStorage.ue-1,* $\{$*(e2econfidentiality, 2)*$\}$*,* p-eval$_a$*) is compatible with* $\mathcal{I}_1$ *and can replace it. We note that, in a real* *environment, storage service replacement also implies functional compatibility at orchestration level and application data migrations.*

4.2. Composition

A certification process for composite services builds on our portable certification process and is driven by a compositional CM Template $\mathcal{T}^c$, where functional and certification requirements are specified for each component service. $\mathcal{T}^c$ is expressed as a set $\{\mathcal{T}_1, \ldots, \mathcal{T}_n\}$ of ordered templates, each to be linked to a component service. A certified service cs_i, having $cert_{\mathcal{I}_i}$, can be selected as a component service *iff* its $\mathcal{I}_i$ is consistent with the corresponding $\mathcal{T}_i$ in $\mathcal{T}^c$. We note that templates, including compositional templates, are specified by a CA and are the cornerstone of the certification chain of trust [21]. We extend Definition 4.1 to compositional instances ($\mathcal{I}^c$) and compositional templates ($\mathcal{T}^c$) as follows.

Definition 4.5 ($\xrightarrow{\mathcal{I}^c}$). *A Compositional Instance $\mathcal{I}^c_i$ is consistent with a Compositional Template $\mathcal{T}^c_i$, denoted as $\mathcal{T}^c_i \xrightarrow{\mathcal{I}^c} \mathcal{I}^c_i$, iff $\forall \mathcal{T}_k \in \mathcal{T}^c_i$, $\exists \mathcal{I}_j \in \mathcal{I}^c_i$ such that $\mathcal{T}_k \xrightarrow{I} \mathcal{I}_j$.*

The consistency check in Definition 4.5 supporting multiple consistent instantiations ($\mathcal{I}^c$) of the same certification process ($\mathcal{T}^c$) is at the basis of composition portability. It provides higher flexibility and lower costs, supporting automatic component substitution and reuse. Shared/reused components do not need to be evaluated multiple times, saving certification effort, and their management does not involve the certification authority.

Example 4.2. *Let us consider an e-Health service that allows patients to plan and pay for a visit, access clinical reports, and get medicine prescription. The e-Health service is a composite service that integrates i) a* Web App *providing access to e-Health functionalities,* ii) *a* Database *that gives access to patients' documents,* iii) *a* Payment *service allowing patients to pay for visits,* iv) *a* Storage *that stores all patients' documents. An e-Health composite service comes with* *strong security requirements: it must guarantee confidentiality of data and communications, and robustness against known vulnerabilities. A compositional CM Template* $\mathcal{T}^c$*=*$\{\mathcal{T}_1,\mathcal{T}_2,\mathcal{T}_3,\mathcal{T}_4\}$ *can then be defined as follow:* i) $\mathcal{T}_1$*=*{*WebApp, *{*(confidentiality-in-transit,3), (vulnerability-free,10)* }*,*$d-eval_{webapp}$} *meaning that the Web App must provide the service over an encrypted channel and be vulnerability free,* ii) $\mathcal{T}_2$=\{*DB,* {*(e2econfidentiality,3)* }*,*$d-eval_{db}$} *and* $\mathcal{T}_4$=\{*Storage,* {*(e2econfidentiality,3)* }*,*$d{-}eval_{storage}$} *meaning that the database and storage must be encrypted and exchange data over an encrypted channel (*e2econfidentiality*) with highest level (i.e., l=3),* iii) $\mathcal{T}_3$ = {*Payment,* {*(confidentiality-in-transit, 2)* }, $d-eval_{payment}$} *meaning that the payment service must provide the service over an encrypted channel (*confidentiality-in-transit*) with medium level (i.e., l=2).*

A consistent Compositional Instance $\mathcal{I}_i^c$ *of the above CM Template* $\mathcal{T}^c$ *can be defined as made by the following set of* $\mathcal{I}$, {{*psql.h-6, confidentiality.0,* $p-eval_{psql}$}, {*nginx.h-2, vulnerability-free.10,* $p-eval_{nginx}$}, {*S3.h-2, e2e-confidentiality.4,* $p-eval_{s3}$}, {*pay.remote, confidentiality,* $p-eval_{pay}$}}.

We note that, thanks to our portability (see Example 4.1), Amazon S3 (i.e., S3.$h{-}2$*) can be replaced with Azure Storage having instance* $\mathcal{I}_m$*=(AzureStorage.ue-1,e2econfidentiality.1,*$p-eval_{azurestorage}$*) leading to another consistent compositional instance* $\mathcal{I}_j^c$*.*

5. Deployment of Certified Composite Services

The enrichment of traditional composition solutions with certification techniques evaluating non-functional properties of composite services introduces the need of rethinking the algorithms driving selection of component services. If, on one side, service selection has been already renewed to accomplish selection of services that prove a set of non-functional properties [4], on the other side, solutions to cost-based service selection need to depart from the assumption that costs are only due to service deployment and resource consumption [12]. The latter must consider costs introduced by certification processes, and by the need of keeping the composition continuously monitored and certified.

5.1. Deployment Composition Matrix

The status of a given CP at time t can be represented as a matrix D of size $C \times F$ of deployed compositions $\mathcal{I}_i^c$, where C is the cardinality of deployed compositions at time t and F the cardinality of all possible functionalities provided by service providers. Matrix D has the following structure

$$D = \begin{array}{c} \\ \mathcal{I}_1^c \\ \mathcal{I}_2^c \\ \mathcal{I}_3^c \\ \vdots \\ \mathcal{I}_C^c \end{array} \begin{array}{c} \begin{array}{cccccc} f_1 & f_2 & f_3 & f_4 & \cdots & f_F \end{array} \\ \begin{pmatrix} \mathcal{I}_{1,1} & \mathcal{I}_{1,2} & \mathcal{I}_{1,3} & \mathcal{I}_{1,4} & \cdots & \mathcal{I}_{1,F} \\ \mathcal{I}_{2,1} & \mathcal{I}_{2,2} & \mathcal{I}_{2,3} & \mathcal{I}_{2,4} & \cdots & \mathcal{I}_{2,F} \\ \mathcal{I}_{3,1} & \mathcal{I}_{3,2} & \mathcal{I}_{3,3} & \mathcal{I}_{3,4} & \cdots & \mathcal{I}_{3,F} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathcal{I}_{C,1} & \mathcal{I}_{C,2} & \mathcal{I}_{C,3} & \mathcal{I}_{C,4} & \cdots & \mathcal{I}_{C,F} \end{pmatrix} \end{array} \tag{2}$$

where each row represents a composite service $\mathcal{I}_i^c$, each column a functionality f_j, and each cell a component service of $\mathcal{I}_i^c$ referred in the matrix with the corresponding CM Instance $\mathcal{I}_{i,j}$=($cs_{i,j}$,$\mathcal{P}_{i,j}$,$p\text{-}eval_{i,j}$). Each service $\mathcal{I}_{i,j}$ is annotated with a sharing level $k \geq 1$, specifying the number of compositions $\mathcal{I}_i^c$ insisting on it. In the following, we denote the component service $\mathcal{I}_{i,j}$ selected as part of the composition $\mathcal{I}_i^c$ as $\mathcal{I}_i^c.\mathcal{I}_j$.

Example 5.1. *Figure 2(a) shows an example of deployment composition matrix D with 4 functionalities (f) and 8 cloud services ($\mathcal{I}$), as follows:*

- *Functionality database (DB): mysql ($\mathcal{I}_1$) and posgresql ($\mathcal{I}_6$) are both certified for property confidentiality at different levels l.*

- *Functionality web application (WebApp): nginx ($\mathcal{I}_2$, $\mathcal{I}_4$, and $\mathcal{I}_7$) are certified for property vulnerability-free at different levels l. A level can refer to the severity of the Common Vulnerability Scoring System (CVSS) score related to the Common Vulnerabilities and Exposures (CVE) discovered on the target; the highest the level the lower the severity discovered.*

- *Functionality storage (Storage): Amazon S3 ($\mathcal{I}_8$) is certified for property end-to-end confidentiality.*

- *Functionality payment (Payment): a remote payment service ($\mathcal{I}_3$) is certified for property PCI-DSS compliance level 1 and the ENGPay remote payment service ($\mathcal{I}_5$) for PCI-DSS compliance level 3. Details about PCI-DSS compliance certification is available in [17] and in Section 7.1.*

These services are composed in 3 composite services $\mathcal{I}^c_i$ (Figure 2(a)): i) *an e-commerce service $\mathcal{I}^c_1$ based on Database $\mathcal{I}_1$, WebApp $\mathcal{I}_2$, payment service $\mathcal{I}_3$;* ii) *a shared-economy app $\mathcal{I}^c_2$ based on Database $\mathcal{I}_1$, WebApp $\mathcal{I}_4$, and payment service $\mathcal{I}_5$;* iii) *the e-health service $\mathcal{I}^c_3$ of Example 4.2 based on Database $\mathcal{I}_6$, WebApp $\mathcal{I}_7$, storage $\mathcal{I}_8$, and payment service $\mathcal{I}_3$. Figure 2(b) shows a graphical overview of the composite services highlighting shared component services, that is, $\mathcal{I}^c_1$ and $\mathcal{I}^c_3$.*

5.2. Cost Factors

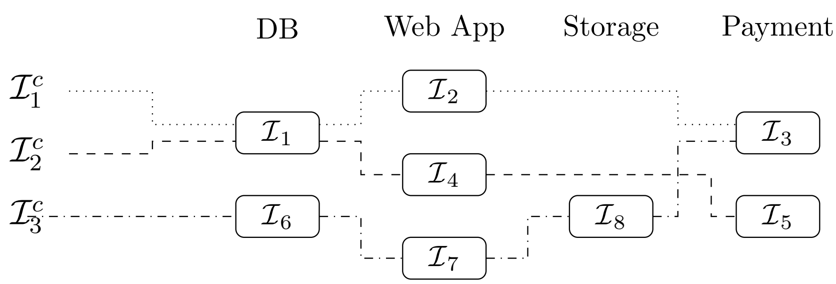
The management of a certified service introduces a cost on the cloud provider that can be evaluated using the Deployment Compositional Matrix and depends on three main factors: *direct (deployment) costs*, *indirect (certification) costs*, and *mismatch costs*. Each of these cost factors can be characterized using one of the four different cost behaviors identified by Horngren [22], and later used for cloud services by de Medeiros et al. [12]: *i) fixed costs*, a resource cost function that is completely independent from volume and time, indeed constant; *ii) variable costs,* a resource cost function that varies depending on volume or time, and is equal to zero for volume equal to zero; *iii) mixed costs,* a resource cost function that is the sum of a variable and a fixed cost function; and *iv) step costs,* a resource cost function that varies following different patterns.

Direct (deployment) costs ($\alpha(\mathcal{I}.cs,\ \mathcal{I}.\mathcal{P},\ \mathcal{I}.\mathrm{k})$). They are defined by the cloud provider as the amount of resources to be allocated to a cloud service *cs* w.r.t. the certified properties $\mathcal{I}.\mathcal{P}$. They are usually estimated as a mixture of fixed deployment and variable usage costs [12]. Direct costs comprise servers,

$$
\begin{array}{c} \\ \mathcal{I}^c_1 \\ \mathcal{I}^c_2 \\ \mathcal{I}^c_3 \end{array}
\begin{pmatrix}
DB & WebApp & Storage & Payment \\
\mathcal{I}_1 & \mathcal{I}_2 & - & \mathcal{I}_3 \\
\mathcal{I}_1 & \mathcal{I}_4 & - & \mathcal{I}_5 \\
\mathcal{I}_6 & \mathcal{I}_7 & \mathcal{I}_8 & \mathcal{I}_3 \\
 & & &
\end{pmatrix}
$$

$\mathcal{I}_1$ = {mysql.h-1, confidentiality, 1, ...}
$\mathcal{I}_2$ = {nginx.h-2, vulnerability-free, 4, ...}
$\mathcal{I}_3$ = {pay.remote, PCI-DSS compliance, 1, ...}
$\mathcal{I}_4$ = {nginx.h-4, vulnerability-free, 7, ...}
$\mathcal{I}_5$ = {ENGPay.remote, PCI-DSS compliance, 3, ...}
$\mathcal{I}_6$ = {psql.h-6, confidentiality, 0, ...}
$\mathcal{I}_7$ = {nginx.h-2, vulnerability-free, 10, ...}
$\mathcal{I}_8$ = {S3.h-2, e2e-confidentiality, 4, ...}

(a)



(b)

Figure 2: An example of CP Status Matrix D with 8 services and 3 compositions

infrastructure, power, networks, and personnel costs [23]. They also consider the cost of orchestrating the composition, and managing service versioning and migration. An appropriate cost behavior can be a step function that depends on properties $\mathcal{I}.\mathcal{P}$ and the sharing level k, that is, the number of compositions insisting on a given service. Figure 3(a) shows an example of a function of direct costs; we observe a small cost increase between k=2 and k=4 due to power consumption, and a more substantial increase from k=5 when a vertical scaling of resources is required to satisfy all requests.

Indirect (certification) costs ($\beta(\mathcal{I}.p\text{-}eval, \mathcal{I}.k)$). They are defined by the cloud provider with the support of the certification authority as the amount

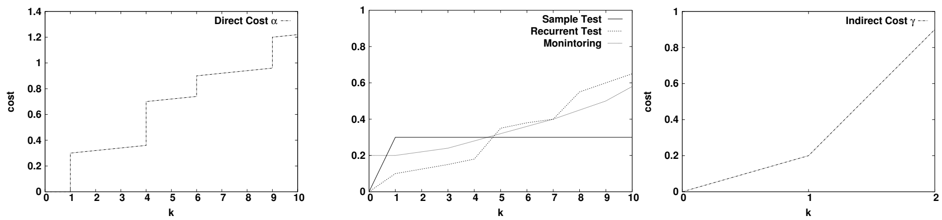


Figure 3: Cost functions

of resources to be allocated to the certification infrastructure (Section 7) for continuous evaluation of service compositions. We note that, to execute evidence collection in *p-eval*, our certification infrastructure considers three types of collection activities (Figure 3(b)) having different costs.

- **Sample test**: one time evaluation of a specific part of the ToC. No need to keep evaluation resources allocated after the evaluation.
- **Recurrent test**: a scheduled, repeatable evaluation of a specific target; it is often part of a complex evaluation. The evaluation resources are permanently allocated to re-execute the evaluation multiple times.
- **Monitoring**: continuous and permanent evaluation of the target.

Mismatch costs ($\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$). They are defined by the certification authority, as inefficiency due to the distance between provided properties $\mathcal{I}.\mathcal{P}$ and required properties $\mathcal{T}.R$, with $\mathcal{T}\xrightarrow{I}\mathcal{I}$. Providing higher security properties means in general allocating more resources than needed (e.g., more computational effort for encryption with 128-bit key when 32-bit key was required). This loss of resources depends on the distance $Dist(\mathcal{I}.p, \mathcal{T}.r)$ for each $p\in\mathcal{P}$ and corresponding $r\in R$, according to property levels in Section 3. Figure 3(c) shows an example of mismatch cost function for a property/requirement distributed over three levels. We note that the function is not necessarily homogeneous over the number of property levels. An important boost for higher levels might be observed, because high security levels may require more hardware facilities.

5.3. Cost Profiles

The CP behavior balances the contribution of direct, indirect, and mismatch costs to the computation of the total cost, while the total cost is the combination of each cost factor. We express the CP behavior as three cost profiles mapping to different CP strategies inspired by the deployment patterns in [24, 25].

- **Sharing profile** is typical of cloud providers that prefer to share resources, increasing the distance between requested and provided security properties (higher mismatch costs).
- **Fitting profile** is typical of cloud providers that prefer to achieve higher precision between requested and provided security properties, at the price of increasing the need of horizontal scaling. As a result, more component services are deployed precisely addressing users' needs, decreasing the average sharing level (higher direct and indirect costs).
- **Average profile** where direct, indirect, and mismatch costs equally contribute to the total cost.

We note that a degeneration of the fitting profile where an ad hoc composition is deployed for each request is a good approximation of the actual strategy adopted by cloud providers.

6. Fuzzy-Based Cost-Effective Deployment of Service Compositions

We propose a fuzzy-based cost evaluation approach, which evaluates the cost of composite services in Matrix D on the basis of cost factors and profiles in Section 5. Our solution extends the one in [15] to provide a more accurate infrastructure and easy to tune approach for cost evaluation and profile setting. In fact, the solution in [15] assumed uniform cost factors to balance their contribution to the final cost, and required a difficult and inaccurate tuning of the cost profiles on the needs of the cloud providers.

Fuzzy logic has been applied successfully in many fields including software cost estimation [26]. Let Y be the universe of discourse containing cost values

y (i.e., cost factors in this paper). As customary, the membership degree of element y to a fuzzy set S is characterized by a membership function $\mu_S(y)$. A fuzzy set S in Y is denoted as follow.

$$S = \{(y, \mu_S(y)) | y \in Y)\} \tag{3}$$

where $\mu_S(y)$ is the membership function of the fuzzy set S. Let us then consider each cost function α, β, γ as a separate universe of discourse Y_α, Y_β, Y_γ. We define a standardized partition of each of them into different fuzzy sets. In the following, we discuss Y_α; the same discussion holds for Y_β and Y_γ.

We define a standard fuzzy partition $\{S_{\alpha_l},\ S_{\alpha_m},\ S_{\alpha_h}\}$ of Y_α such that $\forall y_\alpha \in Y_\alpha,\ \sum_{j\in\{l,m,h\}} \mu_{S_{\alpha_j}}(y_\alpha)=1$. Each fuzzy set corresponds to a linguistic concept, that is, *LOW* for S_{α_l}, *MEDIUM* for S_{α_m}, and *HIGH* for S_{α_h}. There are a number of different shapes for the membership functions related to each linguistic concept. In this paper, we use R-function and L-function for LOW and HIGH membership functions and trapezoidal function for MEDIUM membership function. Given these linguistic variables mapping the concepts of LOW, MEDIUM, and HIGH costs for each cost factor, we define different sets of fuzzy rules. The rules, one for each profile, are used by the fuzzy inference system to infer the cost (Fc) introduced by each single component service $\mathcal{I}^c_t.\mathcal{I}_j$ at time t.

Example 6.1. *Let us consider a component service* $\mathcal{I}^c_i.\mathcal{I}_j$ *($\mathcal{I}$ for brevity) to be deployed at time t, following a sharing profile. Examples of fuzzy rules can be defined as follows:*

If $\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k)$ *is HIGH and* $\beta(\mathcal{I}.\text{p-eval}, \mathcal{I}.k)$ *is not LOW and* $\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$ *is LOW* ***then*** *$Fc_t(\mathcal{I})$ is HIGH.*

If $\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k)$ *not HIGH and* $\beta(\mathcal{I}.\text{p-eval}, \mathcal{I}.k)$ *is not LOW and* $\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$ *is LOW* ***then*** *$Fc_t(\mathcal{I})$ is MEDIUM.*

If $\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k)$ *is LOW and* $\beta(\mathcal{I}.\text{p-eval}, \mathcal{I}.k)$ *is LOW and* $\gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})$ *is LOW* ***then*** *$Fc_t(\mathcal{I})$ is LOW.*

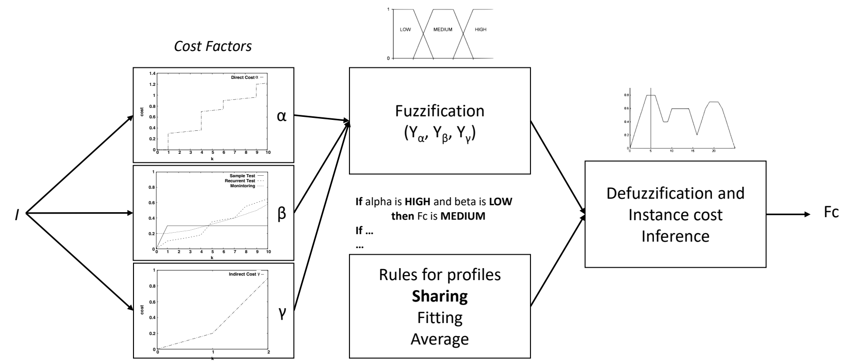


Figure 4: Fuzzy cost inference for a given $\mathcal{I}$ and considering rules set for sharing profile (in bold).

Figure 4 shows the overview of our approach for fuzzy-based cost computation where, given a specific $\mathcal{I}^c_t.\mathcal{I}_j$ at time t, cost factors α, β, and γ are mapped to fuzzy domains (Θ). This mapping is based on standard partitioning into different membership functions, which are specific for each cost function.[5] Then, for each cost profile, a set of rules are defined and executed (Ξ) to infer the cost of each single $\mathcal{I}^c_i.\mathcal{I}_j$ ($\mathcal{I}$ in Equation 4 for brevity), using a defuzzification function (Ψ), as follows.

$$Fc_t(\mathcal{I}) = \Psi(\Xi(\Theta(\alpha(\mathcal{I}.cs, \mathcal{I}.p, \mathcal{I}.k), \beta(\mathcal{I}.p\text{-}eval, \mathcal{I}.k), \gamma(\mathcal{T}.R, \mathcal{I}.\mathcal{P})))) \tag{4}$$

Considering the CP status Matrix in Equation 2, total fuzzy cost TFc_t is the sum of the cost of each deployed composite service $\mathcal{I}^c_i \in \{\mathcal{I}^c_1, \ldots, \mathcal{I}^c_t\}$, with $t \leq C$. TFc_t can be formally expressed as

$$TFc_t = \sum_{\substack{\mathcal{I}^c_i \\ 1 \leq i \leq C}} \sum_{\substack{\mathcal{I}^c_i.\mathcal{I}_j \\ 1 \leq j \leq |\mathcal{I}^c_i|}} Fc_t(\mathcal{I}^c_i.\mathcal{I}_j) \tag{5}$$

[5]Each CP tunes the membership functions of α and contributes to the tuning of the ones of β, while the membership functions of γ are defined by the CA.

where $\sum_{\substack{\mathcal{I}_i^c.\mathcal{I}_j \\ 1 \leq j \leq |\mathcal{I}_i^c|}} Fc_t(\mathcal{I}_i^c.\mathcal{I}_j)$ is the cost of a composite service $\mathcal{I}_i^c$ and calculated as the sum of the costs of the corresponding component services $\mathcal{I}_i^c.\mathcal{I}_j$.

We note that total fuzzy cost is not the real cost incurred for a given deployment. It represents the cost perceived by a cloud provider according to the selected profile (i.e., sharing, fitting average) and the mixture of the different cost sources (i.e., direct, indirect, mismatch).

6.1. Composition Cost Minimization

The aim of our solution is to find the best deployment $\{\mathcal{I}_1^c, \ldots, \mathcal{I}_n^c\}$ of composite services that *i)* satisfies a set $\{\mathcal{T}_1^c, \ldots, \mathcal{T}_n^c\}$ of composition requests, that is, guarantees $\mathcal{T}_i^c \xrightarrow{\mathcal{I}^c} \mathcal{I}_i^c$, with i=1,. . .,n, *ii)* minimize the cost TFc_n (Equation 5) for the CP. We assume that a new composition request $\mathcal{T}_i^c$ is received every time instant t, introducing a uniform time of arrival for composition requests. Finding the optimum deployment has however an exponential asymptotic behavior $\mathcal{O}\Big((|l| * F * t + t)^t\Big)$, in the worst case, with $|l|$ the number of property levels, F the number of functionalities, and t the number of composition request. Being t the dominating factor that varies over time, the exponential asymptotic behavior becomes $\mathcal{O}(t^t)$, which clearly does not fit the pseudo real-time requirement in our paper. It is therefore necessary to design heuristic approaches for solving the problem in polynomial time, even for relatively large composite services.

Many heuristic approaches balancing efficiency and quality in terms of precision and recall can be used for minimizing TFc_t at time t, though not all of them are applicable in a cloud environment, where *i)* composition requests are consecutive, *ii)* the requests may need to be served quickly. In our previous paper [15], we considered a simpler approach that analyzes each request $\mathcal{T}_t^c$ independently and aims to find $\mathcal{I}_t^c$ providing the lowest cost increment with respect to the total cost at time $t-1$. This approach however has some drawbacks, which could affect the quality of the retrieved results. First, it provides a static approach where each $\mathcal{I}_i^c$ remains for its entire life-cycle deployed in the original deployment slot; then, it supports real-time deployment of each incoming request opening the door to wrong choices that could materialize only in the subsequent requests.

We propose two heuristic algorithms: *i)* heuristic *sliding window* that selects a cloud service deployment within a time-forwarding window w of composition requests $\mathcal{T}_i^c$ and *ii)* heuristic *sliding window with migration* that extends the heuristic sliding window with the possibility of migrating component services $\mathcal{I}_i^c.\mathcal{I}_j$.

Heuristic 1: Sliding Window. It is based on the idea of finding the best solution at time t using a time-forwarding window w. The heuristic selects the best deployment $\mathcal{I}_t^c$ at time t by evaluating a set of $|w|$ consecutive requests $\mathcal{T}_i^c$, with $t \leq i \leq t+|w|$, received within a window w of size $|w|$. In other words, the selected $\mathcal{I}_t^c$ represents the composite service contributing to the global optimum within window w.

At time t, the heuristic receives as input the CP status matrix D, which contains all deployed $\mathcal{I}_i^c$ with $1 \leq i \leq t-1$, all costs with related de-fuzzyfication functions, window size $|w|$, and the total fuzzy cost TFc_{t-1}.

Upon collecting the last $|w|$ requests $\mathcal{T}_t^c, \ldots, \mathcal{T}_{t+|w|}^c$, the heuristic calculates all possible candidate sets $\mathcal{I}_t^c, \ldots, \mathcal{I}_{t+|w|}^c$. For each candidate $\mathcal{I}_t^c, \ldots, \mathcal{I}_{t+|w|}^c$, our heuristic calculates the total fuzzy cost $TFc_{t+|w|}$ and chooses the one that entails the minimum increase of cost. The minimum increase of cost is calculated as the difference between the total fuzzy cost $TFc_{t+|w|}$ within window w and the current total fuzzy cost TFc_{t-1}. Both $TFc_{t+|w|}$ and TFc_{t-1} are calculated using Equation 4. Once the deployment $\mathcal{I}_t^c, \ldots, \mathcal{I}_{t+|w|}^c$ with minimum cost increase is selected, $\mathcal{I}_t^c$ is instantiated to satisfy request $\mathcal{T}_t^c$; the window is then shifted of one time interval and the process restarts to satisfy request $\mathcal{T}_{t+1}^c$ when a new request is received at time $\mathcal{T}_{t+1+|w|}^c$.

We note that $|w|$ must be chosen carefully to balance the quality of the retrieved solution and the performance/complexity of the overall heuristic. This decision is left to the CP based on its requirements or preferences. We also note that a degeneration of this approach with a sliding window of dimension $|w|=1$

yields to the greedy approach presented in [15].

Heuristic 2: Sliding Window with Migration. It extends heuristic 1 with a better management of component deployment. Heuristic 2 supports service versioning and replacement (see Definition 4.3 and Definition 4.4), and in turn resource consolidation, for cost optimization. Migration in fact allows CP to modify its status matrix, moving to a new deployment scenario with lower costs. The global effect on the total cost, called *Migration Impact* (mi), is the difference between the total fuzzy cost TFc_t^{mi} after migration and the total fuzzy cost TFc_t before migration:

$$mi = TFc_t^{mi} - TFc_t \tag{6}$$

Migration impact $mi{<}0$ introduces a cost saving; migration impact $mi{\geq}0$ introduces a cost increase.

A migration is triggered when a new composition request $\mathcal{T}_t^c$ is processed and results in the deployment of a new cloud service $\mathcal{I}_i$ first instantiated in $\mathcal{I}_t^c$ ($\mathcal{I}_t^c.\mathcal{I}_i$). For clarity, we describe our heuristic using compositions with a single component service, since every functionality f is independent and therefore can be processed in parallel with the others. The migration process is composed of two sequential phases and 4 steps as follows.

1. *Service migration*: this phase aims to optimize the cost of composite services $\mathcal{I}_i^c$ in D at time $t{-}1$. In particular, it migrates component service $\mathcal{I}_i^c.\mathcal{I}_i$, such that $\mathcal{I}_i^c.\mathcal{I}_i \xrightarrow{\mathcal{I}} \mathcal{I}_t^c.\mathcal{I}_t$, to the new cloud service $\mathcal{I}_t^c.\mathcal{I}_t$ deployed at time t (step 0) according to mi. Phase service migration starts with an ordering process, which introduces a migration priority among deployed services. Services $\mathcal{I}_i^c.\mathcal{I}_i$ are sorted in descending order according to function property distance $Dist(\mathcal{T}_i^c.\mathcal{T}_i.r, \mathcal{I}_i^c.\mathcal{I}_i.p)$ (see Equation 1), where $\mathcal{T}_i^c.\mathcal{T}_i.r$ is the property originally requested for composition request $\mathcal{T}_i^c$ and $\mathcal{I}_i^c.\mathcal{I}_i.p$ is the property of the corresponding deployed composition $\mathcal{I}_i^c$ (step 1). Once all services are sorted, for each $\mathcal{I}_i^c.\mathcal{I}_i$, the migration impact mi is calculated and, if $mi{<}0$, $\mathcal{I}_i^c.\mathcal{I}_i$ is migrated to $\mathcal{I}_t^c.\mathcal{I}_t$ (step 2).

2. *Resource consolidation*: this phase considers all component services $\mathcal{I}_i^c.\mathcal{I}_i$ in the CP status matrix D migrated during the previous phase. Since each service instance $\mathcal{I}_i$ is shared among different composite services, a migration changes the level of sharing of $\mathcal{I}_i$ and introduces the need of a consolidation process to optimize the total fuzzy cost TFc (step 3). Resource consolidation is a *binary join* operation between two services $\mathcal{I}_i$ and $\mathcal{I}_j$, with $\mathcal{I}_i.p<\mathcal{I}_j.p$, which migrates all service composition $\mathcal{I}_i^c$ that are deployed on $\mathcal{I}_i$ to $\mathcal{I}_j$, that is, $\mathcal{I}_i^c.\mathcal{I}_i$ is migrated to $\mathcal{I}_i^c.\mathcal{I}_j$. Among all possible pairs ($\mathcal{I}_i$,$\mathcal{I}_j$), heuristic 2 chooses the one that offers the best mi. Resource consolidation is recursively executed until no ($\mathcal{I}_i$,$\mathcal{I}_j$) offers a negative mi.

Example 6.2. *Let us consider a CSP offering compositions with a single functionality f and a property p with 3 levels. In the following, we describe the working of heuristic 2 as an extended version of heuristic 1.*

Step 0 – New request $\mathcal{T}_t^c$ (Figure 5(a)). *A new request $\mathcal{T}_t^c$ at time t triggers the execution of heuristic 2. The status of the CSP is depicted in the status Matrix D in Figures 5(a), where each row represents the deployed composition $\mathcal{I}_i^c$, each column the deployed instance $\mathcal{I}_j$ offering functionality f and property p with level l, and each cell the request $\mathcal{T}_i^c.\mathcal{T}_j.r$ to be satisfied by the corresponding property p of $\mathcal{I}_j$. For instance, in Figure 5(a), cloud service $\mathcal{I}_1$ offers a property p at level 1 ($p = (\hat{p}, 1)$) and is shared by composite services $\mathcal{I}_1^c$ and $\mathcal{I}_3^c$, whose templates $\mathcal{T}_1^c$ and $\mathcal{T}_3^c$ require property $\mathcal{T}_1^c.\mathcal{T}_1.r$=$(\hat{p}, 1)$ and property $\mathcal{T}_3^c.\mathcal{T}_1.r$=$(\hat{p}, 1)$, respectively. In the following, for simplicity, we consider the same $\hat{p}$ for both r and p, and then refer to levels $r.l$ and $p.l$ only. At time t, composition request $\mathcal{T}_8^c$ with $r.2$ is received. A composition instance $\mathcal{I}_8^c$ of $\mathcal{T}_8^c$ is deployed on a new cloud service $\mathcal{I}_4$ (denoted with a gray background in Figure 5(a)) offering $p.2$. We note that the result of step 0 is both the final result of heuristic 1 and the initialization step of heuristic 2.*

Step 1 – Service ordering (Figure 5(b)). *All component services $\mathcal{I}_i^c.\mathcal{I}_j$, with $1 \leq j \leq 3$ and with $1 \leq i \leq 7$ in our example, are then sorted in descending order by*

	$\mathcal{I}_1$ (p.1)	$\mathcal{I}_2$ (p.3)	$\mathcal{I}_3$ (p.3)	$\mathcal{I}_4$ (p.2)
$\mathcal{I}_1^c$	*r.1*			
$\mathcal{I}_2^c$		*r.3*		
$\mathcal{I}_3^c$	*r.1*			
$\mathcal{I}_4^c$		*r.1*		
$\mathcal{I}_5^c$			*r.3*	
$\mathcal{I}_6^c$			*r.2*	
$\mathcal{I}_7^c$		*r.2*		
$\mathcal{I}_8^c$				*r.2*

(a) Step 0

composition.cloud service	$\mathcal{I}_2^c.\mathcal{I}_2$	$\mathcal{I}_5^c.\mathcal{I}_3$	$\mathcal{I}_7^c.\mathcal{I}_2$	$\mathcal{I}_6^c.\mathcal{I}_3$	$\mathcal{I}_1^c.\mathcal{I}_1$	$\mathcal{I}_3^c.\mathcal{I}_1$	$\mathcal{I}_4^c.\mathcal{I}_2$
property distance	(+1)	(+1)	(0)	(0)	(-1)	(-1)	(-1)

(b) Step 1

order	*migration*	*migration impact*	*action*	$\mathcal{I}_4(p.2)$
1	$\mathcal{I}_7^c.\mathcal{I}_2 \xrightarrow{\mathcal{I}^c} \mathcal{I}_7^c.\mathcal{I}_4$	mi=-2	migrate	$\mathcal{I}_t^c$,$\mathcal{I}_7^c$
2	$\mathcal{I}_6^c.\mathcal{I}_3 \xrightarrow{\mathcal{I}^c} \mathcal{I}_6^c.\mathcal{I}_4$	mi=-1	migrate	$\mathcal{I}_t^c$,$\mathcal{I}_7^c$,$\mathcal{I}_6^c$
3	$\mathcal{I}_1^c.\mathcal{I}_1 \xrightarrow{\mathcal{I}^c} \mathcal{I}_1^c.\mathcal{I}_4$	mi=+2	-	$\mathcal{I}_t^c$,$\mathcal{I}_7^c$,$\mathcal{I}_6^c$
4	$\mathcal{I}_3^c.\mathcal{I}_1 \xrightarrow{\mathcal{I}^c} \mathcal{I}_3^c.\mathcal{I}_4$	mi=+3	-	$\mathcal{I}_t^c$,$\mathcal{I}_7^c$,$\mathcal{I}_6^c$
5	$\mathcal{I}_4^c.\mathcal{I}_2 \xrightarrow{\mathcal{I}^c} \mathcal{I}_4^c.\mathcal{I}_4$	mi=-2	migrate	$\mathcal{I}_t^c$,$\mathcal{I}_7^c$,$\mathcal{I}_6^c$,$\mathcal{I}_4^c$

(c) Step 2

	$\mathcal{I}_1$ (p.1)	$\mathcal{I}_2$ (p.3)	$\mathcal{I}_3$ (p.3)	$\mathcal{I}_4$ (p.2)
$\mathcal{I}_1^c$	*r.1*			
$\mathcal{I}_2^c$		*r.3*		
$\mathcal{I}_3^c$	*r.1*			
$\mathcal{I}_4^c$				*r.1*
$\mathcal{I}_5^c$			*r.3*	
$\mathcal{I}_6^c$				*r.2*
$\mathcal{I}_7^c$				*r.2*
$\mathcal{I}_8^c$				*r.2*

(d) Step 3

	$\mathcal{I}_1$ (p.1)	$\mathcal{I}_2 \bigcup \mathcal{I}_3$ (p.3)	$\mathcal{I}_4$ (p.2)
$\mathcal{I}_1^c$	*r.1*		
$\mathcal{I}_2^c$		*r.3*	
$\mathcal{I}_3^c$	*r.1*		
$\mathcal{I}_4^c$			*r.1*
$\mathcal{I}_5^c$		*r.3*	
$\mathcal{I}_6^c$			*r.2*
$\mathcal{I}_7^c$			*r.2*
$\mathcal{I}_8^c$			*r.2*

(e) Step 3

Figure 5: An example of heuristic 2 execution

measuring distance $Dist(\mathcal{T}_i^c.\mathcal{T}_j.r, \mathcal{I}_4.p)$.

Step 2 – Service migration (Figure 5(c)). *The migration impact mi in Equation 6 is calculated for each of the* $\mathcal{I}_i^c$ *(denoted with a gray background in Figure 5(b)) showing a distance that is less or equal to zero (step 2). Figure 5(c) shows the results of our migration, that is,* $\mathcal{I}_6^c$ *is migrated from* $\mathcal{I}_3$ *to* $\mathcal{I}_4$*, and* $\mathcal{I}_4^c$ *and* $\mathcal{I}_7^c$ *are migrated from* $\mathcal{I}_2$ *to* $\mathcal{I}_4$*. We note that all these migrations are of type* *replacement as presented in Definition 4.4.*

Step 3 – Resource consolidation (Figures 5(d) and 5(e)). *Upon phase service migration ends, phase* resource consolidation *is executed and considers all* $\mathcal{I}_j$ *such that at least one composite service* $\mathcal{I}_i^c$ *insisting on it has been migrated to* $\mathcal{I}_4$. *In our example, service instances* $\mathcal{I}_2$ *and* $\mathcal{I}_3$ *are candidates for the binary join (denoted with a light grey background in Figure 5(d)). Since the join between* $\mathcal{I}_2$ *and* $\mathcal{I}_3$ *has a negative mi=-2, resource consolidation is convenient and applied. Figure 5(e) finally shows the new CP status after the execution of heuristic 2, where the result of the join operation is denoted with a gray background. We note that all migrations due to consolidation are of type versioning as presented in Definition 4.3.*

7. Experimental Evaluation

We experimentally evaluated the performance and quality of our approach for cost-effective deployment of service compositions, and the utility of our portable certification process.

7.1. Experimental Setup

We considered a scenario where a cloud provider hosts the three compositions depicted in Figure 2. For simplicity but with no lack of generality, we focused on the payment functionality only, which is used in all compositions. This choice was due to the fact that, as already discussed, considering the entire composition as a whole does not give any additional insights on the soundness of the proposed approach and its performance/cost. Our methodology in fact treats each functionality independently and the total cost is calculated as the sum of the cost of each functionality. CP offers two payment services, a standard payment service and ENGPay payment service offered by Engineering S.p.A., one of the biggest system integrators in Italy, all certified for property PCI-DSS compliance. We considered three different certification levels for property

PCI-DSS compliance $\mathcal{P}_c$ from basic confidentiality ($\mathcal{P}_c$.level=1) to full PCI-DSS ($\mathcal{P}_c$.level=3), via generic CIA – Confidentiality, Integrity, Authentication ($\mathcal{P}_c$.level=2). Standard payment service is certified for property PCI-DSS compliance at level 1; ENGPay is offered with two levels of certification, level 2 and level 3.

We developed a *request simulator* that randomly generates requests for a payment service with a specific property level. We then built 10 data sets of 300 consecutive random requests $\mathcal{T}^c$ submitted to the cloud provider. For all data sets, we evaluated the deployment obtained using the sliding window and migration heuristics in Section 6.1 with sharing and fitting profiles in Section 5.3. We evaluated retrieved results according to *i)* a set of evaluation metrics, *ii)* the fuzzy membership functions, and *iii)* the cost functions.

Evaluation metrics. We used three metrics to evaluate our approach.

Metric 1 measures the execution time needed to deploy composite services addressing composition requests.

Metric 2, called $\Gamma_t(TFc, TFc^{'})$, is the relative cost increment. It is calculated as the difference between the two areas identified by Total Fuzzy cost functions TFc and $TFc^{'}$ in the interval $[1, t]$. It is formally defined as follows:

$$\Gamma_t(TFc, TFc^{'}) = \frac{\sum_{i=1}^{t}(TFc_i - TFc_i^{'})}{\sum_{i=1}^{t} TFc_i} \tag{7}$$

where TFc_i and $TFc_i^{'}$ are the two Total Fuzzy cost functions evaluated at time i. We used Total Fuzzy cost to calculate Γ, since our goal here is to evaluate the overall cost increase and not the contribution of each cost factor (α, β, and γ).

Metric 3, called Δ_t, evaluates the cumulative number of portability events (versioning or replacement) occurred until a given time t. It provides a measure of how often a portability event and a consistency check are needed to support our dynamic composition certification, and in turns a measure of its utility.

Fuzzy membership functions. Our fuzzy system is based on membership functions and fuzzy rules that depend on the cost factors to be evaluated. We adopted the generalized bell-based memberships f for all cost factors

$$f(x; a, b, c) = \frac{1}{1 + |\frac{x-c}{a}|^{2b}} \tag{8}$$

where c is the center of the curve, a controls the width of the curve, and b controls the slope of the curve. To optimize the membership function definition, we evaluated the distribution of costs to adjust parameters a, b, c to the meaning of the corresponding linguistic variable. More precisely, we used a fuzzy c-mean approach to have an initial idea on the membership shapes using 100 requests from each of the 10 data sets. Given this shape we tuned the membership parameters to fit the fuzzy clusters with a *gbell* shape. We note that this process, as well as the cost function definition, is a tuning process that may depend on the cloud provider peculiarities. In general, the selected cost and membership functions are suitable for a generic cloud provider working with cloud service compositions, while the rule sets address the peculiarities of the profiles.

Cost functions. We used cost functions α, β, and γ in Figure 3 for the three property levels used in our experiments. We recall that cost function γ is defined using property levels and ranges from 0 to 2. α, β, and γ have been used to compare the cost retrieved by our heuristics (metric 2 and metric 3). Their definition is CP specific and should reflect the costs of the CP infrastructure. To fully evaluate our approach, we defined cost functions such that service migrations are triggered also with low numbers of composition requests.

We run our experiments on a Blade server PowerEdge M630 (VRTX) 2 x Intel(R) Xeon(R) CPU E5-2620 v4 2.10GHz 192GB of RAM 120GB SSD.

7.2. Performance evaluation

We evaluated the performance of our heuristics using *metric 1*, and cost factors α, β, and γ in Figure 3. Similar to the exhaustive algorithm, our heuristics have an exponential asymptotic behavior $\mathcal{O}\Big((|l| * F * |w| + t)^{|w|}\Big)$, with $|l|$ the number of property levels, F the number of functionalities, $|w|$ the size of window w, and t the number of received requests. We note that, since $|w|$ is fixed a priori by our heuristics, the asymptotic behavior becomes polynomial as $\mathcal{O}(t^{|w|})$. Window w however makes our heuristics rapidly unusable given our assumption to serve requests in pseudo real time (in the order of minutes), though their complexity is far lower than the one of the exhaustive algorithm, which is $\mathcal{O}(t^t)$.

Figure 6 compares the average execution time of the heuristics and exhaustive algorithm on the 10 data sets, varying window size$|w|$ from 1 to 7. We note that the execution time of all algorithms is reported only for configurations requiring less than 3-minutes. Sharing and fitting profiles show a similar performance trend just partially affected by optimizations based on branch cut. Heuristic 2 shows an additive execution time increment with respect to heuristic 1 due to the migration and consolidation algorithms, which require sorting of compatible requests ($\mathcal{O}((t-1)^2)$ at time t, in the worst case). This additive factor is not anyways substantial, since it depends on the presence and amount of possible migrations (e.g., between $t = 70$ and $t = 80$). Our results show that, as expected, the heuristics approximates polynomial execution time in the window size $|w|$, which can be taken under control by selecting proper $|w|$. For instance, when $|w|$=7, execution time exceeds the 3 minute limit with a number $|\mathcal{T}^c|$ of composition requests equal to 22 for heuristic 1 and 20 for heuristic 2; when $|w|$=6, execution time exceeds the 3-minute limit with $|\mathcal{T}^c|$=155 for heuristic 1 and $|\mathcal{T}^c|$=143 for heuristic 2. The exhaustive algorithm shows the worst execution time, exceeding the 3-minute limit with $|\mathcal{T}^c|$=12. We note that the two heuristics have comparable performance dominated by w. We also note that the exhaustive algorithm has better performance than our heuristics for a number $|\mathcal{T}^c| \leq |w|$ of requests, because the heuristics use a window size $|w|$, while exhaustive algorithm uses the entire set of requests. Therefore, when the number of requests is less than $|w|$, it provides better or comparable performance.

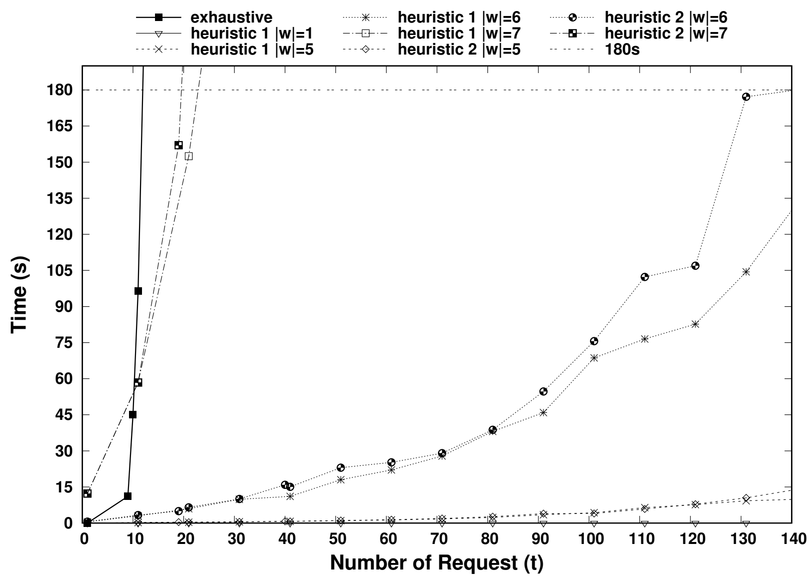


Figure 6: Performance evaluation: heuristic 1 and heuristic 2 varying window size $|w|$.

7.3. Cost and Utility Evaluation

Performance evaluation in Section 7.2 showed the unmanageable complexity of the exhaustive algorithm, which required 21 minutes for deploying 12 requests. We therefore compared the costs of our two heuristics on the 10 data sets and the utility of the portability underpinning them using *metric 2* and *metric 3*, and cost factors α, β, and γ. We first discuss the impact of windows size and then compare heuristic 1 and heuristic 2.

7.3.1. Window size

We measured the impact of window w on heuristic 1 and heuristic 2 using the relative cost increment (*metric 2* – Γ_t) with sharing and fitting profiles and the entire data sets of 300 requests. Table 1 shows the average relative cost increment ($\overline{\Gamma_t}$), expressed in percentage, over our 10 data sets, varying $|w|$. Fitting profile shows a negligible variation of $\overline{\Gamma_t}$ both for heuristic 1 and heuristic 2. Fitting profile in fact does not take significant advantages by looking forward in the incoming requests. In particular, Table 1 shows that the total

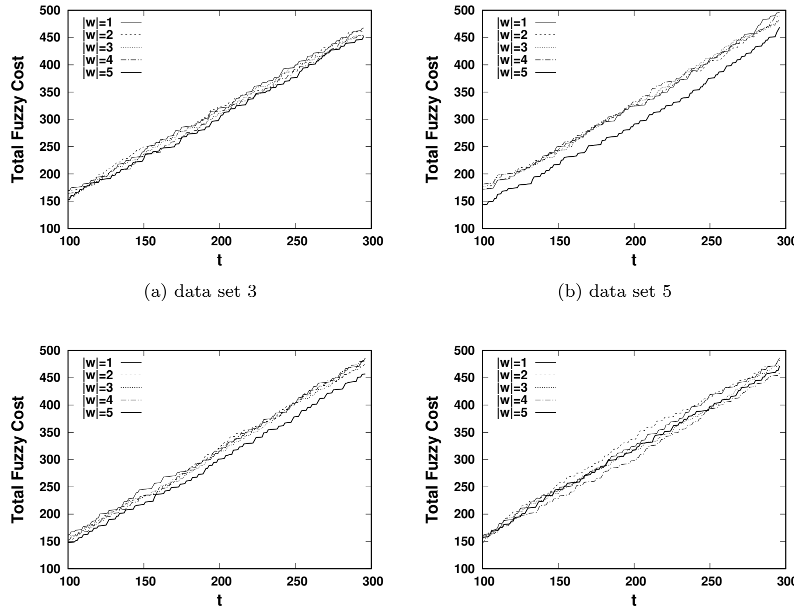
$\overline{\Gamma_t}$		$TFc_{w2}-TFc_{w1}$	$TFc_{w3}-TFc_{w2}$	$TFc_{w4}-TFc_{w3}$	$TFc_{w5}-TFc_{w4}$	$TFc_{w5}-TFc_{w1}$
heuristic 1	sharing	−0.6%	−1.5%	−1.31%	−2.08%	−5.48%
	fitting	0.6%	0.09%	−0.23%	−0.19%	0.29%
heuristic 2	sharing	−1.2%	−1.3%	−1.7%	−0.86%	−5.39%
	fitting	0.46%	0.06%	0.11%	0.007%	0.6%

Table 1: Average relative cost difference ($\overline{\Gamma_t}$ in %) at t=300 for heuristic 1 and heuristic 2, using sharing and fitting profiles, and varying window size $|w|$. We denote with TFc_{wi} the total fuzzy costs TFc with $|w|$=i.

fuzzy costs TFc_{w5} with $|w|$=5 increases on average of 0.29% with respect to the total fuzzy costs TFc_{w1} with $|w|$=1 (denoted as $TFc_{w5}-TFc_{w1}$ in Table 1) for heuristic 1 and of 0.6% for heuristic 2. When the sharing profile is adopted, the average cost decreases as the window size increases. Table 1 shows that the total fuzzy costs TFc_{w5} with $|w|$=5 decrease on average of −5.48% with respect to the total fuzzy costs TFc_{w1} with window $|w|$=1 for heuristic 1 and of −5.39% for heuristic 2.

Figure 7 shows an excerpt of the total fuzzy cost TFc of heuristic 1 for 4 representative data sets for sharing profile, varying the window size from $|w|$=1 to $|w|$=5. We note that an increase in the window size $|w|$ does not always result in a cost decrease. Figure 7(d) shows a data set where heuristic 1 with $|w|$=4 has lower total fuzzy cost than the one with $|w|$=5. This mainly depends on the bias introduced by the random generation of the data sets and by the random selection of the best deployment when different candidate deployments have the same total fuzzy cost. This latter scenario may lead to a sub-optimal deployment drifting from the optimal total cost and is more probable at the beginning of the deployment process where there are more deployments with the same cost.

Figure 8 shows an excerpt of the total fuzzy cost TFc of heuristic 2 for 4 representative data sets for sharing profile, varying the window size from $|w|$=1 to $|w|$=5. We note that the behavior of heuristic 2 (Figure 8) is similar to the one of heuristic 1 (Figure 7), which is reasonable considering the refinement nature of heuristic 2. We also note that *i)* the drifting effects causing sub-



(a) data set 3

(b) data set 5

(c) data set 8

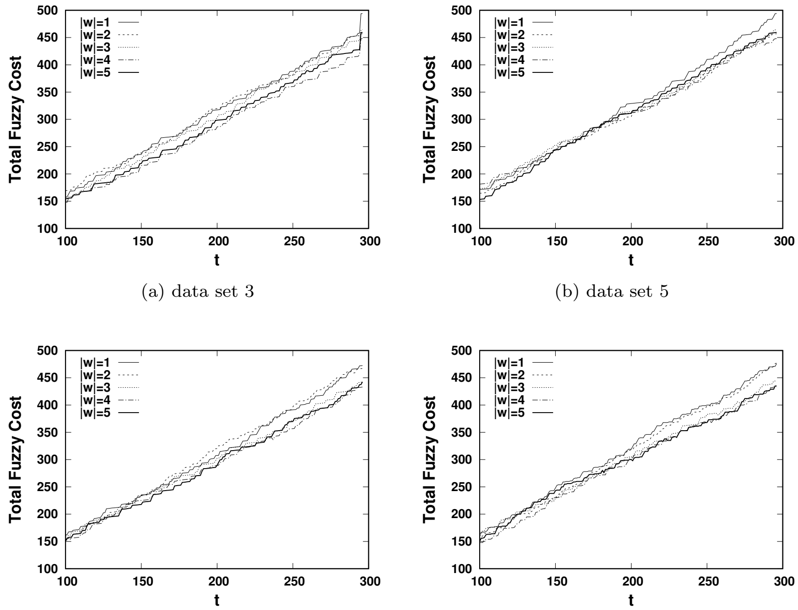
(d) data set 10

Figure 7: Heuristic 1 cost evaluation (TFc) for sharing profile varying window size $|w|$. Time frames 100–300 have been plotted to improve figure readability.

optimal deployments are reduced especially for bigger windows (greater than $|w|$=3) and *ii)* the variance of the total fuzzy cost observed with $|w|$=3, $|w|$=4, and $|w|$=5 using heuristic 2 is lower than the one observed using heuristic 1, meaning that heuristic 2 reduces the gap between different window sizes. This effect is also visible in Table 1 where the average improvement between $|w|$=4 and $|w|$=5 is lower compared to the others.

7.3.2. Heuristic 1 vs. Heuristic 2

We compared heuristic 1 and heuristic 2 using the relative cost increment (*metric 2* – Γ_t), the total fuzzy cost TFc with sharing and fitting profiles, and the entire data sets of 300 requests. Table 2 shows the average relative cost difference ($\overline{\Gamma_t}$) between heuristic 2 and heuristic 1, expressed in percentage, over our 10 data sets, varying $|w|$. Negative values indicate a cost decrease in heuristic 2 with respect to heuristic 1. We note that, even with a minimal



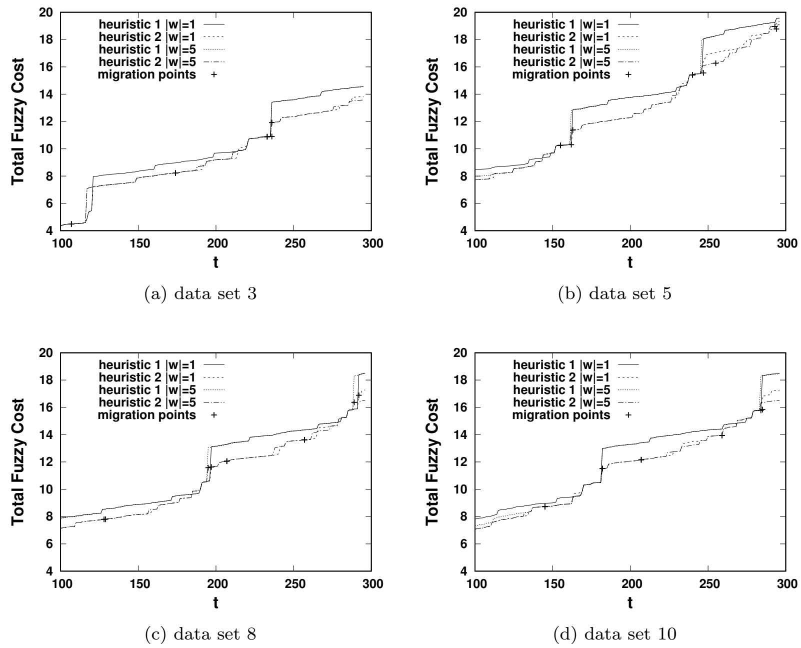
(a) data set 3
(b) data set 5
(c) data set 8
(d) data set 10

Figure 8: Heuristic 2 cost (TFc) evaluation for sharing profile varying window size $|w|$. Time frames 100–300 have been plotted to improve figure readability.

$\overline{\Gamma_t}$	$\|w\|$=**1**	$\|w\|$=**2**	$\|w\|$=**3**	$\|w\|$=**4**	$\|w\|$=**5**
sharing	−0.54%	−1.20%	−1.16%	−1.89%	−0.46%
fitting	−3.96%	−3.81%	−3.88%	−3.08%	−2.78%

Table 2: Average relative cost difference ($\overline{\Gamma_t}$ in %) at t=300 between heuristic 2 and heuristic 1, using sharing and fitting profiles, and varying window size $|w|$. Negative values indicate a cost decrease in heuristic 2 with respect to heuristic 1.

bias effect introduced by different windows sizes, heuristic 2 outperforms, on average, heuristic 1 regardless the used profile. More specifically, considering the sharing profile, our results show that the average relative cost difference between the two heuristics is around −1%. Considering fitting profile, the average relative cost difference is more than three times higher for all window sizes and is around −3.5%. For this reason, in the following, we further discuss the effect of heuristic 2 on fitting profile only.



(a) data set 3 (b) data set 5

(c) data set 8 (d) data set 10

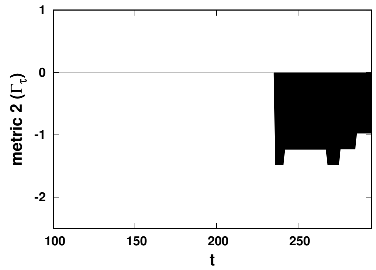
Figure 9: Comparison between heuristics 1 and 2 with fitting profile (TFc), using $|w|$=1 and $|w|$=5. Time frames 100–300 have been plotted to improve figure readability. Migration events are marked with "+".

Figure 9 shows the comparison of the total fuzzy costs TFc of heuristic 1 and heuristic 2, using 4 representative data sets and window size $|w|$=1 and $|w|$=5. We note that, when a migration is triggered in heuristic 2 (denoted with "+" in the figure), a substantial decrease in TFc is observed (e.g., at t=161 and t=244 for data set 5 in Figure 9(b)). We also note that, when $|w|$=1, the migrations are synchronous with the cost increment, while, when $|w|$=5 and in general with $|w|$>1, the migrations are triggered before the cost increment due to the look-forward effect of window w. We finally note that the total fuzzy costs TFc in Figure 9 presents a "step behavior" that mimics the one of cost factors α and β and confirms the findings in Table 1 about the negligible impact of window size on both heuristic 1 and heuristic 2 for fitting profile.

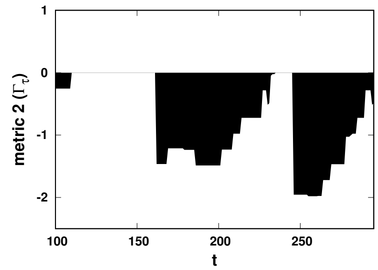
To further evaluate the contribution of migrations in heuristic 2, Figure 10 presents the distribution of the difference between Γ_t of heuristic 2 and Γ_t of heuristic 1, where a negative value indicates a cost saving when heuristic 2 is used, with $|w|$=5 for 4 representative data sets and fitting profile. Here, a recurrent pattern "a negative peak followed by uphill steps" can be easily identified in all data sets and reflects the negative peak (cost decrease) introduced by a migration followed by a step-like cost degradation showing a convergence between the two heuristics after migration. For instance, in case of data set 5, after migration at t=162, the cost shows a negative peak followed by a cost degradation (cost increase), until the following migration at t=189. We note that degradation can lead to a cost increase (e.g., positive difference for data set 8 and data set 10), where heuristic 1 is less costly than heuristic 2. This effect is however limited to a very short period of time and the cost difference is very small. Considering all 10 data sets and $|w|$=5, heuristic 1 outperforms heuristic 2 only 7 times for data set 8 and 6 times for data sets 1, 9, 10, showing a maximum cost difference of 0.23%. Quantitatively (*metric 3* $-\ \overline{\Delta_t}$), heuristic 2 produced 8 migrations on average on the 10 data sets for sharing profile and window $|w|$=5, 5 migrations for fitting profile and window $|w|$=5.

We note that, the comparison in this section, being based on total fuzzy costs, is affected by the "normalization" introduced by defuzzification, and reflects a *perceived cost* more than a *concrete cost*. To provide a more tangible quantitative analysis of the cost decrease introduced by heuristic 2, Figure 11 presents a comparison in terms of α and β for sharing and fitting profiles over the 10 data sets. In particular, *i)* Figure 11(a) compares heuristic 1 (fitting profile, $|w|$=1) with heuristic 2 (fitting profile, $|w|$=5), *ii)* Figure 11(b) compares heuristic 1 (sharing profile, $|w|$=1) with heuristic 2 (sharing profile, w=5). Heuristic 2 provides an average cost reduction over heuristic 1 on the real cost $\alpha + \beta$ of -18.5% for sharing profile and -8.2% for fitting profile.

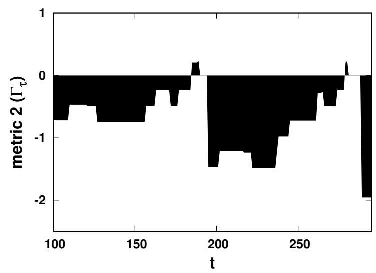
To conclude, while proving the effectiveness of both heuristics, our experiments show the utility of certification portability supporting migrations (replacement), as well as new instantiations of the same services (versioning). We remark that, while migrations generate intrinsic costs [27], they are in most of the cases "operational costs" due to computation or bandwidth degradation [28].



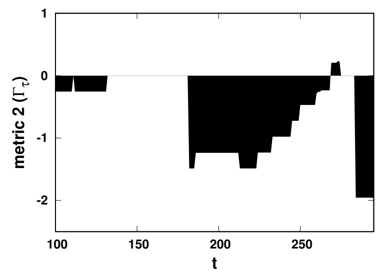
(1) data set 3



(2) data set 5



(3) data set 8



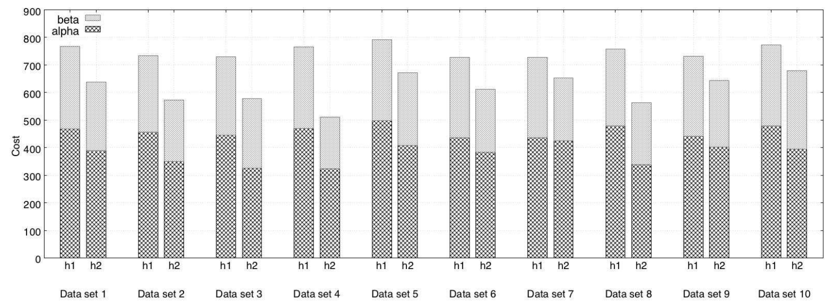
(4) data set 10

Figure 10: Filled curve plot of the distribution of the difference between Γ_t of heuristic 2 and Γ_t of heuristic 1. Time frames 100–300 have been plotted to improve figure readability.

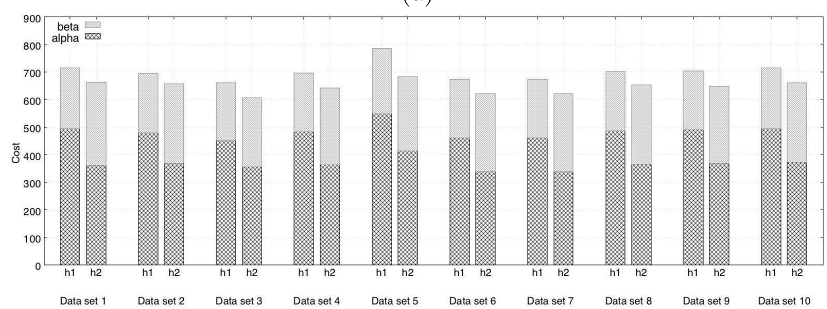
These costs can be normally mitigated by a number of well-established techniques (e.g., cold start servers, microservice architectures), as well as elasticity-based solutions [29]. In case of no mitigations, our approach can deal with migration events as additional CP costs to be added within the general cost functions. However, this scenario is not in the scope of this experimental evaluation and will be considered in our future work.

8. Related Work

Research on cloud service composition has recently focused on the problem of selecting component services on the basis of non-functional (including security) requirements [4, 7, 8, 9, 30, 31, 32]. Wang et al. [9] propose a network-aware service composition, which builds on candidate services geolocation to keep stable network performance. Qi et al. [7] propose a QoS-aware composi-



(a)



(b)

Figure 11: Comparing α, β total cost for the 10 data sets for sharing (a) and fitting (b) profiles using heuristic 1 window $|w|$=1 and heuristic 2 $|w|$=5, respectively.

tion method supporting cross-platform service invocation in cloud environment, using the execution time of single components. Manshan et al. [31] propose a fuzzy way solution to represent and solve QoS-based web services composition. Wu et al. [30] discuss a composition method providing a trustworthy selection of component services and guaranteeing trust in the composition. Kurdi et al. [8] focus on multiple cloud composition providing a combinatorial optimization algorithm for cloud service composition, aimed to maximize the fulfilment of clients' requests with minimal overhead. Arman et al. [32] propose a solution for moving application to the cloud, aiming to select the cloud service that matches at best application requirements and plan characteristics. Another line of research relevant for the work in this paper evaluated the costs of service composition algorithms [12, 13, 14, 33, 34]. Greenberg et al. [23] analyzed the most relevant direct and indirect costs in cloud service data centers, identifying the following: *i)* server costs (45% of the total costs) depend on server utilization and optimization, *ii)* infrastructure costs (25% of the total costs) comprise all facilities for power delivery, air conditioning, ups, and the like, *iii)* power draw (15% of the total costs) consists of all power consumption, and *iv)* network costs (15% of the total costs) comprises costs for switches, routers, agreements and traffic with ISP. Patel et al. [35] extend the above costs with the cost of personnel per rack and license costs, and identify model and functions representing the whole data center costs. We added on top of these works the costs of managing service certification and continuous service verification, as well as the costs due to *service versioning* and *service replacement*. These costs are due to the need of guaranteeing business continuity or, in other words, the cost of keeping the replica of components up and running while migrating a service. Jiang at al. [36], identified CPs physical resources utilization as an emerging problem and proposed a cloud capacity planning based on an ensemble time-series prediction method. He et al. [13] propose three novel QoS-aware service selection approaches for composing multi-tenant service-based systems. Li et al. [34] compare costs and service behaviours from different CPs, while Medeiros et al. [12] provide different cost patterns which may fit different types of services and service composition. Singh et al. [37] propose an agent-based and autonomous framework able to optimize the resource provisioning cost; the approach only focuses on virtual machine composition.

The solution in this paper extends the the cost-based certification approach in [15]. It provides a certification-based service composition for the cloud, which continuously evaluates non-functional properties using declarative and procedural modeling. It also investigates the composition costs from a cloud provider point of view [11, 14, 33], providing a cost-effective approach, using a fuzzy-based approach, for composite service deployment. The cost evaluation considers the costs introduced by the certification infrastructure and mismatch costs of maintaining services providing more than what is strictly needed for ensuring clients non-functional requirements.

9. Conclusions

We proposed an approach to cost-effective deployment of certified cloud composite services. The proposed solution provides a composition approach that comparatively evaluates service certificates to build a composite service addressing clients' requirements. It also provides a fuzzy-based cost evaluation methodology aimed to minimize the CP total costs of composition, identifying the best composition among possible alternative deployments. Our cost minimization approach has been implemented using two heuristics algorithms and assuming certified properties as *must-have* requirements. It has then been extensively experimentally evaluated simulating composition requests and comparing average composition costs of the two heuristics varying specific settings like window size and profile. This paper leaves some space for future work. First, our approach to cost optimization will consider scenarios distinguishing between non-functional requirements annotated by the users as soft (should-have) or hard (must-have), where cost improvements can be achieved by relaxing soft properties. Second, possible applications of the migration concept will be investigated to the aim of improving sub-optimal deployments in the cloud.

Acknowledgments

This work was partly supported by the program "piano sostegno alla ricerca 2015-17" funded by Università degli Studi di Milano, and the program FFABR "Fondo per il Finanziamento delle Attività Base di Ricerca", funded by the Italian MIUR.

References

[1] S.-Y. Hwang, E.-P. Lim, C.-H. Lee, C.-H. Chen, Dynamic web service selection for reliable web service composition, IEEE TSC 1 (2) (2008) 104–116.

[2] H. Wang, P. Ma, Q. Yu, D. Yang, J. Li, H. Fei, Combining quantitative constraints with qualitative preferences for effective non-functional properties-aware service composition, JPDC 100 (2017) 71 – 84.

[3] D. Ye, Q. He, Y. Wang, Y. Yang, An agent-based service adaptation approach in distributed multi-tenant service-based systems, JPDC 122 (2018) 11 – 25.

[4] M. Anisetti, C. Ardagna, E. Damiani, Security certification of composite services: A test-based approach, in: Proc. of ICWS 2013, San Francisco, CA, USA, 2013.

[5] F. Tao, D. Zhao, Y. Hu, Z. Zhou, Resource service composition and its optimal-selection based on particle swarm optimization in manufacturing grid system, IEEE TII 4 (4) (2008) 315–327.

[6] T. Xiang, X. Li, F. Chen, Y. Yang, S. Zhang, Achieving verifiable, dynamic and efficient auditing for outsourced database in cloud, JPDC 112 (2018) 97 – 107.

[7] L. Qi, W. Dou, X. Zhang, J. Chen, A qos-aware composition method supporting cross-platform service invocation in cloud environment, J. Comput. Syst. Sci. 78 (5) (2012) 1316–1329.

[8] H. Kurdi, A. Al-Anazi, C. Campbell, A. Al Faries, A combinatorial optimization algorithm for multiple cloud service composition, Comput. Electr. Eng. 42 (C) (2015) 107–113.

[9] X. Wang, J. Zhu, Y. Shen, Network-aware qos prediction for service composition using geolocation, IEEE TSC 8 (4) (2015) 630–643.

[10] K. Kofler, I. ul Haq, E. Schikuta, User-centric, heuristic optimization of service composition in clouds, in: Proc. of Euro-Par 2010, Ischia, Italy, 2010.

[11] A. Jula, E. Sundararajan, Z. Othman, Cloud computing service composition: A systematic literature review, Expert Systems with Applications 41 (8) (2014) 3809–3824.

[12] R. Medeiros, N. S. Rosa, L. F. Pires, Predicting service composition costs with complex cost behavior, in: Proc. of IEEE SCC, New York, NY, USA, 2015.

[13] Q. He, J. Han, F. Chen, Y. Wang, R. Vasa, Y. Yang, H. Jin, Qos-aware service selection for customisable multi-tenant service-based systems: Maturity and approaches, in: Proc. of IEEE CLOUD, New York, NY, USA, 2015.

[14] P. Leitner, W. Hummer, S. Dustdar, Cost-based optimization of service compositions, IEEE TSC 6 (2) (2013) 239–251.

[15] M. Anisetti, C. Ardagna, E. Damiani, F. Gaudenzi, A cost-effective certification-based service composition for the cloud, in: Proc of IEEE SCC 2016, San Francisco, CA, USA, 2016.

[16] M. Anisetti, C. A. Ardagna, E. Damiani, J. Maggesi, Security certification-aware service discovery and selection, in: Proc. of IEEE SOCA, 2012, pp. 1–8.

[17] M. Anisetti, C. Ardagna, E. Damiani, G. Polegri, Test-based security certification of composite services, ACM TWEB 13 (1) (2018) 3.

[18] M. Anisetti, C. Ardagna, E. Damiani, F. Gaudenzi, A certification framework for cloud-based services, in: Proc. of ACM SAC 2016, Pisa, Italy, 2016.

[19] C. Consortium, D2.1: Security-aware SLA specification language and cloud security dependency model, `https://cordis.europa.eu/docs/projects/cnect/0/318580/080/deliverables/001-D21SecurityawareSLAspecificationlanguageandcloudsecuritydependencymodelv101.pdf`, Accessed in Date June 2019.

[20] M. Anisetti, C. Ardagna, E. Damiani, F. Gaudenzi, A semi-automatic and trustworthy scheme for continuous cloud service certification, IEEE TSC (2017).

[21] M. Anisetti, C. Ardagna, E. Damiani, A certification-based trust model for autonomic cloud computing systems, in: Proc. of IEEE ICCAC, 2014, pp. 212–219.

[22] C. T. Horngren, G. Foster, S. M. Datar, M. Rajan, C. Ittner, A. A. Baldwin, Cost accounting: A managerial emphasis, Issues in Accounting Education 25 (4) (2010) 789–790.

[23] A. Greenberg, J. Hamilton, D. A. Maltz, P. Patel, The cost of a cloud: Research problems in data center networks, SIGCOMM Comput. Commun. Rev. 39 (1) (2008) 68–73.

[24] S. Newman, Building Microservices, O'Reilly Media, Inc., 2015.

[25] C. Sadtler, Z. X. Chen, S. Imazeki, M. Kelm, S. Kofkin-Hansen, Z. Q. Kou, B. McChesney, et al., IBM Workload Deployer: Pattern-based Application and Middleware Deployments in a Private Cloud, IBM Redbooks, 2012.

[26] I. Maleki, L. Ebrahimi, S. Jodati, I. Ramesh, Analysis of software cost estimation using fuzzy logic, IJFCST 4 (3) (2014) 27–41.

[27] S. Kikuchi, Y. Matsumoto, Performance modeling of concurrent live migration operations in cloud computing systems using prism probabilistic model checker, in: Proc of IEEE CLOUD, 2011, pp. 49–56.

[28] D. Breitgand, G. Kutiel, D. Raz, Cost-aware live migration of services in the cloud., in: SYSTOR, 2010.

[29] U. Sharma, P. Shenoy, S. Sahu, A. Shaikh, A cost-aware elasticity provisioning system for the cloud, in: Proc. of ICDCS, IEEE, 2011, pp. 559–570.

[30] X. Wu, B. Li, R. Song, C. Liu, S. Qi, Trust-based service composition and optimization, in: Proc of APSEC 2012, Hong Kong, 2012.

[31] M. Lin, J. Xie, H. Guo, H. Wang, Solving qos-driven web service dynamic composition as fuzzy constraint satisfaction, in: Proc. of IEEE EEE, 2005, pp. 9–14.

[32] A. Arman, S. Foresti, G. Livraga, P. Samarati, A consensus-based approach for selecting cloud plans, in: Proc. of IEEE RTSI, 2016, pp. 1–6.

[33] D. Worm, M. Zivkovic, H. van den Berg, R. van der Mei, Revenue maximization with quality assurance for composite web services, in: Proc. of IEEE SOCA, Taipei, Taiwan, 2012.

[34] A. Li, X. Yang, S. Kandula, M. Zhang, Cloudcmp: Comparing public cloud providers, in: Proc. of IMC 2010, Melbourne, Australia, 2010.

[35] C. D. Patel, J. S. Amip, Cost model for planning, development and operation of a data center,`http://hpl.hpl.hp.com/techreports/2005/HPL-2005-107R1.pd`.

[36] Y. Jiang, C.-s. Perng, T. Li, R. Chang, Self-adaptive cloud capacity planning, in: Proc. of IEEE SOCA, 2012, pp. 73–80.

[37] A. Singh, D. Juneja, M. Malhotra, A novel agent based autonomous and service composition framework for cost optimization of resource provisioning in cloud computing, Journal of King Saud University - Computer and Information Sciences 29 (1) (2017) 19 – 28.