deQAM: A Dependency Based Indirect Monitoring Approach for Cloud Services

Heng Zhang, Jesus Luna, and Neeraj Suri
Dept. of Computer Science
TU Darmstadt, Germany
Email:{zhang, jluna, suri}@deeds.informatik.tu-darmstadt.de

Ruben Trapero
Atos Research & Innovation
Calle de Albarracin 25, Madrid, Spain
Email: ruben.trapero.external@atos.net

***Abstract*—Cloud service monitoring is a critical need for both providers and customers to assess the state of resources and the level of delivery of services. However, existing Cloud service monitoring methods are typically inapplicable in case the targeted service parameters are inaccessible, e.g., Cloud Service Providers do not allow external access to some service parameters for varied reasons (proprietary IPR, privacy issues, security concerns or simply access difficulties). Hence, alternate Cloud service monitoring approaches are needed to address this issue. Nonetheless, Cloud services are not provided in isolation and very often share common resources with other services that naturally introduces service dependencies between different services. In this paper, we propose a novel Cloud service monitoring approach which targets such dependencies to conduct indirect monitoring of inaccessible Cloud service parameters by using monitoring information collected from other Cloud services. The proposed monitoring approach can also assess the reliability of the monitored result. The presented case study validates the applicability of the proposed indirect monitoring approach.**

***Keywords*-Service Security; Service Security Assurance; Service Monitoring; Cloud Services;**

I. Introduction

Cloud service monitoring is an important quality assurance mechanism for the actors involved in the provisioning of Cloud services. For example, Cloud service monitoring mechanisms are used by Cloud Service Providers (CSPs) to manage Cloud services such as getting up-to-date service information, planning maintenance tasks or checking the fulfillment of the service agreements also for Cloud Customers. Cloud service monitoring mechanisms are also used by third parties, such as by certification auditors which rely on the monitoring information to issue certificates.

Although Cloud service monitoring is crucial for the actors involved in the Cloud, existing Cloud service monitoring mechanisms are effective only if the related parameter information is directly accessible on the provider's side [2][3][10][11][14]. However, many providers do not allow external parties to access specific service parameters for many reasons, such as privacy reasons, security concerns or lack of proper access interfaces. In these cases, the existing monitoring mechanisms become inapplicable. As a result, it is necessary to design a new Cloud monitoring mechanism to address this gap.

Given that Cloud services often share common resources and work on top of the same platform or infrastructure[1], it is inevitable that dependencies exist between some Cloud services. As a result, these service dependencies can be used for developing the new Cloud service monitoring mechanism which can monitor those inaccessible parameters of the Cloud service by using the parameter information collected from other Cloud services.

For example, a Cloud secure storage service does not allow external parties to directly monitor parameters related to the *data security level* which is an important aspect of a SecSLA[2]. To help Cloud service customers validate the SecSLA compliance, third parties such as security auditors can use service dependencies to indirectly monitor these parameters. In the case of a Cloud secure storage service we can suppose that it works as the combination of three different services, namely an authentication service (S_{Auth}), an encryption service (S_{Encypt}), and a storage service (S_{Store}). Therefore, the security auditor can check parameter information from S_{Auth} to see whether the access is from a legitimate user or it can also access to information from S_{Encypt} to see which type of encryption applied on the user data. However, two major challenges need to be solved for designing an indirect monitoring mechanism for Cloud services: (1) how to quantitatively use service dependencies for monitoring service parameters, and (2) how to evaluate the reliability of the result from the indirect monitoring.

We address these challenges by proposing a novel indirect monitoring scheme. The proposed approach takes advantage of service dependencies for which the value of service parameters can be quantitatively formulated with the provisioning level of Cloud services, and indirectly monitored by aggregating the relevant provisioning levels. To the best of our knowledge, our work is the first attempt to address the indirect monitoring issue in Cloud. The contributions are:

1) An indirect Cloud service monitoring approach termed **de**pendency-based **Q**uantitative **A**ggregation **M**ethodology ($deQAM$) which parameterizes service

[1]An example is the Cloud storage service Dropbox is working on top of AWS S3. cf. https://aws.amazon.com/customerapps/1955

[2]SecSLA is a legal agreement negotiated and contracted between the Cloud Service Provider (CSP) and the Cloud Service Customer (CSC) to guarantee Cloud services delivered with the specific security assurance.

dependencies to monitor the target parameter's value.
2) A reliability assessment approach which evaluates the confidence level of the monitoring result.
3) A case study to validate the applicability of the proposed monitoring approach.

The rest of this paper is organized as follows. Section II presents some basic considerations for designing the indirect monitoring approach. Section III elaborates the proposed monitoring approach. Section IV shows the applicability of the approach on a case study for $deQAM$. Section V discusses the case study results. Section VI presents the related work, and Section VII concludes the paper.

II. Basic Concepts

This section reviews several basic issues related to designing an indirect Cloud service monitoring approach.

A. Cloud Service Provisioning & Indirect Monitoring

The provision of Cloud services typically involves multiple supporting services and also multiple service providers where many services or service providers also share common resources or infrastructure. Taking a Cloud secure storage service for example, it offers the service of encrypting and storing client's data in Cloud using a service hierarchy such as the one depicted in Fig. 1.

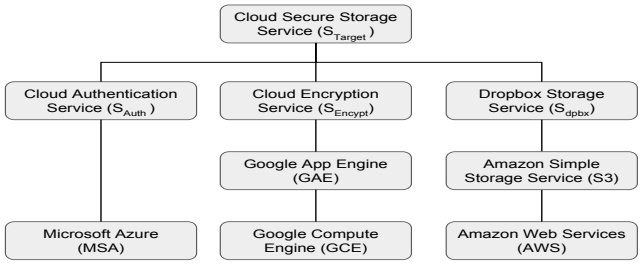

Figure 1. A Cloud secure storage service's service provisioning hierarchy

With the above service provisioning hierarchy, some parameters of the Cloud secure storage service could be indirectly monitored by third parties without requiring direct access to these parameters. For example, the *data security level* is an important security parameter for Cloud service customers (CSCs) using this Cloud secure storage service (S_{Target}). However, this security parameter is impossible to be directly monitored due to privacy protection concerns. Other parameters can be monitored by third parties from other Cloud services residing in the same hierarchy. The possible examples could be the supported *encryption algorithm list* in the Cloud encryption service (S_{Encypt}), *authentication notification* messages in the Cloud authentication service (S_{Auth}) or the *bug report of the security module* in Microsoft Azure (MSA). For example, by studying the related parameters collected from other Cloud services in Fig. 1, it is possible for third parties to indirectly monitor the parameter *data security level*.

B. Service Dependency & Characterization

Service dependency is the directed relation between different services, which refers to one service (a.k.a dependent service) subjected to the persistent constrains from another or multiple services (a.k.a. antecedent service) [19]. Service dependency specifies the provisioning of services that depends on its antecedent services, which might affect the provisioning of the dependent service (e.g., degradations or disruptions or failures) [23]. For the aforementioned example, the provisioning of the Cloud secure storage service (dependent service) depends on its antecedent services, namely the Cloud authentication service and the Cloud encryption service in order to meet the security requirements of the storage service. Therefore, the service dependency has deep influence on the service provisioning of Cloud services.

From the monitoring perspective, the following characteristics of service dependency are worth highlighting:

Direction: The dependency direction states the directional information that identifies the dependent and the antecedent in the service dependency. It can assist in trimming the service dependencies by removing the irrelevant services.

Type: The dependency type reflects antecedent services imposing different types of influence on dependent services. The influence is either the decisive type (enable/disable) or the indecisive type (altering to some extent), affecting the dependent service provisioning.

Strength: The dependency strength represents the degree of dependent services relying on their antecedent services. The higher dependency strength implies the closer service provisioning of dependent services relying on their antecedent services.

C. Uncertainty & Data Estimation

Uncertainty is a key issue in designing an indirect monitoring approach. Uncertainty, which is caused by the incompleteness of knowledge about the monitoring target, deviates the value of the monitoring result from the real value [24].

However, the indirect monitoring approach takes the parameter information from antecedent services without the complete knowledge of the target parameter in dependent services. Therefore, it is nontrivial to address the uncertainty problem for the indirect monitoring approach. In practice, a common solution to address the uncertainty issue is to determine a confidence level (typically 95%) for evaluating the reliability of the monitoring result.

III. The Proposed $deQAM$ Methodology

This section presents the details of the dependency-based Quantitative Aggregation Methodology ($deQAM$) as the proposed indirect Cloud monitoring approach.

A. System Overview

The dependency-based quantitative aggregation methodology ($deQAM$) is the indirect monitoring approach proposed

to monitor the parameters of Cloud services. $deQAM$ uses the related parameter information collected from antecedent services to monitor the dependent service parameter which is difficult to be monitored directly. $deQAM$ adopts a multi-step process to achieve the indirect monitoring as shown in the bottom boxed part of Fig. 2. Briefly, the "utility mapping" step converts the parameter value to a utility level as explained in Section III-B. The "dependency parameterization" step generates the trimmed parameterized service graph with service dependencies. The "monitoring result inference" step computes the utility of the target parameter. The "monitoring result generation" step generates the indirect monitoring result by mapping the target parameter utility back to the parameter value. The "result reliability evaluation" step addresses the reliability issue by estimating the monitoring result's reliability at a 95% confidence level. These steps are detailed in Section III-C.

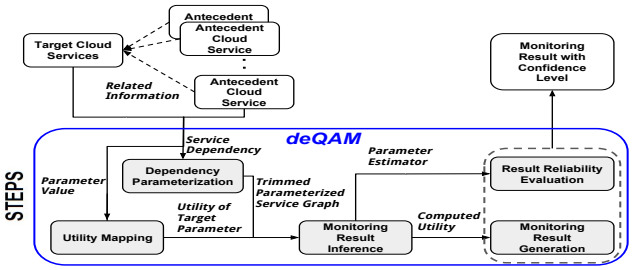

Figure 2. The structure of indirect monitoring methodology ($deQAM$)

B. Transformation of Dependency

$deQAM$ treats the service dependency as quantitative constraints on the Cloud service provisioning. It uses *utility* as the quantitative unit for representing the influence that the service parameter value has on the service provisioning. $deQAM$ categorizes the utility into two classes.

Eigen utility: Eigen utility represents the basic provisioning level of the dependent service affected by the value of other parameters in the same service. If the eigen utility is exceptional (i.e. lower than some thresholds), a serious problem might have occurred with the service provisioning, such as service dysfunctional, interrupted or completely down.

Contributed utility: Contributed utility represents the dependent service's provisioning level affected by the value of parameters of antecedent services. According to the different types of influence, it is further categorized into two types, namely the *mandatory-type contributed utility* and the *optional-type contributed utility*. Specifically, the mandatory-type contributed utility represents the decisive influence on the provisioning of the dependent service which decides whether the dependent service is provisioned or not. The optional-type contributed utility represents the indecisive influence affected on the provisioning level of the dependent service which increases the service provisioning level based on the basic provisioning level.

As a result, the utility of the service parameter value is the sum of the eigen utility and the contributed utility.

C. Design of $deQAM$

$deQAM$ is a multi-step indirect Cloud service monitoring approach that quantitatively analyzes the utility as follows.

Step 1. Utility Mapping: $deQAM$ starts the monitoring process by mapping the parameter value V_p, collected from antecedent services, onto utility U_p. Based on the discussion in Section III-B, we propose two different conversion rules (depicted as f_B for *Boolean* and f_N for *Numerical*) for mapping the parameter value V_p onto utility U_p and the inverse conversion rules (depicted as f_B^{-1} and f_N^{-1}) for the back-mapping process.

$$f\_B : V_p \mapsto U_p = \begin{cases} 1 & V_p \in S_{en} \\ 0 & V_p \in S_{dis} \end{cases}$$

$$f\_B^{-1} : U_p \mapsto V_p = \begin{cases} V_p \in S_{en} & U_p = 1 \\ V_p \in S_{dis} & U_p = 0 \end{cases}$$

The parameter value V_p having the dominant influence on the dependent service's provisioning is mapped to the *boolean* value by using f_B, in which S_{en} is the set of the value for enabling the service provisioning (e.g., *yes, enable or activate*) and S_{dis} is the set of the value for disabling the service provisioning (e.g., *no, disable or inactive*). For example, the parameter value of *login failure message* is $V_1 = true/false$, its utility is mapped onto $U_1 = 0/1$ by using f_B when the service provisioning is *disabled/enabled*.

$$f\_N : V_p \mapsto U_p = \frac{V_p - V_p\_min}{V_p\_max - V_p\_min}$$

$$f\_N^{-1} : U_p \mapsto V_p = \frac{U_p - V_p\_min}{U_p\_max - U_p\_min}$$

The parameter value V_p having the influence of increasing the provisioning level of the dependent service is mapped to the *numerical* value in the range of $[0, 1]$ by using f_N, in which V_p_max and V_p_min are the upper and the lower boundary of V_p's varying range. For example, the parameter value of *CPU usage rate* varies in the range of $[0, 100\%]$. If the collected *CPU usage rate* is $V_2 = 62\%$, then its utility U_2 is mapped onto 0.62 by using f_N. In a similar way, the parameter value of *client storage quota* is an element of the set $\{250GB, 500GB, 750GB, 1000GB\}$ and can be regarded as varying in the range of $[0, 1]$ which is divided into four parts. Then the utility of corresponding elements are mapped onto $0.25, 0.5, 0.75, 1$ by using f_N.

***Step 2. Service Dependency Parameterization**: $deQAM$* characterizes service dependencies with the parameterized service dependency graph which consists of a set of vertexes representing Cloud services and a set of edges representing parameterized dependencies between different Cloud services. The parameterized service dependency graph is defined as:

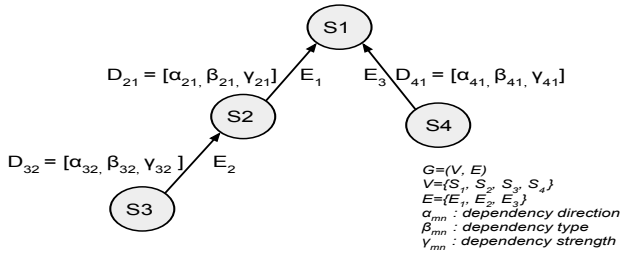

Figure 3. An example of the parameterized service dependency graph

Definition. A parameterized service dependency graph is a multi-parameter directed graph denoted with $G = (V, E)$, in which:

• V are the vertexes of Graph G which represents a finite set of Cloud services denoted by $V = \{S_1, S_2, ..., S_i\}$. S_i means the $i\text{-}th$ service.

• E are the directed edges of Graph G which represents a finite set of dependencies between different services denoted by $E = \{E_1, E_2, ..., E_j\}$, E_j means the $j\text{-}th$ service dependency between two Cloud services. E_j is characterized with a three-parameter vector $\vec{D_{mn}} = [\alpha_{mn}, \beta_{mn}, \gamma_{mn}]$ to parameterize service dependencies obtained from the service provisioning hierarchy as illustrated in Section II-B.

α_{mn}**:** It is the parameter specifying the target-oriented dependency direction between two different Cloud services. Specifically, if the antecedent service is S_m and the dependent service is S_n ($S_m, S_n \in V$) and both services are directly or indirectly affect a given target service, its value is 1. Otherwise, if S_m and S_n does not affect the given target service, the value of dependency direction is -1.

β_{mn}**:** It is the parameter representing the different types of influence of the S_m's parameter value on the S_n's service provisioning. Its value is assigned to 0 to indicate the decisive influence dependency, while its value is assigned to 1 to indicate the indecisive influence dependency.

γ_{mn}**:** It is the parameter defining the rate that the utility mapped from S_m's parameter value contributes onto S_n's service provisioning level caused by the service dependency. It is a real number in the range of $[0, 1]$.

Based on the parameterized service dependency graph G, $deQAM$ adopts a target-centric service dependency trimming method which examines dependencies with regard to the target service (S_{Target}) and recursively checks the corresponding antecedent services to remove irrelevant service dependencies (i.e. service dependencies do not affect the monitoring result) as specified in Algorithm 1. The trimming algorithm takes advantage of a set of services (denoted by S_x) and a set of corresponding parameterized dependency vectors (denoted by E_y) derived from these services regarding to the target service. By recursively comparing the dependency direction α_{xt} of every service in S_x, the trimming algorithm can finally generate the minimized service dependency graph (denoted by $G^{'}$) with regard to the target service. In addition, the trimming algorithm traverses all nodes (services) of the graph to do the dependency direction (α_{xt}) comparison. Therefore, the complexity for the trimming algorithm is $\mathcal{O}(n)$, where n is the node number.

Algorithm 1 Target-Centric Service Dependency Trimming Algorithm

Require: parameterized service dependency graph $G = (V, E)$ of i vertexes and j edges.
Ensure: trimmed parameterized service dependency graph $G^{'} = (V^{'}, E^{'})$.

```
set S_x ∈ V = {S_1, S_2, ..., S_i}, E_y ∈ E = {E_1, E_2, ..., E_j};
set V' = {S_Target} and E';
while before the element number of V' no longer changes do
    for each element S' ∈ V' do
        set S_t = S';
        for each element S_x ∈ V ⋂ S_x ≠ S_t do
            if E_y ∈ E ⋂ α_xt == 1 then
                add S_x and E_y's dependency D_xt into V' and E' respectively;
            end if
        end for
    end for
end while
return Trimmed parameterized service dependency graph G' = (V', E')
```

***Step 3. Monitoring Result Inference**:* Based on the trimmed parameterized service dependency graph G', utility U_φ of the inaccessible parameter φ of the target service S_{Target} is inferred by using parameterized dependencies. The utility computation process consists of three different parts as follows:

Eigen utility: The eigen utility U_{E_φ} of the inaccessible parameter φ of S_{Target} is decided by the baseline of the service provisioning. It is the threshold of the service provisioning without considering any dependency from antecedent services.

Mandatory-type contributed utility: The mandatory-type contributed utility U_{M_φ} of the inaccessible parameter φ of S_{Target} is computed by using the dependencies which impose the dominant influence on the provisioning of

S_{Target}. Due to the dominant influence of the dependency type, the mandatory-type contributed utility U_{M_φ} is the product of all boolean utilities mapped from the antecedent services' parameter value with decisive influence on the service provisioning of S_{Target} (i.e. when $\beta_{mn} = 0$). It can be computed as follows:

$$U_{M\_\varphi} = \prod_{1}^{l} U_l \tag{1}$$

In Equation (1), l is the number of service dependencies with $\beta_{mn} = 0$ in $G^{'}$ and U_l is the utility mapped from the antecedent service S_l' parameter value V_l.

Optional-type contributed utility: The optional-type contributed utility U_{O_φ} of the inaccessible parameter φ of S_{Target} aggregates the utility computed by all optional-type service dependencies. However, it requires to compute the utility caused by the ripple-effect of service dependency which is the phenomenon of the influence (either quantitative or qualitative) propagation from antecedent services to dependent services [17]. Therefore, the optional-type utility computation needs to include both the direct and the ripple-effect optional-type contributed utility computation. As a result, $deQAM$ computes the optional-type contributed utility U_{O_φ} as:

• *Ripple-effect optional-type contributed utility:* The ripple-effect optional-type contributed utility $U_{mn}^{(o.r)}$ is the optional utility indirectly contributed by the parameter of antecedent service S_m to the parameter of dependent service S_n. Given a three-node dependency chain in the trimmed parameterized dependency graph $G^{'}$, it can be represented as $\{S_m \rightarrow S_k \rightarrow S_n\}$. This dependency chain consists of an antecedent service S_m, an intermediate service S_k and a dependent service S_n. The ripple-effect utility $U_{mn}^{(o.r)}$ propagating from S_m to S_n is computed as:

$$U_{mn}^{(o.r)} = (\gamma_{mk} \cdot U_m) \cdot \gamma_{kn} \tag{2}$$

In Equation (2), $\gamma_{mk} \cdot U_m$ is the optional utility mapped from S_m's parameter value V_m and contributed to S_k. Multiplying the coefficient γ_{kn}, it results in the final optional utility contributed to the inaccessible parameter's utility of S_n.

• *Direct optional-type contributed utility:* Based on the ripple-effect optional-type utility computation method, the direct optional-type contributed utility $U_{mn}^{(o.d)}$ is computed as the simplified form, in which the dependency chain contains only two nodes as $\{S_m \rightarrow S_n\}$. Accordingly, the direct optional-type contributed utility is computed as:

$$U_{mn}^{(o.d)} = \gamma_{mn} \cdot U_m \tag{3}$$

The output of Equation (3) is the optional utility mapped from the parameter value of S_m and contributed to the inaccessible parameter's utility of S_n.

Based on Equation (2) and (3), the optional-type contributed utility U_{O_φ} of the inaccessible parameter φ of S_{Target} is aggregated as:

$$U_{O\_\varphi} = \sum_{1}^{p} \sum_{1}^{q} U_{mn}^{(o.r)} + \sum_{1}^{r} U_{mn}^{(o.d)} \tag{4}$$

In Equation (4), p is the number of dependency chains related to S_{Target}, q is the number of all non-adjacent antecedent services of S_{Target} in each dependency chain, and r is the number of direct antecedent services of S_{Target}.

Inaccessible service parameter utility inference: As a result, utility U_φ of the inaccessible parameter φ of S_{Target} is computed with the results from the three different parts as:

$$U_\varphi = U_{M\_\varphi} \cdot (U_{E\_\varphi} + U_{O\_\varphi}) \tag{5}$$

In Equation (5), the mandatory-type contributed utility U_{M_φ} is the dominance over the utility U_φ of the inaccessible parameter φ of S_{Target} like the switch.

Step 4. Monitoring Result Generation: The monitoring result generation step maps the derived utility U_φ back to the inaccessible parameter's value V_φ. To do so, we apply the inverse conversion rules $f_{\_B}^{-1}$ and $f_{\_N}^{-1}$ in **Step 1** for the back-mapping process. However, the special attention is required to manage the case of mapping the parameter value V_φ back to a given set, because the computed utility may not exactly match any element in the set. Therefore, we propose to assign the derived utility U_φ to the nearest utility mapped from some element value in the set. For example, if the derived utility U_φ is 0.32 which does not match any utility in the set like $\{0.25, 0.5, 0.75, 1\}$. However, its two adjacent elements are 0.25 and 0.5. As a result, utility U_φ is replaced by 0.25 which has the minimum deviation to 0.32 than others.

Step 5. Result Reliability Evaluation: Because the monitoring result is inferred by $deQAM$ indirectly, it suffers from uncertainty which has to be estimated. Therefore, it is necessary to address the uncertainty issue by evaluating the reliability of the generated monitoring result V_φ of the inaccessible parameter φ of S_{Target}.

The $deQAM$'s inference process relies on the service dependency parameters α_{mn}, β_{mn}, and γ_{mn} from **Step 2**. As the value of γ_{mn} is generally derived by studying the empirical experimental data or the knowledge of the experts [22], the incomplete knowledge of γ_{mn} is the source of uncertainty for the indirect monitoring result V_φ. Therefore, we estimate γ_{mn} based on a statistic evaluation approach to assess the confidence level of the generated monitoring result V_φ.

The evaluation approach is designed with the assumption that γ_{mn} is derived with the best knowledge of the experts or the best-effort study on empirical experimental data. In other words, the value of γ_{mn} applied in $deQAM$'s utility computing process is the most likely approximation of the real strength of the dependency. Consequently, uncertainty ξ_{mn} is defined as the deviation between the approximation

value and the true value of γ_{mn}. As a result, the distribution of uncertainty can reflect the reliability of γ_{mn}. Specifically, the variance of the uncertainty distribution σ_{mn} is inversely proportional to the reliability of γ_{mn}. Apart from the variance σ_{mn}, the mean of the uncertainty distribution μ_{mn} is set to 0 due to no intentional bias introduced for deriving γ_{mn} according to the assumption. Meanwhile, the Gaussian distribution is commonly used for modeling the uncertainty problem. Therefore, the distribution of γ_{mn}'s uncertainty follows the Gaussian distribution $N(0, \sigma_{mn})$. As the value of γ_{mn} is independently obtained, the uncertainty $\xi(V_\varphi)$ of the generated monitoring result V_φ is the aggregation of all γ_{mn}'s uncertainty. Thus, $\xi(V_\varphi)$ follows the Gaussian distribution as:

$$\xi(V_\varphi) \sim N(0, \sum \sigma_{mn}) \tag{6}$$

However the true value of σ_{mn} is generally unknown. Hence, we adopt the mean of the square products of the uncertainty differences to estimate the true value of $\hat{\sigma}^2_{\xi(V_\varphi)}$ obtained from Equation (6) as:

$$\hat{\sigma}^2_{\xi(V_\varphi)} = \frac{1}{t}\sum_{1}^{t}(\gamma_{mn} - \gamma_{mean})^2 \tag{7}$$

In Equation (7), γ_{mean} is the mean value of all γ_{mn} participated in the utility computation process. While t is the total number of γ_{mn} participated in the utility computation process. Accordingly, a χ^2-distribution can be obtained based on Equation (7) as:

$$\chi^2(t) \sim \frac{\sum_1^t(\gamma_{mn} - \gamma_{mean})^2}{\sigma^2_{\xi(V_\varphi)}} = \frac{t \cdot \hat{\sigma}^2_{\xi(V_\varphi)}}{\sigma^2_{\xi(V_\varphi)}} \tag{8}$$

Therefore, the χ^2-distribution statistic characteristic can be used to determine the confidence level $(1 - c)$ to assess the reliability of the generated monitoring result V_φ [6]. Based on the distribution (8), the confidence level $1 - c$ of $\sigma^2_{\xi(V_\varphi)}$ is derived to represent the generated monitoring result's reliability [13] as:

$$\left(\frac{\sum_1^t(\gamma_{mn} - \gamma_{mean})^2}{\chi^2_{(1-\frac{c}{2},t)}}, \frac{\sum_1^t(\gamma_{mn} - \gamma_{mean})^2}{\chi^2_{(\frac{c}{2},t)}}\right) \tag{9}$$

IV. Case Study

$deQAM$ is applied to an example Cloud service to infer the inaccessible service parameter as an initial effort to validate the proposed approach. To the best of our knowledge, it is the first study for conducting indirect monitoring on the inaccessible parameter of the Cloud service.

Fig. 4.(a) depicts the service provisioning hierarchy of a Cloud service with other services. Without loss of generality, we assign the parameters φ_{s_1} and φ_{s_2} of services $S1$ and $S2$ (respectively) holding the optional-type dependencies indirectly and directly to the parameter φ_{s_5} of the service $S5$, the parameter φ_{s_3} of the service $S3$ holding the mandatory-type dependencies to the parameter φ_{s_4} of the service $S4$ and

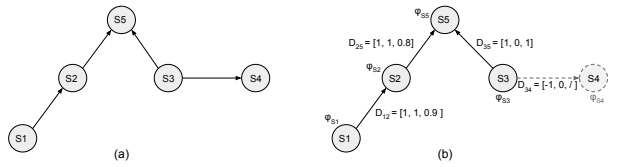

Figure 4. (a) The raw service provisioning hierarchy for Cloud service $S5$
(b) The parameterized service dependency graph for Cloud service $S5$

the parameter φ_{s_5} of the service $S5$. The target monitoring parameter φ_{s_5}, with the possible value varying range of $[0, 100\%]$, is unable to be monitored directly. However, the other information of the parameter from the other services can be collected as in Table I. Then, $deQAM$ conducts the indirect monitoring process as follows:

Table I
Case Study: Information collected for indirect parameter monitoring

Parameter	Dependent Cloud services				
Location	Service 1	Service 2	Service 3	Service 4	Service 5
Name	φ_{s_1}	φ_{s_2}	φ_{s_3}	φ_{s_4}	φ_{s_5}(Target)
Value Type	Numerical	Numerical	Boolean	Boolean	Numerical
Variation Range	[0,100%]	Lv 1/2/3/4/5	(in)activated	(un)available	[0,100%]
Collected Value	30%	Lv 1	activated	available	N.A
Trimming Status	Keep	Keep	Keep	Removed	Keep
DepDirection (α_{mn})	$\alpha_{12} = 1$	$\alpha_{25} = 1$	$\alpha_{35} = 1$	Removed	N.A
DepType (β_{mn})	$\beta_{12} = 1$	$\beta_{25} = 1$	$\beta_{35} = 0$	Removed	N.A
DepStrength (γ_{mn})	$\gamma_{12} = 0.9$	$\gamma_{25} = 0.8$	$\gamma_{35} = 1$	Removed	N.A

• ***Step 1. Utility Mapping*** To monitor parameter φ_{s_5}, $deQAM$ starts by mapping the value of the parameters collected from the antecedent services onto utility. By using the proposed conversion rules f_B and f_N, the utility of the value of the collected parameters $(\varphi_{s_1}, \varphi_{s_2}, \varphi_{s_3})$ are converted as:
$U_{\varphi_{s_1}} = \frac{30\%}{100\%-0} = 0.3$
$U_{\varphi_{s_2}} = \frac{Lv1}{\{Lv1,Lv2,Lv3,Lv4,Lv5\}} = \frac{0.2}{1.0-0} = 0.2$
$U_{\varphi_{s_3}} = 1$, since φ_{s_3} is a Boolean and $V_{\varphi_{s_3}}$ is *activated*.
• ***Step 2. Service Dependency Parameterization*** $deQAM$ constructs the parameterized service dependency graph G for conducting the utility analysis of φ_{s_5}. As in Fig. 4.(b), G contains 5 vertexes denoting the 5 different Cloud services and 4 edges specifying the dependency with the three-parameter (α_{mn}, β_{mn} and γ_{mn}) vectors. Then, the target-centric dependency trimming algorithm is executed to remove the irrelevant services based on the inputs of G and α_{mn}. The output of the algorithm is the trimmed parameterized service dependency graph $G^{'}$ which removes $S4$ as denoted in dotted form. Since the service dependency $\vec{D_{34}}$ is irrelevant to the utility analysis for monitoring φ_{s_5} and its directional parameter α_{34} sets to -1 in the figure.
• ***Step 3. Monitoring Result Inference*** $deQAM$ computes the utility of the parameters collected from $S1$, $S2$ and $S3$.

$*$ *Eigen utility:* φ_{s_5}'s eigen utility $U_{E\_\varphi_{s_5}}$ is decided by the basic provisioning level of $S5$. In this case study, we set $U_{E\_\varphi_{s_5}}$ to 0.5.

$*$ *Mandatory-type contributed utility:* $deQAM$ computes the mandatory-type contributed utility $U_{M\_\varphi_{s_5}}$ by using the monitoring information collected from $S3$. As $U_{\varphi_{s_3}} = 1$, the mandatory-type contributed utility is computed as:

$U_{M\_\varphi_{s_5}} = \prod_1^1 U_{\varphi_{s_3}} = 1$

$*$ *Optional-type contributed utility:* $deQAM$ computes the optional-type contributed utility $U_{O\_\varphi_{s_5}}$ by using the monitoring information collected from $S1$ and $S2$. According to Equation (2) and (3), $S1$'s ripple-effect optional-type utility $U_{15}^{(o.r)}$ and $S2$'s direct optional-type contributed utility $U_{25}^{(o.d)}$ are computed respectively as:

$U_{15}^{(o.r)} = (\gamma_{12} \cdot U_{\varphi_{s_1}}) \cdot \gamma_{25} = 0.216$

$U_{25}^{(o.d)} = \gamma_{25} \cdot U_{\varphi_{s_2}} = 0.16$

By Equation (4), φ_{s_5}'s optional-type contributed utility $U_{O\_\varphi_{s_5}}$ is:

$U_{O\_\varphi_{s_5}} = U_{15}^{(o.r)} + U_{25}^{(o.d)} = 0.376.$

$*$ *Inaccessible service parameter utility inference:* Based on the above computation, utility $U_{\varphi_{s_5}}$ of the inaccessible service parameter φ_{s_5} is derived by Equation (5) as:

$U_{\varphi_{s_5}} = U_{M\_\varphi_{s_5}} \cdot (U_{E\_\varphi_{s_5}} + U_{O\_\varphi_{s_5}}) = 0.876.$

• ***Step 4. Monitoring Result Generation*** $deQAM$ maps utility $U_{\varphi_{s_5}}$ back to φ_{s_5}'s value to generate the monitoring result $V_{\varphi_{s_5}}$. By applying proposed inverse conversion rule $f_{\_N}^{-1}$, the monitoring result $V_{\varphi_{s_5}}$ of the inaccessible parameter φ_{s_5} is generated as follows. Because $U_{\varphi_{s_5}}$ is 0.876 and the varying range of $V_{\varphi_{s_5}}$ is $[0, 100\%]$, the monitoring result $V_{\varphi_{s_5}}$ is derived as:

$V_{\varphi_{s_5}} = \frac{U_{\varphi_{s_5}}}{U_{\varphi_{s_5}\_max} - U_{\varphi_{s_5}\_min}} \cdot 100\% = \frac{0.876}{1-0} \cdot 100\% = 87.6\%$

• ***Step 5. Result Reliability Evaluation*** $deQAM$ determines the confidence level of the generated monitoring result by using Equation (9) and the value of γ_{mn} specified in Table I. In this case study, we assess the reliability of the monitoring result $V\varphi_{s_5}$ with the confidence level $(1-c)$ of 95% (i.e. $1-c = 0.95$). By checking the χ^2 distribution table [13], we find $\chi^2_{(\frac{0.05}{2},3)} = 0.216$ and $\chi^2_{(1-\frac{0.05}{2},3)} = 9.348$. According to Equation (7) and (9) and Table I,

$\gamma_{12} = 0.9, \gamma_{25} = 0.8, \gamma_{35} = 1$, $\gamma_{mean} = (\gamma_{12} + \gamma_{25} + \gamma_{35})/3 = 0.9$.

$\hat{\sigma}^2_{\xi(V_{\varphi_{s_5}})} = \frac{1}{3}\sum_1^3(\gamma_{mn} - \gamma_{mean})^2 = \frac{0.1^2+0.1^2}{3} = 0.0067$

$\frac{\sum_1^3(\gamma_{mn}-\gamma_{mean})^2}{\chi^2_{(1-\frac{0.05}{2},3)}} = \frac{0.1^2+0.1^2}{0.226} = 0.0926$

$\frac{\sum_1^3(\gamma_{mn}-\gamma_{mean})^2}{\chi^2_{(\frac{0.05}{2},3)}} = \frac{0.1^2+0.1^2}{9.348} = 0.0021$

As a result, $deQAM$ determines the uncertainty variance $\hat{\sigma}^2_{\xi(V_{\varphi_{s_5}})}$ of the generated monitoring result $V_{\varphi_{s_5}}$ of inaccessible parameter φ_{s_5} is 0.0067 and the 95% confidence level is derived as the interval of $(0.0021, 0.0926)$.

V. Discussion

Our case study shows that $deQAM$ can successfully infer the monitoring result by using the dependencies across Cloud services. The study also shows that $deQAM$ can evaluate the reliability of the monitoring result by giving a 95% confidence level of the uncertainty estimation. The variance of uncertainty is as small as 0.0067. This implies that the monitoring result is unlikely to unexpectedly deviate from the true value of $V_{\varphi_{s_5}}$. Meanwhile, the 95% confidence level is based on the interval of $(0.0021, 0.0926)$. This narrow confidence interval constrains the randomness of the uncertainty variance. The narrow confidence interval disables the potential big deviation of the real uncertainty variance. As the uncertainty variance 0.0067 falls within the confidence interval, the reliability of the monitoring result is statistically sound.

Our case study also utilizes different types of parameters commonly existing in Cloud services. For instance, the parameter with a varying range represents the service parameter with variable status like CPU usage rate as in IaaS. Similarly, the parameter selected from a set represents the service parameters with several possible levels like different encryption levels of Cloud encryption services. Moreover, the parameter with boolean value represents the service parameter with the switch-effect like authentication parameters deciding the access to Cloud services. Therefore, the case study demonstrates $deQAM$'s capability of handling most of the common service parameter types used in the Cloud.

VI. Related Work

The indirect monitoring for Cloud service is a novel research topic that is only beginning to garner attention. Thus, some works that likely have potential value for design the indirect monitoring approach are surveyed below.

One related field is about service dependency though most works in this field focus on describing the notion of "dependency" from varied perspectives. For example, Winkler et al. [19] classified the service dependencies into discrete types for creating the service dependency model. Eppinger et al. [15] presented the design structure matrix (DSM) for refining the service dependency. Likewise, Qi et al. [16] proposed a dependency graph to analyze the services by trust. However, Garvey et al. [4][21] proposed a functional dependency network analysis framework (FDNA) by studying the characteristics of service dependency. Similarly, Guariniello et al. [1][5][9] proposed the system behavior modeling methodologies by improving the FNDA.

Another related field is about uncertainty, as the indirect monitoring approaches have to address the reliability issue by assessing the uncertainty of the generated monitoring results. For example, Wang et al. [7] proposed an adapted backward Cloud generator algorithm to address the uncertainty of service quality. Tang et al. [8] proposed a Bayesian based methodology to address the prediction uncertainty of service level agreement violations in Cloud. Furthermore, Huynh et al. [12] proposed a state-based policy framework to assess the monitoring quality caused the uncertain. Moreover, Galland et al. [20] proposed the different data value estimation algorithms to address the

uncertainty. Additionally, Li et al. [6] proposed a weighting optimization algorithm to address the uncertainty concern by discriminatingly assigning different weights to data sources. As contemporary work, Yin et al. [18] proposed a semi-supervised learning method to reduce the uncertainty.

VII. Conclusion

Indirect monitoring for Cloud services is an emerging challenge introduced by the inaccessibility of multiple Cloud service parameters. However, the existing monitoring approaches typically are inapplicable without direct access to those service parameters. Therefore, we propose $deQAM$ as a dependency-based quantitative aggregation indirect-monitoring methodology to specifically address this gap.

$deQAM$ introduces the *utility* to correlate the inaccessible parameter of the dependent Cloud service with the related accessible parameters of antecedent Cloud services by using the service dependency. Furthermore, $deQAM$ also adopts a multi-step utility computing procedure to infer the inaccessible parameter's value as the indirect monitoring result. Additionally, $deQAM$ evaluates the reliability of the monitoring result by specifying its confidence level.

The advantages of our approach are the capability of monitoring the inaccessible Cloud service parameters and the adaptability of working in many other indirectly monitoring scenarios. Currently, $deQAM$ can deal with the static indirect monitoring case. In the future, we plan to improve $deQAM$ for dealing with the dynamic monitoring scenario, in which the value of relevant parameters change over time. Overall, our paper provides a valuable starting point for exploring indirect monitoring in Cloud.

Acknowledgment

Research supported in part by EC H2020 ESCUDO-CLOUD GA #644579 and CIPSEC GA #700378.

References

[1] C. Guariniello and D. DeLaurentis, Supporting design via the System Operational Dependency Analysis methodology. Research in Engineering Design, pp. 1–17. Springer (2016)

[2] B. Li, et al., CloudMon: a resource-efficient IaaS cloud monitoring system based on networked intrusion detection system virtual appliances. Concurrency and Computation: Practice and Experience, vol. 27, pp. 1861–1885, Wiley (2015)

[3] F. Fittkau and W. Hasselbring, Elastic application-level monitoring for large software landscapes in the cloud. ESOCC 2015, pp. 80–94, Springer (2015)

[4] P. R. Garvey, et al., Modelling and measuring the operability of interdependent systems and systems of systems: advances in methods and applications. IJSSE 2014, vol. 5, pp. 1–24, Inderscience Publishers Ltd (2014)

[5] C. Guariniello and D. DeLaurentis, Communications, information, and cyber security in systems-of-systems: Assessing the impact of attacks through interdependency analysis. Procedia Computer Science, vol. 28, pp. 720–727, Elsevier (2014)

[6] Q. Li, et al., A confidence-aware approach for truth discovery on long-tail data. PVLDB, vol. 8, pp. 425–436, VLDB Endowment (2014)

[7] S. Wang, et al., Towards an accurate evaluation of quality of cloud service in service-oriented cloud computing. Journal of Intelligent Manufacturing, vol. 25, pp. 283–291, Springer (2014)

[8] B. Tang and M.Tang, Bayesian Model-Based Prediction of Service Level Agreement Violations for Cloud Services. TASE 2014, pp. 170–176 (2014)

[9] C. Guariniello and D. DeLaurentis, Dependency analysis of system-of-systems operational and development networks. Procedia Computer Science, vol. 16, pp. 265–274, Elsevier (2013)

[10] S. Meng and L. Liu, Enhanced monitoring-as-a-service for effective cloud management. IEEE Transactions on Computers, vol. 62, pp. 1705–1720, IEEE (2013)

[11] S. Meng, et al., Reliable state monitoring in cloud datacenters. CLOUD 2012, pp. 951–958, IEEE (2012)

[12] K. T. Huynh, et al., Maintenance decision-making for systems operating under indirect condition monitoring: value of online information and impact of measurement uncertainty. IEEE Transactions on Reliability, vol. 61, pp. 410–425, IEEE (2012)

[13] L. Schmetterer, Introduction to mathematical statistics. vol. 202, Springer Science & Business Media (2012)

[14] P. Leitner, et al., Application-level performance monitoring of cloud services based on the complex event processing paradigm. SOCA 2012, pp. 1–8, IEEE, (2012)

[15] S. D. Eppinger and T. R. Browning, Design structure matrix methods and applications. MIT press (2012)

[16] S. Qi, et al., A trust impact analysis model for composite service evolution. APSEC 2012, vol. 1, pp. 73–78, IEEE (2012)

[17] H. Koziolek, Sustainability evaluation of software architectures: a systematic review. SIGSOFT 2011, pp. 3–12, ACM (2011)

[18] X. Yin and W. Tan, Semi-supervised truth discovery. WWW 2011, pp. 217–226, ACM (2011)

[19] M. Winkler, et al., Analysing Dependencies in Service Compositions. ICSOC/ServiceWave 2009, pp. 123–133, Springer Berlin Heidelberg (2009)

[20] A. Galland, et al., Corroborating information from disagreeing views. WSDM 2010, pp. 131–140, ACM (2010)

[21] P. R. Garvey and A. Pinto, Introduction to functional dependency network analysis. ISSE 2009, Cambridge (2009)

[22] T. Zimmermann and N. Nagappan, Predicting defects using network analysis on dependency graphs. ICSE 2008, pp. 531–540, ACM (2008)

[23] J. Zhou, et al., Dependency-aware Service Oriented Architecture and Service Composition. ICWS 2007, pp. 1146–1149, IEEE (2007)

[24] D. A. DeLaurentis and D. N. Mavris, Uncertainty modeling and management in multidisciplinary analysis and synthesis. ASM 2000, AIAA (2000)